KANDIDATS:
The Porting of an Image Processing System

Linda J. Lallement
B.S., Kansas State University
1975

A MASTER'S REPORT

submitted in partial fulfillment of the
requirements for the degree
MASTER OF SCIENCE
Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas
1978

Approved by:

Linda Shapiro
Major Professor

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENT

I would like to thank Dr. Linda Shapiro, my major professor, for suggesting the topic of this report and her assistance in its writing and final preparation. My thanks also to Dr. William Hankley and Dr. Kenneth Conroe for their advice on this report. I would like to thank Robert Young, Amrendra Singh, and Dr. Robert M. Haralick for their technical assistance. A special thanks goes to Barbara North and my parents for their encouragement.

1. INTRODUCTION

1.1 KANDIDATS

KANDIDATS (KANsas Digital Image DAta System) is an interactive image processing package [1]. Through the use of KANDIDATS commands the user can create, manipulate, and display multiple digitized images. The package also allows the user to maintain information about an image in a standarized format. In general, KANDIDATS eases the interface between the user and the machine.

KANDIDATS was developed by the Image Processing Group of the Remote Sensing Laboratory, University of Kansas. It was developed in 1976 on a PDP-15 computer and is coded in FORTRAN. Images are obtained from and displayed on the IDECS, a versatile analogue display device, or they may be obtained from magnetic tape units on an IBM 7094 machine.

KANDIDATS was ported in part to the IBM-370 at Kansas State University. On the 370, KANDIDATS executes under the Conversational Monitoring System (CMS).

This report describes the KANDIDATS system as it exists at Kansas State University. Chapter 2 gives a general description of the system. Chapter 3 describes the changes made to the system in transporting it to the IBM-370. The KSU KANDIDATS User's Guide in Chapter 4 is a supplement to the KANDIDATS User's Guide at the University of Kansas. Chapter 5 is also a supplement to the User's Guide at the University of Kansas. Along with Chapter 3, it provides the information necessary for adding to or altering the KANDIDATS system. A brief example of the use of KANDIDATS at KSU is given in Chapter 6. Finally, Chapter 7 summarizes the report.

1.2 Extent of Project

The version of KANDIDATS developed on the PDP-15 consists of three major parts: KAND01 (image utility functions), spatial clustering, and pattern discrimination. Of these three parts, only KAND01 was ported to the IBM-370 and that only in part. The main driver, I/O routines, and command processing routines of KAND01 were altered to execute on the 370. The routines that interface with the operating system were written for the 370. The routines necessary to implement the commands listed below were ported to the 370. One new command was added to KANDIDATS. DISPLY was a command added to display images on the COMPUTEK (a visual display device).

KAND01 Commands	EXSIF Commands	
BRIEF	A	MID
EXPL	B	MOD
EXSIF	BLK	NEXT
DISPLY	BRIEF	OPEN
LONG	CIMG	OUT
MESG	CLOS	PROT
STOP	COL	REPL
SVDC	DONE	ROW
TID	FIND	SVDC
VOCA	FORM	TOP
DONE	IMG	VOCA
DLETE	INFO	

In general, four types of changes had to be made to KANDIDATS to transport KANDIDATS from the PDP-15 to the IBM-370. First, any syntactical differences between the versions of FORTRAN on the two machines had to be dealt with. Secondly, the routines that interfaced between KANDIDATS and the operating system had to be written for the 370. Because some functional capabilities are available on the PDP-15 that are not available

on the 370, some structural changes were necessary in the operating system interfaces. Next, differences in the word length on the two machines necessitated some changes. Finally, an additional module, DISPLY, was added to KANDIDATS.

Of the 72 routines that are part of KAND01 as it currently exists on the IBM-370, 46 routines were altered. Twenty-two routines had to be written in either FORTRAN or assembler for the 370. Four routines already written for the 370 were also used. Below is a list of these routines:

ADATE	WF	GDSC20	A	REPL	A
AND	WA	GETBLK	A	RFORMT	WF
BDEX1	A	GETCOR	A	RREAD	A
BYTEFT	WA	GETDN	A	RWRITE	A
BYTEST	WA	GETFLG	A	SCAN2	A
COMIN2	A	GETFN	A	SVOC	A
CORE	E	GETNM	A	TID	A
DATE	E	GETOKE	A	TIME	E
DEFDA	WA	HDOT	WF	TTYID	D
DEFILE	WA	HOME	WF	UNPACK	A
DEFINE	WF	IBFCNT	A	VOCA	A
DEVCHK	A	IBITNM	A	YESNO	A
DISPLY	WF	IBVOCA	A		
DLETE	WA	ICEIL	A		
DOTTER	WF	IFORMT	WF		
ENCODE	WF	INUNIT	WF	WF - routine was written in FORTRAN for the 370	
ERRPRC	A	JSRT2D	A		
EXMOD	AC	KDPUSH	A		
EXNEW	AC	LOG	A	WA - routine was written in Assembler for the 370	
EXPL	A	MDOT	WF		
EXPL1	A	MESG	A		
EXPRT	AC	MODE	A	A - routine was altered for the 370	
EXTSGN	WA	NRNC	A		
FDOT	WF	NUMBAR	A		
FETCH	A	PACK	A	E - routine was an existing 370 routine	
FILCHK	A	PDOT	WF		
FIND	A	RBKLS	A		
FLUSH	A	RCBKL	A		
FORM	WF	RDKINL	A		
FSTAT	WA	RDSC20	A		

2. STRUCTURE OF KANDIDATS

2.1 Data Structures

IDENT Array

The IDENT array is the first record of any image used by KANDIDATS. It contains the information necessary to process the image. The IDENT array is twenty integer words of which only the first nineteen are used. Twenty words is then the minimum record size for an image. The array contains the following values: [1]

IDENT(1)	Left coordinate of the image on the screen.
IDENT(2)	Right coordinate of the image on the screen.
IDENT(3)	Top coordinate of the image on the screen.
IDENT(4)	Bottom coordinate of the image on the screen.
IDENT(5)	Number of bits used to represent one point of the image.
IDENT(6)	Number of points per row in the image.
IDENT(7)	Number of rows in the image.
IDENT(8)	Relative size of the point in the horizontal dimension.
IDENT(9)	Relative size of the point in the vertical dimension.
IDENT(10)	Number of descriptor records in the file.
IDENT(11)	Number of discrete levels from the minimum to the maximum gray tone.
IDENT(12)	Number of machine words per logical record in the image.
IDENT(13)	Number of columns of points per subimage or logical record.

IDENT(14) Number of rows of points per subimage or logical record.

IDENT(15) Minimum gray tone of all images on the file.

IDENT(16) Maximum gray tone of all images on the file.

IDENT(17) Total number of images on a file.

IDENT(18) Number of symbolic images on the file.

IDENT(19) Data mode code:

= 0 positive integer
= 1 negative and positive integers
= 2 real (floating point)
= 3 half integer (integer *2)
= 4 double precision

IDENT(20) Unused at present.

SIF - Standard Image Format

Images on KANDIDATS are of a standardized form called SIF (Standard Image Format) [1]. A SIF file is a file of fixed length records that contain multiple digitized images and the information necessary to process those images. The images may be one of five data modes: integer (positive/negative or strictly positive), real, half integer, or double precision.

The first record of a SIF is the IDENT record described previously. The IDENT array is followed by N descriptor records. The value, N, is the tenth element of the IDENT array. The descriptor records are followed by the image data. There may be more than one image on a file.

The descriptor records of a SIF file are optional and provide image processing history and other general image information. The routines that process descriptor records were not implemented on the IBM-370. Therefore, only a brief description of the descriptor records

is given here. One type of descriptor record is the processing history records. These records maintain information as to what routines have been used to process the image data, the date, and any input files or parameters. The processing history records have nine different fixed record types. Besides the processing history descriptor records are free format descriptor records, which may be mixed with the processing history records. The free format records may contain information as to the ground characteristics of the area represented by an image.

Some terms that will be used in this paper to describe the image data of a SIF file are: pixel, subimage, and image. A pixel or point is the smallest part of an image or picture. It is represented by a single numeric value. Pixels are combined together in rows and columns to form an image. A particular point can be referenced by giving the image number, row, and column numbers. An image may be divided up into rectangular blocks of points called subimages. The subimages of an image are non-overlapping.

An image of N x M points might appear as in Figure 2.1., where N, the number of rows of points in the image, is determined by IDENT(7) and M, the number of columns of points in the image, is determined by IDENT(6). Note that each box represents one data point.

Figure 2.1 might also be used to represent a subimage which was made of N x M points. In the case of a subimage, N is determined from IDENT(14), the number of rows of points in the subimage. M is determined from IDENT(13), the number of columns of points in the subimage.

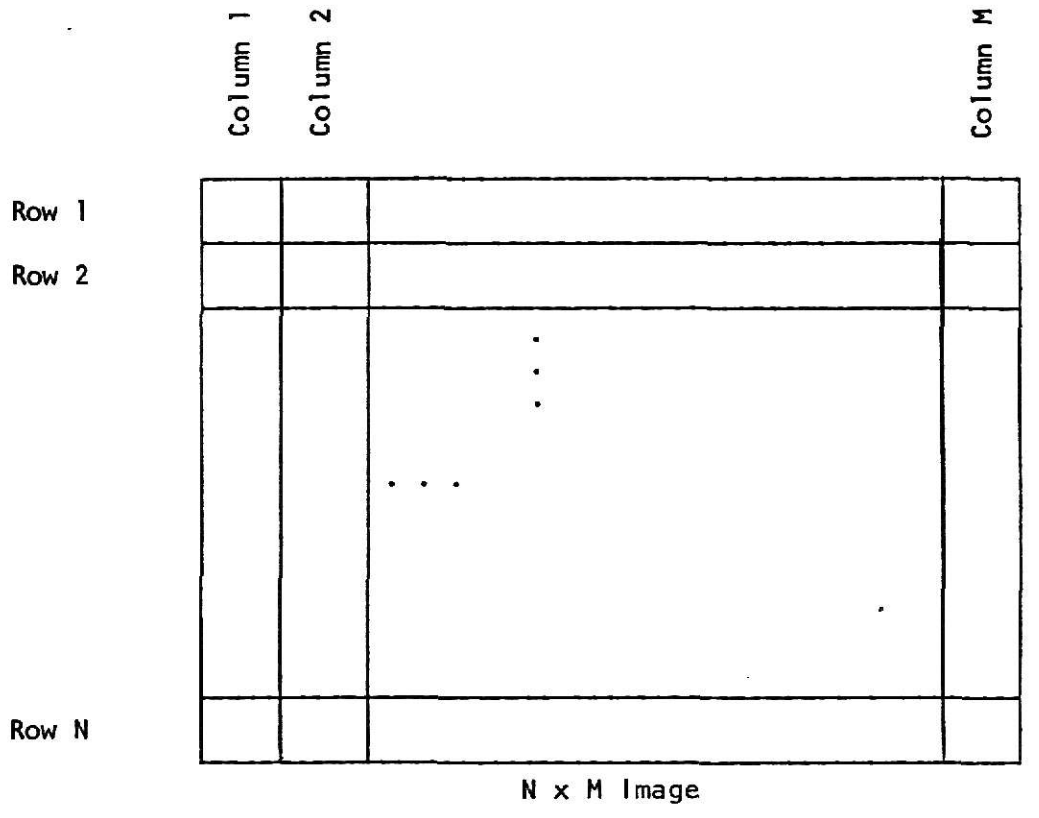
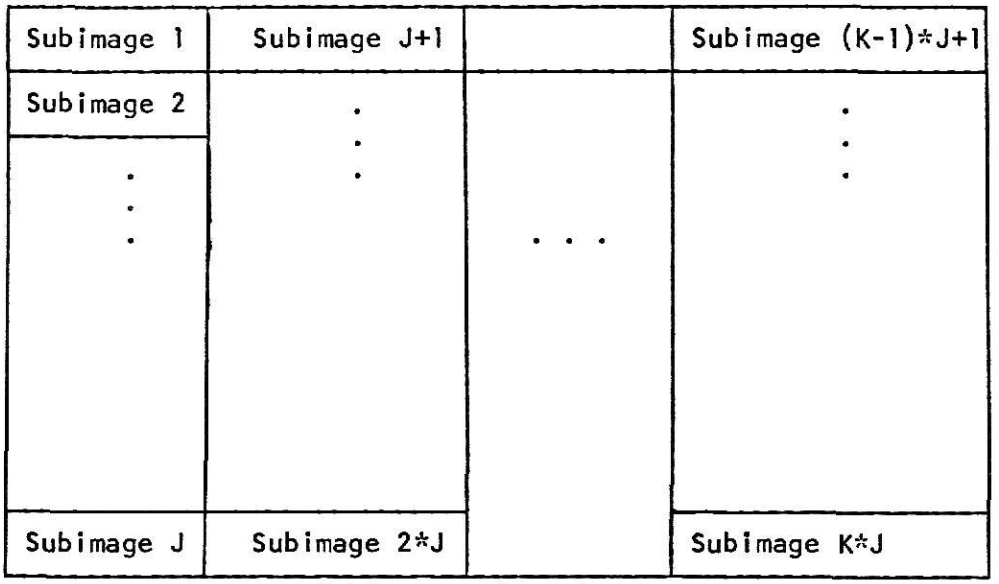


Figure 2.1

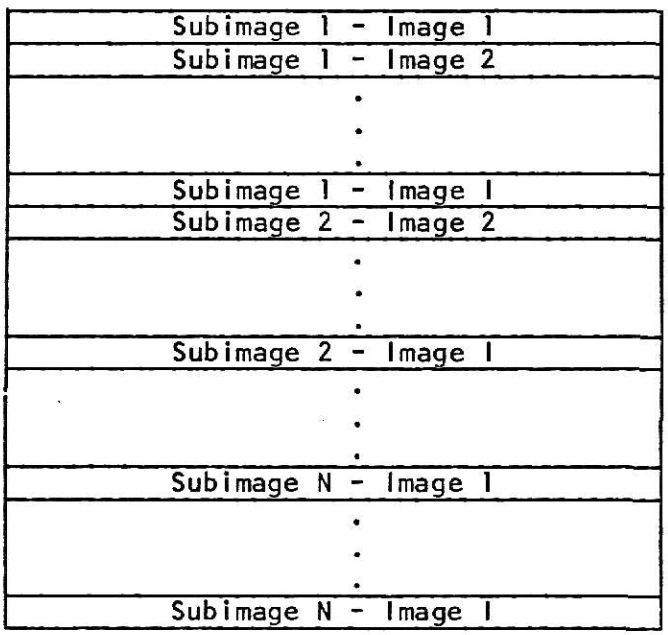
Figure 2.2 shows how subimages are arranged to cover an image. The value of J, the number of rows of subimages, and the value of K, the number of columns of subimages are determined from the number of rows/columns per image and the number of rows/columns per subimage.

Finally, Figure 2.3 shows how images and subimages are arranged on a multi-image SIF file. I (IDENT(17)) is the number of images on the file. N is the number of subimages in each image.



Subimages of an Image

Figure 2.2



SIF File

Figure 2.3

Note that in Figure 2.3 each line represents a logical record.

Images may be stored in one of five data modes. On the 370 these data modes are: positive integer, positive/negative integer, real, half integer (integer *2), and double precision real. The data mode of an image indicates the data type of the pixels of that image. The data mode also determines the number of words necessary to represent a data point for an image. For example, if an image is of data mode integer, one word is needed to represent a pixel, whereas if an image is double precision, two words are needed.

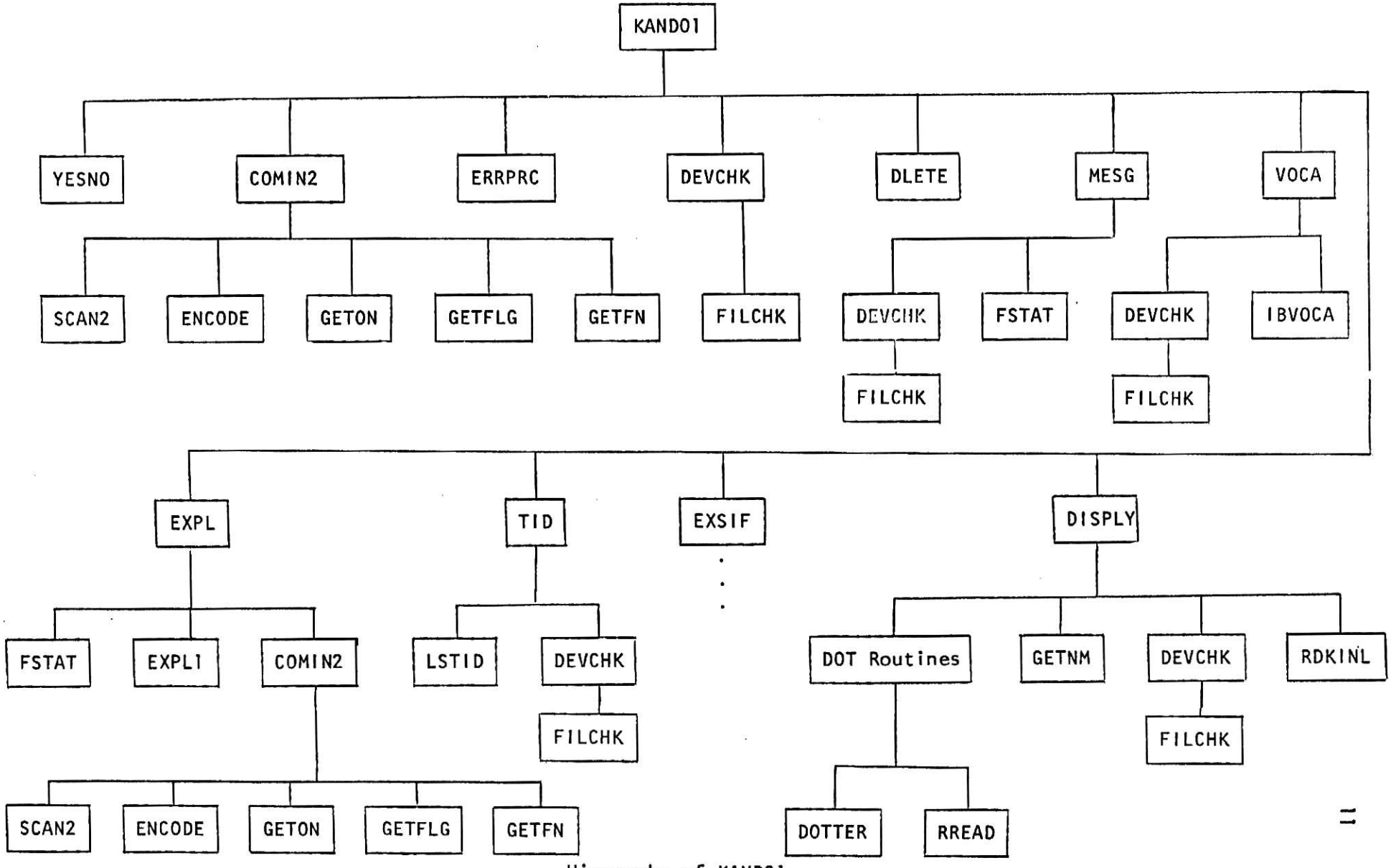
2.2 Hierarchy of Modules

The main driver of KANDIDATS is the routine KAND01. KAND01 calls the routines necessary to prompt the user for command input and parse the resulting command. It then calls the routines necessary to execute the command. There are two block data routines, IBFCNT and BDEX1, which initialize values of common block variables used by KAND01 and its subroutines.

Each of the commands has a driver which may or may not call other routines to execute functions of that command. One command, EXSIF, acts as the driver for a subsystem of KAND01. Like KAND01, EXSIF calls the necessary routines to prompt the user for a subsystem command, parse and validate the command and call the necessary routine to execute the command.

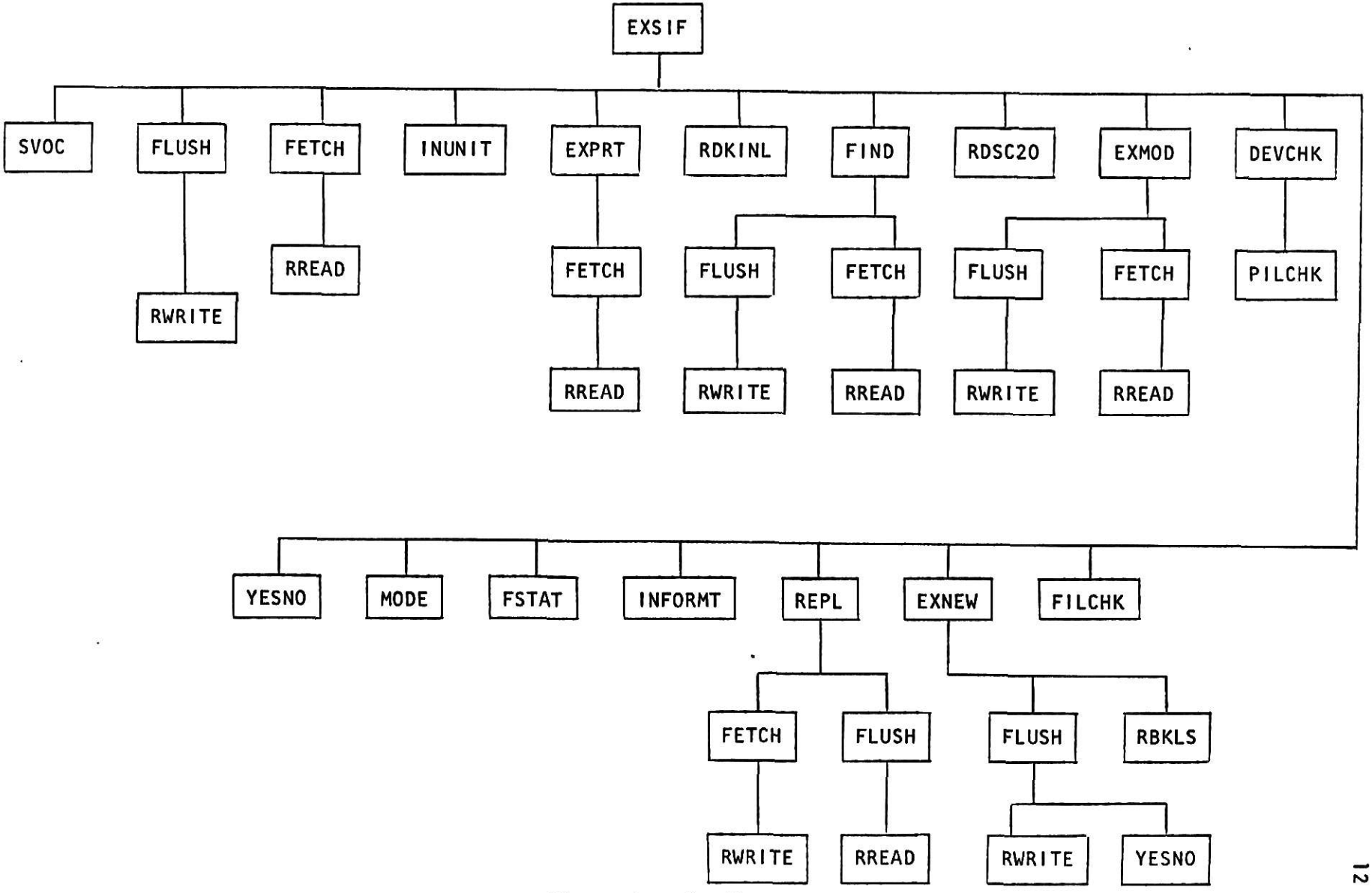
Finally, there are several routines that are common to a large number of the routines of KAND01. These routines are largely I/O routines. The routines which parse commands and check the valid devices for the commands are also shared among the routines.

Figure 2.4 shows the hierarchy of the modules comprising KAND01. Note that the routines called by EXSIF are shown in Figure 2.5. Figure 2.4 does not show all the modules called by KAND01. It does show the main routines implemented on the IBM-370. The same is true for Figure 2.5.



Hierarchy of KANDO1

Figure 2.4



Hierarchy of EXSIF

Figure 2.5

3. CHANGES

3.1 Syntax Changes

Differences between the FORTRAN used on the PDP-15 machine and that used on the IBM-370 necessitated some changes in the syntax of KANDIDATS. These changes are described below.

Alternate Return

The first syntactic change involved the use of the alternate return in a subroutine call. The PDP-15 system allowed the use of variables in the parameter list and after the RETURN statement to control the return from a subroutine. A subroutine might return control to the calling routine in different locations based upon some conditions established within the subroutine.

For example, upon successful completion of its objective, the subroutine would return control to the statement following its call. Error conditions would return control to other locations in the calling routine where the errors would be dealt with. The subroutine would be coded as follows:

```
      SUBROUTINE EXONE(ERR1,ERR2)

      successful completion
      RETURN

      first error condition
      RETURN ERR1

      second error condition
      RETURN ERR2
```

The calling routine would be structured as follows:

```
      CALL EXONE (@1000,@2000)

 1000 CONTINUE
      first error processing

 2000 CONTINUE
      second error processing
```

The action of this code is shown in Figure 3.1.

To alter these functions so that they are compatible with IBM FORTRAN, the character '@' in the call must be changed to a '&'. The call then becomes:

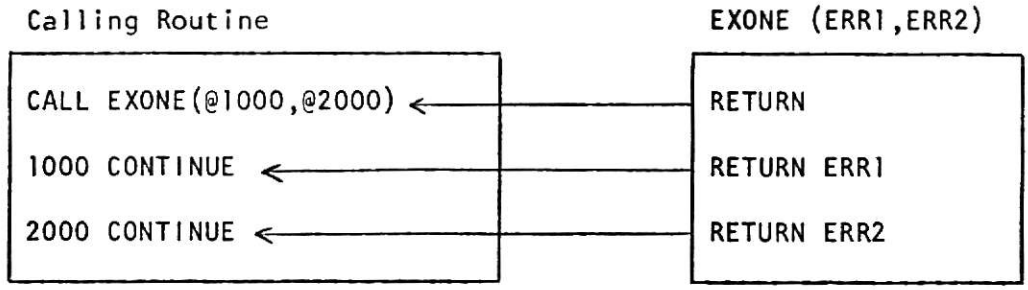
```
      CALL EXONE(&1000,&2000)
```

The alternate return variables in the subroutine's formal parameter list are replaced with '*'s.

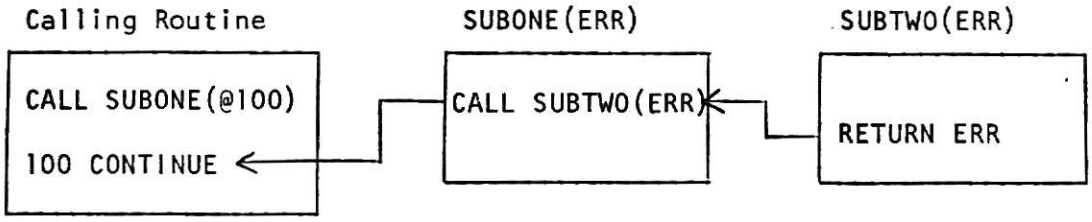
```
      SUBROUTINE EXONE(*,*)
```

Finally, the RETURN variable is replaced with a digit indicating which '*' and corresponding statement label to return control to. RETURN ERR1 becomes RETURN 1 and returns control to statement 1000 in the calling routine. Similarly, RETURN ERR2 becomes RETURN 2, returning control to statement 2000.

In the PDP-15 system, the alternate returns may also be nested as in Figure 3.2. The error return from SUBTWO causes an immediate return from SUBONE to the calling routine. To obtain the same action from IBM FORTRAN, a RETURN statement must be added to SUBONE. The result is shown in Figure 3.3.

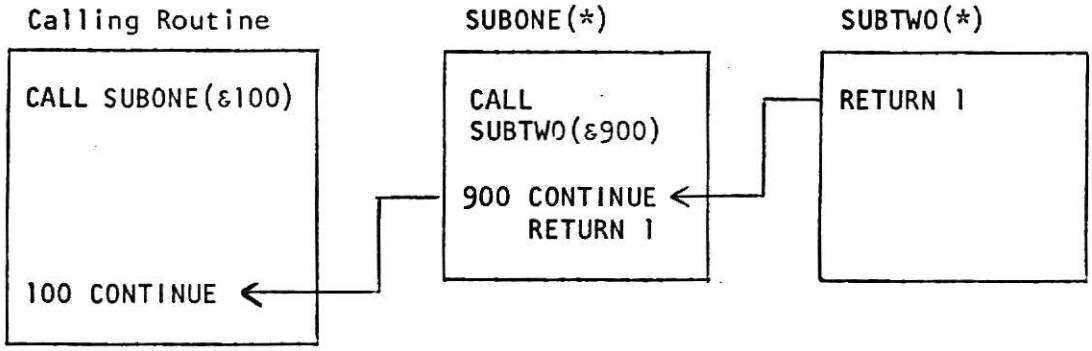


Alternate Error Return
Figure 3.1



Nested Alternate Error Returns

Figure 3.2



Change for Nested Alternate Error Returns

Figure 3.3

Call By Address

Another change for the 370 in the formal parameters involved enclosing each of the parameters in slashes. This caused the parameters to be passed by address rather than by value-return. Passing by address was necessary since arrays of varying dimension and type were often passed from the invoking routine to several levels of subroutines, and previous experience had shown that the IBM value-return passage could cause correct PDP-15 programs to run incorrectly. The statement CALL EXTWO (PARAM1, PARAM2, etc. . . .) would be changed to CALL EXTWO (/PARAM1/, /PARAM2/, etc. . . .).

Length of String Constants

Among the changes resulting from the difference in word lengths on the PDP-15 and IBM-370 machines is the change in the lengths of character string constants. In the PDP-15 KANDIDATS, strings were usually assigned to arrays in DATA statements. For example, a variable of type DOUBLE INTEGER on the PDP-15 machine is 36 bits and can hold 5 7-bit characters. On the IBM machine, an INTEGER variable is 32 bits and can hold 4 8-bit characters. The statements:

```
DOUBLE INTEGER FNAME(2)
DATA FNAME/'FUNCN','AME'/
```

would be changed to

```
INTEGER FNAME(2)
DATA FNAME/'FUNC','NAME'/
```

The change in the number of characters that can be associated with an integer necessitated changes in FORMAT statements. For example, the

format statement for writing out FNAME as described above would be changed from (2A5) to (2A4).

Because of the changes in the length of the data types, the length of the function names, file names, and device names all changed. This caused the routines that scan the input commands to change. For example, the routine GETFN which scans for a function name did the following:

```
C       NO, IS SYMBOL LENGTH OK

        IF (IL .GT. 6) GO TO 180
C       BLANK OUT TEMP. STORAGE
        DO 199 I = 1,9
  199   T(I) = BLANK

C       MOVE SYMBOL TO TEMP. STORAGE
        DO 170 I = 1,IL
  170   T(I) = LINE(I - 1 + IB)
```

where IL is the length of the string found during the scan and T is a double integer array of nine elements. This code was altered for the 370 as follows:

```
C       NO, IS SYMBOL LENGTH OK

        IF (IL .GT. 8) GO TO 180
C       BLANK OUT TEMP. STORAGE
        DO 199 I = 1,8
  199   T(I) = BLANK

C       MOVE SYMBOL TO TEMP. STORAGE
        DO 170 I = 1,IL
  170   T(I) = LINE(I - 1 + IB)
```

where T is an integer array of eight elements. The original T had nine elements for the six characters of the function name and the three character extension. On the IBM machine T has eight elements for the eight characters of the function name.

Invalid Use of String Constants

The PDP version of KANDIDATS uses character string constants in FORTRAN statements other than the DATA statement. The IBM FORTRAN allows string constants in DATA statements or FORMAT statements. Therefore, routines using string constants in the body of the routine had to be altered. Variables were created and assigned the string constant values in a DATA statement.

The PDP statements:

```
IF(FUNC .EQ. 'QUIT') GO TO 900
NAME = 'ØØØØØ'
```

would be changed to the following on the IBM-370:

```
INTEGER QUIT, BLANK
DATA QUIT/'QUIT'/,BLANK/'ØØØØ'/

IF(FUNC .EQ. QUIT) GO TO 900
NAME = BLANK
```

Decrementing DO-LOOP

One difference between the instruction set of the PDP-15 and the IBM-370 FORTRANs involved a decrementing do-loop. The PDP code contained loops with negative modifications of the loop index. Since IBM FORTRAN does not allow this, the loop had to be altered.

For example, in the routine EXPL1 there is a loop which searches backward from the end of a card image for the first non-blank word. Then the card image is printed out up to that last word.

```
      DO 38 I = 16,1,-1
      IF(CARD(I) .NE. BLANK) GO TO 39
38    CONTINUE
      I = 1
39    WRITE(OT,40) (CARD(J), J = 1,I)
      .
      .
      .
```

Taking into consideration that because of a change in data type a card on the IBM-370 consists of twenty words instead of sixteen as on the PDP, the code above can be rewritten using an incrementing loop as follows:

```
      DO 38 I = 1,20
      K = 21-I
      IF(CARD(K) .NE. BLANK) GO TO 39
38    CONTINUE
      K = 1
39    WRITE(OT,40) (CARD(J), J = 1,K)
      .
      .
      .
```

Octal to Hex Constants

The PDP FORTRAN allows octal constants to be assigned to variables in DATA statements. The syntax for these constants is the character '#' followed by octal digits. For the 370, these constants needed to be changed to hexadecimal constants. The syntax for the hexadecimal constant is a 'Z' followed by hexadecimal digits. Additional changes were necessary because of the change from ASCII characters on the PDP-15 to EBCDIC characters on the 370. These changes are described further under the section Prompting and Position Control Characters.

3.2 Machine Related Changes

Prompting and Position Control Character

KANDIDATS on the PDP-15 machine uses octal constants to assign prompting and carriage control characters to variables. The octal representation of the ASCII characters are used to control the output line position on the terminal and the line printer. The IBM-370 uses EBCDIC characters and they must be represented in hexadecimal notation. The same sort of change was necessary for the characters used as prompts.

For example, the bell character followed by the rubout character in ASCII would be #037760. The rubout character was not used on the 370, but the hexadecimal representation of the EBCDIC bell character is Z2F000000.

I/O Devices

The difference in the peripheral units between the two machines necessitated some major changes in KANDIDATS. On the PDP-15 machine there are multiple disk packs, IDECS channels, line printer channels, tape units and a single teletype unit. The 370 does not have the IDECS. Under CMS on the 370, the tapes and additional disk units are not easily accessible from an executing program. The printer is a virtual printer. Any output to the printer from KANDIDATS on the 370 is written to a disk file. This disk file is placed in the user's virtual reader when KANDIDATS terminates. On the 370, a virtual punch can be used as an additional input/output device.

This change in the I/O devices had its largest effect on the module DEVCHK. The word used to indicate the valid devices for a given command, DEVMSK, was changed as shown below.

Bit	Destination	Source
0	Unused	
1	None	None
2	Disk	None
3	Disk	Disk
4	Disk	Punch
5	Disk	Terminal
6	Unused	
7	Punch	None
8	Punch	Disk
9	Punch	Punch
10	Punch	Terminal
11	Unused	
12	Terminal	None
13	Terminal	Disk
14	Terminal	Punch
15	Terminal	Terminal
16	Unused	
17	Printer	None
18	Printer	Disk
19	Printer	Punch
20	Printer	Terminal

The values assigned to three variables in the common block, IBFCNT, also were changed. The device mnenomics and type codes were changed to:

Mnemonic	Type	Code
DPA	Disk	1
DPB	Disk	(unused at this time)
PRTR	Printer	4
PUN	Punch	2
TERM	Terminal	3

These changes will have some impact on the user of KANDIDATS. First, the source and destination device mnemonics will change to those described above. Secondly, those commands accessing tapes will no longer be available.

Machine Dependent Constants

A number of the I/O routines of KANDIDATS contained calculations which were dependent upon the number of words per data type, the number of bytes per word, or the number of bits per word. Since these values can vary from one machine to the next, a common block, MACH, was created to contain variables whose values are dependent on the machine on which KANDIDATS is executing. Wherever a constant that was machine dependent was used in calculations, it was replaced with a variable from this common block. The values of these variables are set in the block data subroutine IBFCNT. This common block should facilitate the transporting of KANDIDATS to a third machine.

Below is a description of the common block, MACH, with the value of each variable on the PDP-15 and IBM-370 machines.

Variable	PDP-15	IBM-370	Description
NMBTPM	18	32	Number of bits per machine word
NWORDS			Number of words per given data type
	1	1	(1) - absolute integer
	1	1	(2) - integer
	2	1	(3) - real
	2	.5	(4) - double integer on PDP half integer on IBM
	3	2	(5) - double precision
NMWDIO	1	1	Number of machine words per IO word
NMWMIN	20	20	Number of machine words necessary to contain the IDENT array

Variable Types

The number of words and thus bytes per data type is different on the IBM-370 than on the PDP-15. The table below shows these differences.

	Number of Words		Number of Characters	
Type	PDP-15	IBM-370	PDP-15	IBM-370
Integer	1	1	Not used	4
Double integer	2	Not used	5	Not used
Half integer	Not used	.5	Not used	2
Real	2	1	5	4
Double Precision	3	2	Not used	8

A major change because of these differences in the data types was the elimination of double integer variables on the 370. All variables that were of type double integer in the original version of KANDIDATS had to be changed. For the most part these variables were changed to integer with some adjustment in the number of elements in arrays where necessary. For example, device names which had been 5 characters long became 4 characters long when changed from double integer to integer.

A side effect of changing the data type of some variables was the necessity of changing some common blocks. Since double precision variables must be aligned on double word boundaries in common blocks and integer and real variables on full word boundaries, the variables in a common block must be arranged in order from double word to full word. Thus when variables in a common block changed type, they had to be rearranged.

One final change in the data type of variables was the change in one of the allowable data types of an image. Again, since on the PDP-15 an image could be of type double integer, any reference to that type of

image had to be changed to half integer. This change was partially accomplished by the creation of the common block MACH as described in the previous section.

Routines that Interface with the Operating System

Because the operating system on the PDP-15 is different from that on the 370, any routines that interface with that operating system must be altered. These routines are, for the most part, the routines that manipulate files. These routines were often written in assembler and therefore had to be written for the 370 rather than just being altered.

The routine, CTRLT, on the original version of KANDIDATS was used to enable the user to interrupt the execution of a command and return control to KAND01, the main driver. On the IBM machine under CMS, the only means of interrupting the execution of a program is to strike the attention key. However, when the attention key is struck, control returns to CMS. Since the function of CTRLT could not be duplicated on the 370 without changes to the operating system, that routine was dummied out.

The routine DLETE locates a named file on disk and deletes that file. Its parameters are the unit number, the file name, and a code indicating whether the file was found or not. On the 370, the unit number is not used. DLETE, as it is written on the 370, behaves the same as the CMS ERASE command. It was implemented in assembler using the system macro instructions FSSTATE and FSERASE.

The routine ADATE returns the current data and time for use by KANDIDATS. It returns the data and time in the format MM/DD/YYØØØHH:MM.

ADATE is a FORTRAN routine which calls two system routines, DATE and TIME, and another FORTRAN routine, FORM. DATE returns the current date. TIME returns the time of day in hundredths of seconds. FORM arranges the time in the form described above. ADATE and FORM had to be written for use on the 370.

FSTAT is another assembler routine written for KANDIDATS on the 370. It verifies the existence of a named file on the disk. The PDP-15 version of KANDIDATS passes the unit number, the name of a file, and a variable indicating whether or not the file was found to FSTAT. The unit number is not used on the IBM version of FSTAT. The file name is set up for use by the expansion of the FSSTATE macro instruction. An additional function is performed by the IBM version of FSTAT. FSTAT retrieves the record size in bytes and the number of records for a given file and places those values in the common block FILSIZ for use in opening a file.

Under the original KANDIDATS when a file was opened, it was opened first for sequential access and the IDENT array was read in. The file was then closed and the information from the IDENT array was used to re-open the file for direct access. On the 370, however, a file once opened for sequential access could not be re-opened for direct access. To solve this problem a file is always opened for direct access. The record size and number of records used to open a file for direct access were obtained from the common block, FILSIZ, described above. This change in the opening of files eliminated the routines SEEK, ENTER, and CLOSE. SEEK opened an input file for sequential access and ENTER opened an output file for sequential access. CLOSE closed a sequential access file.

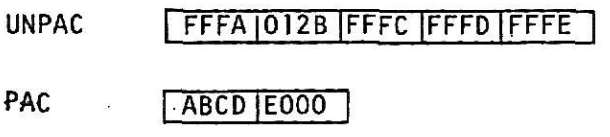
The routine DEFINE initializes a file for direct access input/output operations. On the 370, this involves a file definition, which associates a unit number with a file and a DEFINE FILE, which associates a variable with the file and indicates whether the file will be formatted or unformatted. DEFINE is a FORTRAN routine that calls two assembler routines: DEFILE and DEFDA.

DEFILE performs the functions of a FORTRAN DEFINE FILE statement. A DEFINE FILE statement requires constants for the values of unit number, number of records and record size. Since these values are dynamically determined on KANDIDATS, however, a routine had to be written that would use the values of variables to perform a DEFINE FILE. The routine written to do this is DEFILE.

DEFDA performs a dynamic file definition. The file definition is for a direct access file. DEFDA places the file name, unit number, and record size into a command string. This command string is passed as a parameter in a supervisor call macro.

PACK and UNPACK

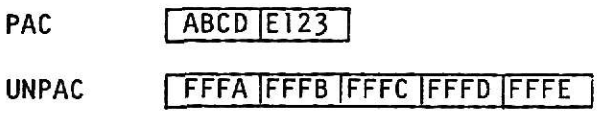
Two FORTRAN routines written to handle packing and unpacking bytes are PACK and UNPACK. PACK takes an array having bytes which are right justified in a word and packs the bytes in an array. For example, given an array UNPAC which contains the bytes right justified, they would be packed into the array PAC as shown below:



Example of Packing

Figure 3.4

UNPACK does just the opposite, as shown below:



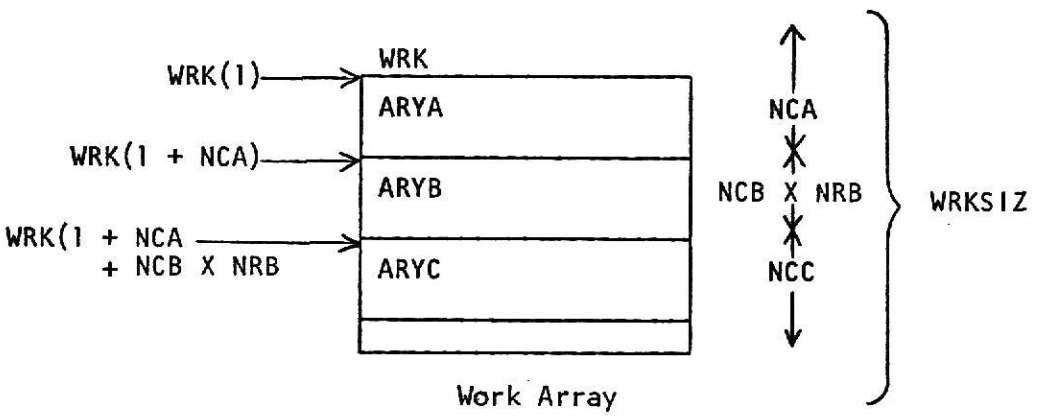
Example of Unpacking

Figure 3.5

These two routines are generalized so that the packing and unpacking may be done with any fixed number of bits rather than just eight-bit bytes.

Dynamic Dimensioning of Arrays

In KANDIDATS there are work arrays that need to be divided into several arrays. Consider Figure 3.6:



Work Array

Figure 3.6

The work array, WRK, is a one-dimensional array of size, WRKSIZ. This work array can be divided into three arrays: ARYA, ARYB, and ARYC. ARYA and ARYC might each be one-dimensional arrays with NCA and NCC elements, respectively. Assume that the array, ARYB, has NCB columns and NRB rows or NCB X NRB elements.

In the original version of KANDIDATS, three routines: ADJ1, ADJ2, and ADJ3, handle adjustable dimensioning. The above arrays would initially be dimensioned as follows:

```
DIMENSION WRK(512), ARYA(1), ARYB(1,1), ARYC(1)
DATA WRKSIZ/512/
```

The dimension of 1 acts as a place holder and establishes the dope vectors and dimensions for each of the arrays. Once the number of rows have been determined for each of the arrays, the dimensions of the arrays can be adjusted by doing the following:

```
CALL ADJ1(ARYA,WRK(1))
N = 1 + NCA
CALL ADJ2(ARYB,WRK(N), NRB)
N = N + NCB*NRB
CALL ADJ1(ARYC,WRK(N))
```

The effect of this code is the same as:

```
DIMENSION ARYA(NCA), ARYB(NRB,NCB), ARYC(NCC)
```

The routines ADJ1, ADJ2, and ADJ3 alter the dope vectors for arrays of one, two, and three dimensions respectively. The starting address of the array must be set and the number of elements in the various dimensions. To get the same adjustable dimensioning in the IBM version of KANDIDATS, the addresses of the arrays ARYA, ARYB, and ARYC must be passed to a subroutine along with the dimensions of the arrays.

There are two cases of array adjustment in the original version of KANDIDATS. Case one: the arrays are used in the same routine in which they are dimensioned. Case two: the arrays are used in a subroutine called by the routine in which the arrays are dimensioned.

To handle case one for the IBM version of KANDIDATS, a new subroutine must be introduced. The code that uses an array is removed from the routine where the array is adjusted to a new subroutine. For example, the original version of KANDIDATS might have had a routine with the following code:

```
      INTEGER WORK(512), ARY(1,1)
      .
      .
      .
      CALL ADJ2(ARY,WORK(N),NROWS)

      DO 10 I = 1,NROWS
        DO 20 J = 1,NCOLS
          X = X + ARY(I,J)
   20 CONTINUE
   30 CONTINUE
```

For the 370, the original routine would become:

```
      INTEGER WORK(512)
      .
      .
      .
      CALL NEW(WORK(N),NROWS,NCOLS,X)
```

The routine that is created, NEW, would be:

```
      SUBROUTINE NEW(/ARY/,/NROWS/,/NCOLS/,/X/)

      INTEGER ARY(NROWS,NCOLS)

      DO 10 I = 1,NROWS
        DO 20 J = 1,NCOLS
          X = X + ARY(I,J)
   20 CONTINUE
   10 CONTINUE
```

The starting address of the array ARY is established by passing the element of the work array, WORK, where ARY is to start. The dimensions of the array must also be passed to the subroutine.

Case two is handled in much the same way. In this case, however, the subroutine already exists. Consider the following example:

```
      INTEGER WORK(512), ARY(1,1)
      .
      .
      .
      CALL ADJ2(ARY,WORK(N),NROWS)
      .
      .
      .
      CALL SUB1(ARY)
```

For the 370, the calling routine would become:

```
      INTEGER WORK(512)
      .
      .
      .
      CALL SUB1(WORK(N),NROWS,NCOLS)
```

and SUB1 would become:

```
      SUBROUTINE SUB1(/ARY/,/NROWS/,/NCOLS/)
      INTEGER ARY(NROWS,NCOLS)
```

Note that the array ARY disappears completely from the calling routine and that extra parameters must be added for the dimensions of the arrays.

3.3 Structural Changes

Unit Number Assignment

On the PDP-15 a file could be assigned to a given unit number, deleted and a different file could be assigned to the same unit number. On the 370, however, once a file has been assigned to a unit number, no other file may be assigned to that same unit number. This necessitated a change in the structure of KANDIDATS.

The routine RDKINL is called to *initiate* a file for input/output. RDKINL was changed to call the routine, INUNIT, which maintains a 1-1 mapping of unit numbers to filenames. A common block, UNTS, was created which contains an array, UTAB. UTAB is a double precision array of five elements. UTAB is initialized to blanks. When a file is first initialized, the file name is placed in the first blank element of UTAB. The index of that element plus 10 is used as the unit number for the file. Thus the unit numbers for files range from 11 to 15. If a file is deleted, the name remains in the unit number table keeping that unit number from being assigned to any other file name. Note that this limits the number of files that can be used in any one KANDIDATS session to five. This limit can be increased by increasing the number of elements in the array UTAB. Any initializations of a file after the first will search the table UTAB for the file name and return the corresponding unit number.

Visual Display Routines

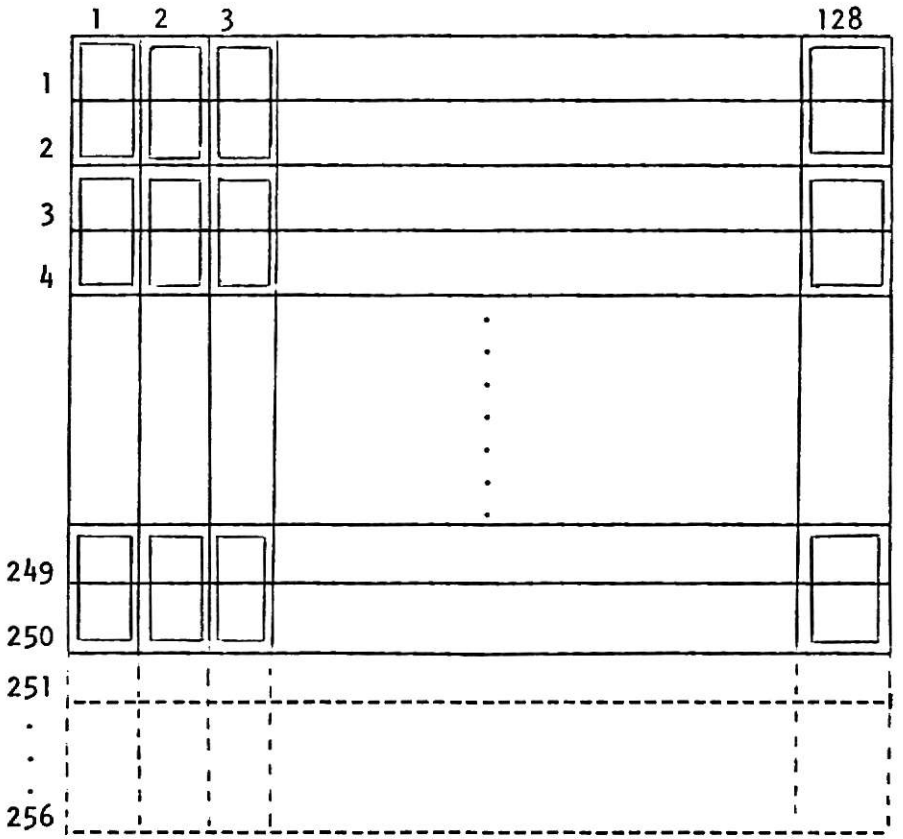
To allow the user to display images on the COMPUTEK from KANDIDATS, the command DISPLY was created. The general form of the command is:

DISPLY TERM <- DPA file name

This causes the named image to be displayed with five gray tone levels on the COMPUTEK screen.

DISPLY initializes the file for input/output. This includes determining the number of image points per window to divide the image into 128 windows per row and 128 windows per column. The number 128 comes from the 256 dots per row and column of the COMPUTEK screen divided by two. It takes a 2 by 2 window of dots to represent one point of the image. If the number of points per image is not evenly divisible by 128, the image is extended in either the number of rows or the number of columns such that it is evenly divisible.

Consider an image which is 128 points per row by 250 points per column. Since 250 is not divisible by 128, the image is extended by 6 rows of the lowest gray tone of the image. Figure 3.7 shows this extended area as a dashed line.



Display

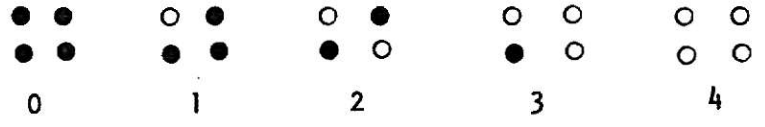
Figure 3.7

Note that each window is highlighted. In this image the windows are 2 rows by 1 column.

DISPLY calls a routine, based on the data type of the image, which reads in the image and averages the gray levels of the windows. The range of high to low gray tones is mapped onto the range 0 - 4. For example, if an image has gray tones ranging from 11 - 30, they would be mapped as follows:

Gray Tones	Level
11 - 14	4
15 - 18	3
19 - 22	2
23 - 26	1
27 - 30	0

At this point, each window of the image has a gray level between 0 and 4. The routine, DOTTER, is called to translate the gray level into a 2 by 2 pattern of dots. The translation is shown in Figure 3.8 where a dark point is represented by a solid dot and a light point by an open dot.



Dot Patterns

Figure 3.8

Subsystems

Subsystems on the original version of KANDIDATS were initiated through the call of a routine KCALL. On the 370, the call to subsystems are direct. The only subsystem currently implemented on the IBM-370 version of KANDIDATS is the module EXSIF.

4. KSU KANDIDATS USER'S GUIDE

4.1 Introduction

KANDIDATS (KANsas Digital Image DATa System) is an image processing package developed by the University of Kansas Center for Research, Inc. KANDIDATS was partially altered so that it will execute on the IBM-370 at Kansas State University. This document is intended to supplement the Kansas University KANDIDATS user guide, KANDIDATS II (Technical Report 0920-2, September 1976) [1] for the Kansas State user of KANDIDATS. It documents only those parts of KANDIDATS that were implemented on the IBM-370.

4.2 KANDIDATS Digital Images

Multi-Band Digital Images

KANDIDATS processes multi-band digital images. A digital image is a numeric representation of an image. If the two-dimensional image is divided into rows and columns, then each row and column number pair specifies a point of the image. The value at each point of the image is a number that indicates light intensity of that point.

Images on KANDIDATS are maintained in files of a standardized form. Files of this form are called Standard Image Format (SIF) files. SIF files have three types of records: the identification record, the descriptor records, and the image data records. On KANDIDATS as it is implemented at Kansas State, the descriptor records are unused.

The first record on a SIF file is the identification record (IDENT). This record is comprised of twenty integer words which provide the basic information necessary to process the file. The values of the array, IDENT, are:

IDENT(1) Left coordinate of the image on the screen.

IDENT(2) Right coordinate of the image on the screen.

IDENT(3) Top coordinate of the image on the screen.

IDENT(4) Bottom coordinate of the image on the screen.

IDENT(5) Number of bits used to represent one point of the image.

IDENT(6) Number of points per row in the image.

IDENT(7) Number of rows in the image.

IDENT(8) Relative size of the point in the horizontal dimension.

IDENT(9) Relative size of the point in the vertical dimension.

IDENT(10) Number of descriptor records in the file.

IDENT(11) Number of discrete levels from the minimum to the maximum gray tone.

IDENT(12) Number of machine words per logical record in the image.

IDENT(13) Number of columns of points per subimage or logical record.

IDENT(14) Number of rows of points per subimage or logical record.

IDENT(15) Minimum gray tone of all images on the file.

IDENT(16) Maximum gray tone of all images on the file.

IDENT(17) Total number of images on the file.

IDENT(18) Number of symbolic images on the file.

IDENT(19) Data mode code:

= 0 positive integer
= 1 negative and positive integers
= 2 real (floating point)
= 3 half integer (integer *2)
= 4 double precision

IDENT(20) Unused at present.

The descriptor records immediately follow the identification record. Although descriptor records may be present on a SIF file on the IBM-370, they cannot be processed by the current system. The descriptor records give additional information about a file. They provide statistical information, processing history, and ground characteristics of an image. The descriptor records are optional.

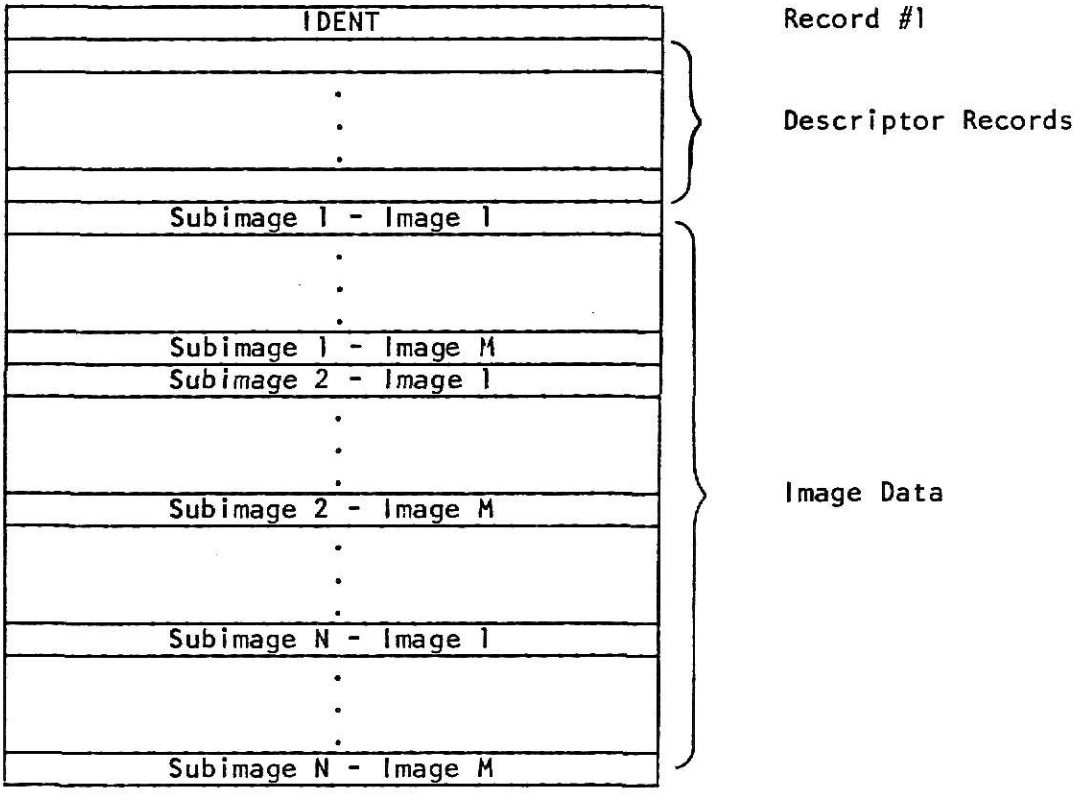
Following the descriptor records are the image data records. An image may be divided into non-overlapping rectangular regions called subimages. The subimages cover an image as shown in Figure 4.1, where the image has J rows of subimages and K columns of subimages. Each subimage would comprise a record of the image file.

Figure 2 shows the organization of the different records on the file. Note that the SIF file consists of M images. Each image has N subimages. The file has the first subimage of all images followed by the second subimage for all the images, etc.

Subimage 1	Subimage J+1	. . .	Subimage (K-1)J+1
Subimage 2	.		.
.	.		.
.	.		.
.			
Subimage J	Subimage 2*J		Subimage K*J

Subimage of an Image

Figure 4.1



SIF File

Figure 4.2

4.3 Command String Interpreter

The user interacts with KANDIDATS by entering commands that follow a standardized form. The form is:

```
#:
VERB DEST FILE1 (FLAGS) ← SOURC FILE2,FILE3 (FLG)
```

#:	are the prompting characters for KANDIDATS.
VERB	is the function to be executed.
DEST	is the optional device to which output is to be sent.
FILE1	is the name of the file on this device which receives the output (optional).
(FLAGS)	are letters of the alphabet in parenthesis which specify options for the given command.
< -	are the characters which delimit the source and destination halves of the command.
SOURC	is the optional device from which input is to be taken.
FILE2, FILE3	are the names of files from which input may be taken. They are delimited by a comma.
(FLG)	is the same as the first set of flags.

The function and file names may be up to eight characters long. The source and destination devices are one of the following:

Mnemonic	Device	Use
DPA	Disk	Source and destination
DPB	Disk	Source and destination
RDR	Virtual reader	Source
PRTR	Virtual printer	Destination
PUN	Virtual punch	Destination
TERM	Terminal	Source and destination

The KANDIDATS prompting characters are displayed at the user's terminal. On the line following the prompt, the user may enter up to 72 characters. There may be one and only one command per line.

Utility Subsystem EXSIF

If the user enters the EXSIF subsystem by entering the command EXSIF, the command form changes. The form of commands under the subsystem is as follows:

```
[A]
[B]
[VERB ]
[VERBA]   PARM1   PARM2
[VERBB]
```

A and B	are the prompting characters. If the current file is the A file, then A is the prompting character. If B is the current file, then B is the prompting character. If there is no current file, the prompting character is A.
VERB	is the function to be executed on the current file.
VERBA	is the function to be executed on file A.
VERBB	is the function to be executed on file B.
PARM1 PARM2	are values that may have to be specified for a given command.

Two files may be opened for use at one time by the EXSIF subsystem. These files are referred to as the A and B files. The prompt character indicates which file is currently being worked with. As with the KANDIDATS mainline, the function name may be up to eight characters. Some of the commands may have the character A or B attached to the end of the function name. The letter on the end of the function name indicates which file the

function is to be applied to. The parameters are dependent on the function. Some functions require no parameters, some one, and others two. For example, to change the mode in the IDENT array of the image PICEXP (which is assumed to be file A) to absolute integer, the following EXSIF commands would be entered:

```
MIDA     19,0
```

Optional Functions

In both the KANDIDATS main driver and the EXSIF subsystem, the command entered causes execution to be transferred to the appropriate subroutine to handle the processing of that function. If additional information is required for the processing of a function, the routine may query the user. The flags mentioned previously are used as part of the command string input to control the optional functions of a command. The meaning of the flag varies depending on the function.

Interaction Forms

Throughout KANDIDATS various messages are displayed prompting the user for additional information. There are two forms in which these messages may appear: a long form and a short form. The short form is intended to speed interaction with the user who is experienced with KANDIDATS. The experienced user may be familiar with what additional information is required by a given command and not need a complete description of the requested input. An example of the long and short forms occurs in the routine DISPLY.

The long form is:

```
IMAGE NUMBER TO DISPLAY (1 - n)
      ENTER 0 TO STOP  (13) -
```

The short form of the same message is:

```
DISPLAY IMAGE (1 - n) -
```

where n is an integer value plugged into the message.

The default form is the long message form. The user may change the form of interaction by entering the commands LONG or BRIEF in the main KANDIDATS routines. In the EXSIF subsystem, the form of the message may be changed by entering the command, BRIEF, followed by the parameter ON, for the short form and OFF, for the long form.

4.4 KANDIDATS Interactive Aids

There are four commands available to the KANDIDATS User that provide information as to available commands, their use, and system information.

1. MESG DEST

This command displays on the requested destination device, DEST, the text of a file called KANMSG SIF. The destination device may be either the terminal, TERM, or the virtual printer, PRTR. The text of KANMSG gives the current status of the KANDIDATS system. Changes made to the system will be noted in this file. It also provides the user with information as to the general command string form, the devices available and their mnemonics, and the unit numbers associated with the various devices. The conventions for naming files are explained.

2. SVOC DEST

This command lists on the destination device, DEST, all the available commands of KANDIDATS. The destination device may be the terminal or the virtual printer. The commands are listed in reverse alphabetical order. All possible commands are listed, not just those that are currently implemented.

The SVOC command is also a command in the EXSIF subsystem. If the SVOC command is entered in the EXSIF subsystem, the destination device is not specified and defaults to the terminal. Only the valid EXSIF commands are displayed.

3. VOCA DEST (FLAGS)

The VOCA command requires a destination device, DEST, which may be either the terminal or the virtual printer. This command lists on that

device a short description (up to 32 characters) of the requested command(s). The flags that the user may specify are A and M. An A indicates that the user wishes to see a description of all the commands. An M stands for multiple descriptions. The user enters one command at a time, that he or she wishes to have explained, in response to a '?' prompt. When the user has finished, he or she may exit the VOCA command by entering QUIT.

4. EXPL DEST (FLAGS)

EXPL provides a more detailed description of the KANDIDATS commands. The destination device and flags are the same as those described for the VOCA command. The information provided for a command by the command EXPL is as follows:

IMPLEMENTED - Yes, the command is currently implemented or no, it is not.

ACTION - A description of the function performed by the command.

DEST - Gives the device types that may legally appear in the destination field of the command string. Any default values are given.

SOURCE - Gives the device types that may legally appear in the source field of the command string. Any default values are given.

FLAGS - Tells the valid characters that may be used as flags for the command. The action that corresponds to each flag is also given.

CTRLT - This section is carried over from the University of Kansas version of KANDIDATS and is not used at Kansas State.

COMMENTS - Any additional remarks or explanation of the use of the commands are given in this section.

4.5 Error Processing

When an error occurs on the KANDIDATS system, a message is displayed for the user. This message may be displayed by the routine in which the error occurred and may give the user the opportunity to take corrective action. If the error is severe enough that no corrective action is possible, the error code along with the sequence of routines leading to the routine in which the error occurred are displayed to the user by the main driver KAND01. The user is then prompted for the next command input line.

The table below lists the error codes currently possible, the routine in which the error code is set, and its meaning in that routine.

Code	Routine	Meaning
-2001	RDKINL RDSC20 RREAD RWRITE	Illegal file code
-2002	RDKINL	Number of bits/point has an illegal value for the data mode
-2002	FDOT HDOT MDOT PDOT	Average of gray tones within Window not within the range of gray tones
-2003	RDKINL	Image size incorrectly specified
-2004	RDKINL	Illegal request
-2005	KAND01 RDKINL	File does not exist
-2006	RDKINL	No available unit numbers
-2006	RWRITE	I/O work area too small
-2007	DISPLY	File does not exist
-2007	RDSC20 RREAD RWRITE	EOF (end of file)

Code	Routine	Meaning
-2008	DISPLY	Subimages are not full image rows
-2008	RDSC20 RWRITE	Write error
-2009	RREAD	Read error
-2010	RREAD RWRITE	Image out of range
-2011	RDKINL	Illegal min/max/nql/nbits combination
-2012	MODE RDKINL RREAD RWRITE	Illegal data mode indicated by IDENT(19)
-2013	RREAD RWRITE	Number of records to access less than one
-2014	RREAD RWRITE	Subimage out of range
-2015	RREAD RWRITE	Number of images in the file less than or equal to zero

4.6 Entering KANDIDATS

At KSU, KANDIDATS is executed from a CMS virtual machine running under VM/370. The user should have some familiarity with the CMS commands and VM/370 as described in the IBM CMS User's Guide [2], the IBM CMS Command and Macro Reference [3], and the KSU CMS Guide [4]. After having signed on to a CMS virtual machine, the user must perform the following steps:

1. The KANDIDATS object files must be obtained and placed on the user's A disk. These files may be obtained currently from the account DPH3CMN3 data set DSMN3.VMMN3.CMSLIB2. The files may be retrieved with a file type of TEXT as follows:

   ```
   OSRETR * TEXT
   ```

   When the files are in the virtual reader, the user enters:

   ```
   READ * TEXT
   ```

2. To load KAND01 the user's virtual machine must be at least 512 K. This can be accomplished by entering the command:

   ```
   DEF STOR AS 512K
   ```

3. KAND01 is loaded by entering:

   ```
   LOAD KAND01 IBFCNT BDEX1 DUMMY
   ```

4. The load module is created by entering:

   ```
   GENMOD KAND01
   ```

5. To clear unnecessary files from the A disk, the commands below must be used:

```
ERASE * TEXT
ERASE LOAD MAP
```

6. Other files that will be required to execute KANDIDATS may be obtained from the file manager space of the account DPH3CMN3. These files are:

```
KAN      EXEC
KANMSG   SIF
KANEXP   SIF
TEK      EXEC (if executing on the COMPUTEK)
```

7. Any image files that the user may need for processing must be obtained. If the files are large or numerous, the virtual machine may have to be enlarged using the DEF STOR command.

8. Any files that are compressed must be expanded:

```
EXPAND  KANMSG  SIF
EXPAND  KANEXP  SIF
```

9. To start the execution of KAND01, the user enters:

```
KAN
```

10. The user must then respond to the following questions:

```
IS THE TERMINAL THE COMPUTEK?  (YES/NO)
DO YOU WISH TO HAVE CHECKPOINTING?  (YES/NO)
DO YOU WISH MESG TURNED OFF?  (YES/NO)
```

If the COMPUTEK question is answered YES, then the TEK EXEC is executed which changes the character and line delete characters for the user's terminal to those used by the COMPUTEK. If the second question is answered NO, checkpointing is turned off for the user's machine. Checkpointing attempts to automatically save the user's files onto his file manager space every few minutes. If the MESG question is answered NO, the user will not receive any messages from other users' machines. Turning messages off will prevent the display of an image from being interrupted by a message from another user.

If the files KANEXP SIF, KANMSG SIF, or KAND01 MODULE are not available on the user's disk when KAN is executed, an error message will be displayed and KAN will terminate. Similarly, if the file TEK EXEC is not available when the user answers the COMPUTEK question YES, an error message will be displayed and execution will end.

5. SYSTEM PROGRAMMING

This chapter describes the KANDIDATS system as it currently exists at Kansas State University and generally updates the University of Kansas KANDIDATS User's Guide [1] for Kansas State. It describes the major routine interfaces, documentation customs and some explanation of how various routines accomplish their functions.

5.1 Labeled Common Areas

This section describes the labeled common areas used in KANDIDATS. If variables within a labeled common area need to be initialized, they are initialized in one of two block data routines: IBFCNT and BDEX1.

COMAND

The labeled common area COMAND contains the information that is needed by system routines.

```
COMMON /COMAND/ IFC, OTDEVN, OTDEV, OTTYP, OTFIL(2), INDEVN,
          INDEV, INTYP, INFIL(2,3), LFLAG(26)
```

IFC — is a pointer to the current function code in the array FNAME in labeled common IBFCNT

OTDEVN — is a pointer to the destination device name in the array DNAME in labeled common IBFNCT

OTDEV — is the destination device unit

OTTYP — is the destination device type:

- = 0 no destination device given in command
- = 1 disk pack
- = 2 virtual punch
- = 3 terminal
- = 4 virtual printer
- = 5 virtual reader

OTFIL — eight character destination file name

INDEVN is a pointer to the source device name in the array DNAME in the labeled common IBFCNT

INDEV is the source device unit

INTYP is the source device type, see OTTYP

INFIL up to three *input file names*, *eight* characters each

LFLAG 26 logical flags, each flag corresponds to a letter of the alphabet and is set true if this letter appears in parenthesis in the last command input, otherwise it is false

COMLIN

The labeled common area COMLIN is used by the command interpreter to hold the last input line. This makes the input line available to subroutines other than those that comprise the command interpreter.

```
COMMON /COMLIN/ LINE(72)
```

LINE contains the 72 characters of the command line, each element of the array contains one character of the command

ERROR

The labeled common area ERROR provides the information necessary to process errors.

```
COMMON /ERROR/ IEV, STKPNT, STKSIZ, STACK(2,20), DEBUG
```

IEV integer event variable containing error codes,

= 0 good status
< 0 error condition

STKPNT pointer to current position in stack

STKSIZ number of elements in stack

STACK array of routine names that have been called

DEBUG Boolean that indicates whether the routine names should be displayed as KDPUSH is called

EXSIF1 and EXSIF2

The labeled common areas EXSIF1 and EXSIF2 contain those variables necessary for the operation of the subsystem EXSIF.

```
COMMON /EXSIF1/ FNAME(60), PTEMP, PTEMP1, FTEMP, FTEMP1, ACTION,
          ALT, BIT, BLK, COL, CURBLK, CURCOL, CURFIL, CRUIMG,
          CURROW, DEVMSK, DFIL, DPTR, I, IB, IMG, J, LSTCOL,
          LSTROW, MCOL, MCOLHI, MCOLLO, MROW, MROWHI, MROWLO,
          MTEMP, MTEMP1, NCHARS, NEWBLK, NEWIMG, NUM, RET1,
          RET2, RET5, RET6, RET7, ROW, X, XU, Y, YU, CPARAM(49),
          DEVTAB(4), LABEL(2), NUMBRS(10), STATUS(60), DNAME1(5),
          DNAME2(3), DNAME3(3), SPARAM(49), LIMITS(2,4), PFORM(2,8),
          TFILE1(2), TOKEN(9), BFLAG, BRFTAB(2), CHANGE, RENAME,
          EXISTS, EXSIFN, BELLS

COMMON /EXSIF2/ HTEMP, HTEMP1
```

Only the most significant elements of these labeled common areas are described here.

FNAME array of EXSIF subsystem command names

CPARAM parameter values for file currently being processed

STATUS number associated with the command which indicates what files, if any, must be open to perform the function

= 0 doesn't matter
= 1 current file must be open
= 2 file A must be open
= 3 file B must be open
= 4-7 illegal or unused at this time

SPARAM parameter values for secondary file

EXSWKA

The labeled common area EXSWKA contains the work array used in the EXSIF subsystem and the size of that work array.

```
COMMON /EXSWKA/WRKSIZ, WORK(1300)
```

IBCSIZ

The common area IBCSIZ contains information necessary to logical unit use.

```
COMMON /IBCSIZ/ BRIEF, INTT, OTTT, LONG, SHORT, RUN, TTYIN,
          TTYOT, IBMOT, RUNDAT, EXPDAT, SCDEV1, SCDEV2, SCDEV3,
          CDRDAT, MSGDAT
```

BRIEF	Boolean indicating whether messages to the user are to be brief
INTT	unit from which command input is to be taken
OTTT	unit to which KANDIDATS Messages are to be displayed
LONG	Boolean indicating whether messages to the user are to be long
SHORT	Boolean indicating messages are to be short
TTYIN	logical unit from which terminal input is to be taken
TTYOT	logical unit to which terminal output is to be sent
IBMOT	logical unit to which virtual printer output is to be sent
RUNDAT	logical unit for batch command input. Unused at this time
EXPDAT	logical unit from which information for the EXPL command is taken
SCDEV1 SCDEV2 SCDEV3	scratch logical unit numbers for disk files. Unused at this time
CDRDAT	logical unit number used for virtual reader input
MSGDAT	logical unit containing information used by the MESG command

IBFCNT

The common area, IBFCNT, contains function names and device information necessary to interpret KAND01 commands.

```
COMMON /IBFCNT/ FNAME(80), DNAME(20), DTYPE(21), DUNIT(21)
```

FNAME — array containing the names of functions

DNAME — array containing the mnemonics for the devices available

DTYPE — array containing the code corresponding to the device name indicating the type of that device. Note that DTYPE(1) = 0, DTYPE(2) corresponds to DNAME(1), etc.

= 0 no device
= 1 disk pack
= 2 virtual punch
= 3 terminal
= 4 virtual printer
= 5 virtual reader

DUNIT — array containing the unit numbers of the devices named in DNAME. DUNIT(1) = 0 and DUNIT(1) corresponds to DNAME(2), etc.

IO

The labeled common area IO contains the array work areas for KAND01.

```
COMMON /IO/ IWORK(512), NSIZE
```

IWORK — the work area

NSIZE — size of the array IWORK

MACH

MACH is a common area which contains the variables that hold values that are machine dependent.

```
COMMON /MACH/ NMBTPM, NWORDS(5), NMWDIO, NMWMIN
```

NMBTPM number of bits per machine word

NWORDS number of words for each type of data mode

NWORDS(1) - integer
NWORDS(2) - absolute integer
NWORDS(3) - real (floating point)
NWORDS(4) - alternate integer
NWORDS(5) - double precision

NMWDIO number of words per IO word

NMWMIN number of machine words necessary to contain the IDENT array
20*NWORDS(1)

NEWLNE

NEWLNE contains the variables that contain the hex representation of characters that cause the terminal cursor to go to a new line.

```
COMMON /NEWLNE/ IALT, BELLS
```

IALT character to suppress carriage return, unused

BELLS bell character

UNTS

The labeled common area UNTS keeps track of which files are assigned which unit numbers.

```
COMMON /UNTS/ UTAB(5)

UTAB        table of file names
            index + 10 is corresponding unit number
```

5.2 Command String Interpreter

The main driver of KANDIDATS, KAND01, calls the routine COMIN2 which accepts the input commands for the system. COMIN2 prompts the user for input and parses that input. As the various parts of the command are parsed, they are validated. The order in which the various parts of the command are expected is as follows:

```
function  destination  destination  (flags)  < -
            device        file

          source       source       source    (flags)
          device       file 1       file 2
```

The function name is located in the table of function names. If the name is not found in the table, it is invalid. The index in the table of a valid function name is used by KAND01 to determine the appropriate command driver to call. The destination and source devices, if present, must be located in the device table to be valid.

As the various parts of the command are recognized, values are placed in the labeled common area, COMAND, for use by the individual command drivers. These individual command drivers determine if the necessary information was provided for the processing of that command. If not the command driver may request additional information.

5.3 KANDIDATS Drivers

The drivers of the individual commands have a standard format. This format is as follows:

```
      CALL KDPUSH('PROG','NAME')
      CALL DEVCHK (DEVMSK, NSFILS, NDPAIR, &900)

      CALL RDKINL (IDAT, FILNM, IDENT, IRDWRT, IEV, &900)
      .
      .
      .
      CALL Processing routines
      .
      .
      .
      CALL KDPOP
      RETURN

  900 CONTINUE

          Error Routines

      END
```

The routine KDPUSH places the program name (PROGNAME in the example) on the stack used for error processing. DEVCHK validates the source and destination devices for the command. If files are required for use by the command, the routine RDKINL is called to initialize the file for access. The driver may do some processing or may call on other routines to do the processing. Finally, if an error has not occurred, the routine KDPOP is called to clear the program name from the stack.

5.4 KANDIDATS Routines

Internal Routine Documentation

Each of the routines of KANDIDATS is internally documented to facilitate alterations to the code. This documentation is as follows:

I. Identification

Version A Date mm/dd/yy Source - routine name

Program Name	routine name
System	machine
Source Language	language
Author	name

II. Purpose

A brief description of the main function of the routine should occur here.

III. Operating Procedures

A description of the entry points of the routine and the argument list of those entry points.

IV. Hardware Required

Any hardware required by the routine is listed here.

V. Program Environment

A brief description of the software environment necessary for the successful functioning of the routine.

VI. Limitations

A brief description of any limitations in the functioning of the routine.

VII. Data Formats

A description of any formats used in the routine.

VIII. Internal Description

If further explanation of the workings of the routine is needed, it is given here.

IX. Statement Numbers

The statement numbers used in the routine are listed.

X. Subroutines Called

Subroutines called by this routine are listed.

The first line of each routine is a comment with the name of the routine left justified and also centered with the characters separated by dashes. For the routine ERRPRC, this line would be:

```
EERRPRC                    E-R-R-P-R-C
```

Following this line would be the introductory documentation described above. Throughout the code of the routine should be comments that describe the action of the code. Blank comment cards should be used to separate the various sections of code which may have different functions.

Internal Routine Structure

Code in KANDIDATS should be kept as structured as possible. Routines should be kept to one main function. The code should be written in a top-down manner and should be readable. Following the SUBROUTINE statement should be the type statements in the following order:

```
INTEGER
INTEGER*2
REAL
DOUBLE PRECISION
LOGICAL
DIMENSION
COMMON
EQUIVALENCE
DATA
```

Variables of the same type should be defined together.

The code follows the type statements. Five conventions for coding are:

1. The flow of control should be downward as much as possible. When errors occur, control should be transferred to a statement numbered 9000-9999.

2. CONTINUE statements should be used as statements for control to be transferred to. As shown in the example below, comment statements should be included after each CONTINUE statement.

```
100 CONTINUE
      comments on function that follows processing
```

   This allows the reader to encounter the comments before the code that it explains. Another convention that aids reading the code is to indent the comments from the code.

3. The terminating statement of a DO-loop should always be a CONTINUE statement. The body of a DO-loop should be indented to aid reading. For example:

```
      DO 100 I = 1,K
         T(I) = BLANK
  100 CONTINUE
```

4. There should be only one point for normal return and one point of return for each error return. An example is shown below.

```
      GO TO 9000
      .
      GO TO 9009
      .
      GO TO 800
      .
      .
      .
      GO TO 800
      .
      .
      .
  800 CONTINUE
C
C           NORMAL RETURN
C
      RETURN
C
 9000 CONTINUE
C
C           ERROR RETURNS
C
      IEV = -1000
      GO TO 9999
C
 9009 CONTINUE
C
C           SECOND ERROR RETURNS
C
      IEV = -2000
C
 9999 CONTINUE
C
           RETURN 1
```

5. The format statements should come at the end of the executable statements in a routine. Statement numbers for input format statements should range 4000 - 4999 and output statement numbers from 6000 - 6999.

I/O

There are conventions concerning I/O that should be followed by KANDIDATS routines. When a routine requests extra input, there are two forms that this request may take: a long form and a brief form. The long form gives the information necessary for the inexperienced user to enter a response to the request. The request should include a description or name of the needed input, a range of possible values if available, and the format that will be used to accept the input. The brief form of requesting input is for the experienced user who is familiar with the requests made by a command. All questions should be formulated with both a long and brief form. The use of the two forms is controlled by two logical variables LONG and BRIEF.

These two variables, along with others, are part of the common block IBCSIZ. This common block is described in section 5.1. IBCSIZ also contains variables that contain the unit numbers of the input and output devices. Currently I/O for question-response is to and from the terminal.

Format statements for requests should be numbered 6000 - 6999, and input formats should be numbered 4000 - 4999. These format statements should be grouped together at the end of the routine. Integer values

to be entered should be read in with an I3 format where the possible values do not exceed 999. An example of a possible request/response is shown below.

```
(Long)    6000 FORMAT('IMAGE NUMBER TO DISPLAY (1 - ',I3,
         2       ')'/, 'ENTER 0 TO STOP (I3) - ')
(Short)   6001 FORMAT('DISPLAY IMAGE (1 - ', I3, ') - ')
(Input)   4000 FORMAT(I3)
```

Where possible, input should be checked for valid input values. If the input was invalid, it may be possible to request the input again using the long form of request.

5.5 Adding Commands

(1) Add the name of the new command to the array FNAME in the common block IBFCNT in the block data routine IBFCNT.

(2) Add a brief 30 character description of the command to the arrays VOC(1 - 8) in the subroutine VOCA in the corresponding position of the name in FNAME.

(3) In KANDO1 the computed GO TO that branches based on the position of the command in FNAME should be altered. A statement number should be added in the position corresponding to the new command.

(4) Add the new statement number to KANDO1 and call the new commands driver to that point.

(5) Any altered or added routines should be recompiled. The text (object) files of these routines should be saved with the other text files.

(6) Add an explanation of the command to the file KANEXP SIF using the CMS editor. The command should be added in alphabetical order to the file.

If the commands are to be ported from the original version of KANDIDATS, the following should be done:

(1) The original source of the routines that comprise the new command must be obtained. Alterations must be made to the code as described in Chapter 3.

(2) Once the code has been tested, the text files of the routines used by the command must be saved with the other text files.

(3) Finally, the stub for the command driver must be removed from the file DUMMY FORTRAN. DUMMY must be reassembled.

5.6 Image I/O Routines

There are three image disk I/O routines. RDKINL initializes files for access, RREAD and RWRITE (read and write respectively), all or any part of an image.

Before a file can be accessed, it must be initialized by calling RDKINL. RDKINL can initialize an old or new file. For an old file the disk is checked to see if the file exists. If the file exists, the record size and number of records for that file are obtained from the operating system. The unit number of the file is found and a DEFINE FILE is performed for that file. At this point the first record of the file is read from disk. This record contains the IDENT array which will provide the necessary information for file access. For a new file the IDENT array is provided and checked for validity. If the values are valid, a unit number is assigned to the file and a DEFINE FILE is performed. Finally, the IDENT array is written in the first record of the file.

RREAD and RWRITE can read/write records of any of the five data modes of an image. All files are random access files. These routines may read/write any specific or all images of a file. The record to begin accessing and the number of records to access must be specified.

5.7 Error Processing

Error processing in KANDIDATS uses an event variable (IEV) to contain an error code. When an error occurs, the code associated with that error is assigned to the event variable and an error return is taken. When control returns to the driver KAND01, the event variable is checked for an error code (IEV < 0). If an error has occurred, the routine ERRPRC is called to process the error. ERRPRC displays the value of the event variable and the contents of the stack.

The stack is maintained by two routines, KDPUSH and KDPOP. KDPUSH places the name passed to it on the stack. Each routine which may set the error code when first entered calls KDPUSH passing its own name. On normal return, the routine calls KDPOP which removes the name on the top of the stack. An error return does not call KDPOP, leaving the name on the stack. The stack, therefore, provides a trace of the routines called leading up to the error.

5.8 Important System Routines

This section documents the interfaces to some important system routines. These are common routines used for interpreting commands, image I/O and operating system interface routines.

I. Command String Interpreter and Error Checking

- A. COMIN2
- B. DEVCHK
- C. FILCHK
- D. IBVOCA

II. Image Access Routines

- A. RDKINL
- B. RREAD
- C. RWRITE

III. Operating System Interface Routines

- A. ADATE
- B. DEFDA
- C. DEFILE
- D. DEFINE
- E. DLETE
- F. FORM
- G. FSTAT
- H. INUNIT

The initial identification given here for these routines is for the University of Kansas version of KANDIDATS. The date and author of the changes for the IBM-370 is given following PORTED/DATE and PORTED/AUTHOR.

1. Command String Interpreter and Error Checking

A. COMIN2

Program Title: COMIN2

Version: B

Date: 1/20/74

Author: Bruce J. Morgan

Documented by: Bruce J. Morgan

Language: FORTRAN IV

Implemented on: PDP-15

Ported/Date: 7/27/77

Ported/Author: Linda Lallement

Ported/To: IBM-370

Purpose:

This routine is the driver of the command string interpreter for KANDIDATS. It accepts and partially decodes command strings. It is set up so that command input may be taken from either the terminal or a disk run file. Currently the run files are not implemented.

Entry point:

```
COMIN2 (IN, PROMPT, FNAME, DNAME, IFC, OTDEVN, OTFIL,
       INDEVN, INFIL, LFLAG, *)
```

Arguments:

IN	Command input file code. Unit number 9 for the terminal. (input)
PROMPT	A two character prompt left justified in an integer word. These characters are displayed at the terminal before the command is accepted. (input)
FNAME	A double precision array of the valid function names. The array is terminated by a blank name. (input)

DNAME	Integer array of 4 character device names. Array is terminated by a blank name. (input)
IFC	Pointer to the function name in the array FNAME. (output)
OTDEVN	Pointer to the destination device name in the array DNAME. If no destination device was specified in the command, OTDEVN is zero. (output)
OTFIL	The destination file name if one was given. If none was given, OTFIL is blank. (output)
INDEVN	Pointer to the source device name in the array DNAME. If source device name was not given, then INDEVN is zero. (output)
INFIL	An array of up to 3 source file names. If none are specified, INFIL is blank. (output)
LFLAG	Array of 26 logical flags corresponding to the letters of the alphabet. An element of the array is set to .TRUE. if the letter occurs in the command string. Otherwise the flag is .FALSE.
*	If an end-of-file is read from disk, this error return is taken.

Subroutines required:

GETDN
SCAN2
ENCODE
GETFN
GETFLG

B. DEVCHK

Program Title: DEVCHK

Version: A

Date: 1/20/74

Author: Bruce J. Morgan

Documented by: Bruce J. Morgan

Language: FORTRAN IV

Implemented on: PDP-15

Ported/Date: 12/3/77

Ported/Author: Linda Lallement

Ported/To: IBM-370

Purpose:

This routine is used by the command string processor to translate the device mnemonic into a device type and a unit number. The device type is checked against the legal devices for a command. If the device type was disk and no file name was specified in the command, a file name is requested.

Entry point:

DEVCHK (DEVMSK, NSFILS, NDPAIR, *)

Arguments:

DEVMSK — An integer word used to define the valid device pair combinations for a command. The bit is on if the device pair is valid and off if invalid. (input)

Bit	Destination	Source
0	Unused	
1	None	None
2	Disk	None
3	Disk	Disk
4	Disk	Punch
5	Disk	Terminal
6	Unused	
7	Punch	None
8	Punch	Disk

Bit	Destination	Source
9	Punch	Punch
10	Punch	Terminal
11	Unused	
12	Terminal	None
13	Terminal	Disk
14	Terminal	Punch
15	Terminal	Terminal
16	Unused	
17	Printer	None
18	Printer	Disk
19	Printer	Punch
20	Printer	Terminal

NSFILS — The number of source disk file names required by a command. (input)

NDPAIR — The bit number in DEVMSK corresponding to the device pair given in the command. (output)

* — The non-standard error return taken if one of two conditions occur:

1. An illegal pair of devices is given in the command. A message is displayed at the terminal prior to the return.

2. A null response is given to a file name request.

The variables OTDEV, OTTYP, INDEVN, and INTYP in the labeled common area COMAND are set to reflect input and output device types and unit numbers.

Subroutines Called:

FILCHK

C. FILCHK

Program: FILCHK

Version: A

Date: 01/20/74

Author: Bruce J. Morgan

Documented by: Bruce J. Morgan

Language: FORTRAN IV

Implemented on: PDP-15

Ported/Date: 7/10/77

Ported/Author: Linda Lallement

Ported/To: IBM-370

Purpose:

This routine is part of the command string processor. It is called by DEVCHK to verify the existance of the disk file names when the disk was specified as a device type in a command. For output files, a non-blank name must be present. For input files, not only must a name be present, but there must be a file with that name on the user's disk. If any names are missing, they are requested through the terminal. The files are not initialized by this routine.

Entry point:

FILCHK (MODE, DAT, FILNAM, *)

Arguments:

MODE — Indicates kind of checking to be done:

= 0 output file
= 1 input file

(input)

DAT — Unit number on which input files must be found. (input)

FILNAM	Array containing the name of the file to be checked. If a new name is requested, FNAME will be changed on return. (input/output)
*	Error return taken if no response (carriage return) is given to a request for a file name.

Subroutines Called:

FSTAT

D. IBVOCA

Program: IBVOCA

Version: A

Date: 1/20/74

Author: Bruce J. Morgan

Documented by: Bruce J. Morgan

Language: FORTRAN IV

Implemented on: PDP-15

Ported/Date: 8/21/77

Ported/Author: Linda Lallement

Ported/To: IBM-370

Purpose:

This routine processes a VOCA or SVOC command. The VOCA command function prints the command name a short description of the command. SVOC lists all the available commands.

Entry point:

IBVOCA (OTSLOT, VNAME, LONG, FNAME, FVOCA)

Arguments:

OTSLOT	Unit number on which output is to be displayed. (intput)
VNAME	Program name whose vocabulary is to be displayed. Currently VNAME is always 'KANDO1ØØ'. (input)
LONG	Logical switch indicating the type of vocabulary listing that is desired: = .TRUE. VOCA vocabulary = .FALSE. SVOC vocabulary (input)

FNAME	Array containing list of valid function names. This array must be terminated with a blank name. (input)
FVOCA	Array containing the descriptions of the function names in FNAME. There is a one-to-one correspondence between the commands in FNAME and their description in FVOCA.

Subroutines Called:

COMIN2

II. Image Access Routines

A. RDKINL

Program: RDKINL

Version: C

Date: 6/5/74

Author: D. Johnson

Documented by: D. Johnson

Language: FORTRAN IV

Implemented on: PDP-15

Ported/Date: 11/5/77

Ported/Author: Linda Lallement

Ported/To: IBM-370

Purpose:

This routine initializes an image file for input or output. This includes assigning a unit number, performing a DEFINE FILE and establishing the IDENT array.

Entry point:

RDKINL (IDAT, FILNM, IDENT, IRDWRT, IEV, *)

Arguments:

IDAT	Is the unit number of access the file on. (output)
FILNM	Name of the file to access. (input)
IDENT	Array of 20 integers providing identification information for an image. (input for new image/output for old image). See Chapter 2 for a complete description of the IDENT array.
IRDWRT	Indicates whether an old or new file is being accessed. (input) = 1 old file (read only) = 2 new file (write only)

IEV	Integer event variable. (output)	
	= 1	success
	= -2001	illegal file name
	= -2002	number of bits/point has an illegal value for data mode
	= -2003	size incorrectly specified
	= -2004	illegal request
	= -2005	file does not exist
	= -2006	no available unit numbers
	= -2011	illegal min/max/nql/nbits combination
	= -2012	illegal data mode
*	Error return	

Subroutines Called:

FSTAT
DEFINE
KDPUSH
KDPOP
GETSIZ
DLETE
INUNIT

B. RREAD

Program:	RREAD
Version:	B
Date:	7/8/74
Author:	D. Johnson
Documented by:	D. Johnson
Language:	FORTRAN IV
Implemented on:	PDP-15
Ported/Date	11/5/77
Ported/Author	Linda Lallement
Ported/To	IBM-370

Purpose:

This routine reads accesses SIF files in all five data modes.

Entry point:

```
RREAD (IDAT, MARRAY, ND, NR, NC, IMG, LINE, IDY, IDENT,
      IEV, *)
```

Arguments:

IDAT	Unit number to access the file on. (input)
MARRAY	3-D array. This array must be of the same type as the file to be accessed. The number of rows must equal the number of lines to read for each image. The number of columns must equal the number of points per line. The depth must be the number of images. For example: for 4 images, one line for each image and 100 points per line the dimension would be MARRAY (4,1,100). (output)
ND	Number deep for MARRAY. (input)
NR	Number of rows for MARRAY. (input)

NC	Number of columns for MARRAY. (input)
IMG	Image number to access. (input) = 0 all images in the file determined by IDENT(17) = N where 0 < N ≤ IDENT(17) only the data for image N will be returned
LINE	Record number of the image to begin access on. (input)
IDY	Number of records to access for each image. (input)
IDENT	Array of 20 integer words providing information about the image. (input)
IEV	Integer event variable. (output) = 1 Success = -2001 Illegal file code = -2007 EOF = -2009 Read error = -2010 Image out of range = -2012 Illegal data mode = -2013 IDY less than 1 = -2014 Subimage out of range = -2015 Number of images in file less than or equal to zero
*	Alternate error return

Subroutines Called:

UNPACK
FREAD
HREAD
PREAD
RDR
KDPUSH
KDPOP

C. RWRITE

Program: RWRITE

Version: B

Date: 7/10/74

Author: D. Johnson

Documented by: D. Johnson

Language: FORTRAN IV

Implemented on: PDP-15

Ported/Date: 10/8/77

Ported/Author: Linda Lallement

Ported/To: IBM-370

Purpose:

This routine allows the user to write SIF files in all five data modes.

Entry point:

```
RWRITE (IDAT, MARRAY, ND, NR, NC, IMG, LINE, IDY, IDENT,
       IEV, *)
```

Arguments:

IDAT	Unit number to access file on. (input)
MARRAY	3-D array. This array must be of the same type as the file to be accessed. See RREAD for dimension information. (input)
ND	Number deep for MARRAY. (input)
NR	Number of rows for MARRAY. (input)
NC	Number of columns for MARRAY. (input)
IMG	Images to access. (input) = 0 write all images in the file = N where $0 < N \leq$ IDENT(17), write image N only

LINE	Record number of the image to begin writing to. (input)
IDY	Number of records to write for each image. (input)
IDENT	Array of 20 integers containing information about the file. (input)
IEV	Integer event variable. (output)

= 1	Success
= -2001	Illegal file code
= -2006	/IO/ too small
= -2007	EOF
= -2008	Write error
= -2010	Image out of range
= -2012	Illegal data mode
= -2013	IDY less than one
= -2014	Subimage out of range
= -2015	Number of images in files less than or equal to zero

*	Alternate error return

Subroutines Called:

PACK
FWRITE
HWRITE
PWRITE
WRTR
KDPUSH
KDPOP

III. Operating System Interface Routines

A. ADATE

Program Title: ADATE

Version: A

Date: 7/7/77

Author: Linda Lallement

Language: FORTRAN IV

Implemented on: IBM-370

Purpose:

This routine returns the current date and time in four integer words. The format is: MM/DD/YYbbbHH:MM

Entry point:

ADATE (FDATE)

Arguments:

FDATE	Formatted date as described above. (output)

Subroutines called:

DATE - FORTRAN system routine
FORM
TIME - FORTRAN system routine

Pseudo-Code:

```
Call DATE to get the numeric month, day, and year

Call TIME to get the time of day in seconds

Hours ← seconds of day/360000

Minutes ← (seconds - hour * 360000)/6000

Call FORM to convert the date, hours and minutes into
the proper form

Return
```

B. DEFDA

Program Title: DEFDA

Version: A

Date: 8/2/77

Author: Linda Lallement

Documented by: Linda Lallement

Language: 370 Assembler

Implemented on: IBM-370

Purpose:

This routine initializes a file for direct-access operations. It associates a logical unit number with a given file name. The routine executes the FILEDEF command of CMS. The unit number, record size, block size, and name are plugged into a parameter list for a supervisor call (SVC).

Entry point:

DEFDA (UNIT, RECSIZ, NAME)

Arguments:

UNIT	Logical unit number for the file. (input)
RECSIZ	Record length in bytes per record. (input)
NAME	Name of the file. (8 characters) (input)

Subroutines called:

None

Pseudo Code:

Convert unit number to character representation.

Move into parameter list.

Convert record length to character representation.

Move into parameter list.

Move the record size into the parameter list for the block size.

Move the file name into the parameter list.

Execute the file definition by executing an SVC.

Return

Parameter List for SVC:

```
COMAND          DS      0D              Command list for SVC
                DC      CL8'FILEDEF'
UNIT            DC      CL8'FILEDEF'    Unit number
                DC      CL8'DISK'
FN              DC      CL8'DISK'       File name
FT              DC      CL8'SIF'        File type
FM              DC      CL8'A'          File mode
                DC      CL8'C'
                DC      CL8'DSORG'
                DC      CL8'DA'         Direct access
                DC      CL8'PERM'
                DC      CL8'RECFM'
                DC      CL8'F'          Fixed length records
                DC      CL8'LRECL'
RECSIZ          DC      CL8'LRECL'      Record length
                DC      CL8'BLOCK'
BLKSIZ          DC      CL8             Block size
                DC      X'FFFFFFFFFFFFFFFF'
```

C. DEFILE

Program Title: DEFILE

Version: A

Date: 9/3/77

Author: Linda Lallement

Documented by: Linda Lallement

Language: 370 Assembler

Implemented on: IBM-370

Purpose:

This routine performs the FORTRAN DEFINE FILE statement for variable values of unit number and number of records in the file for formatted and unformatted files. This is done by plugging these values into a parameter list for the code that performs the DEFINE FILE.

Entry point:

DEFILE (UNIT, RECN, RECSIZ, LETTER, ASSOC)

Arguments:

UNIT	Unit number. (input)
RECN	Maximum number of records in the file. (input)
RECSIZ	Size of records (input) bytes for formatted. Words for unformatted.
LETTER	The single letter 'E' or 'V'. (input) E - formatted V - unformatted
ASSOC	Associated variable. (input)

Subroutines called:

None

Pseudo Code:

Move unit number into parameter list.

Move number of records into parameter list.

Move record size into parameter list.

Move letter into parameter list.

Move address of associated variable into parameter list.

Branch to define file code.

Return.

Parameter List:

```
UNIT        DC      CL1' ', AL3 (RECN)
EU          DC      C' ', AL3 (RECSIZ)
AVAR        DC      X'80', AL3 (ASSOC)
ASSOC       DS      F           Address of associated
                                variable.
RECN        DS      F           Number of records.
RECSIZ      DS      F           Size of records.
```

D. DEFINE

Program Title: DEFINE

Version: A

Date: 9/4/77

Author: Linda Lallement

Documented by: Linda Lallement

Language: FORTRAN IV

Implemented on: IBM-370

Purpose:

This subroutine initializes a file for direct-access operations. It contains the DEFINE FILE statement for both formatted and unformatted I/O. It also performs the FILEDEF to associate the logical unit number with the file.

Entry point:

DEFINE (UNIT, RECSIZ, RECN, NAME, ASSOC, MODE, ADJ, DEL)

Arguments:

UNIT	Logical unit number for the file. (input)
RECSIZ	Record length in bytes per record for formatted I/O and in words per record for unformatted I/O. (input)
RECN	The number of records in the file. (input)
NAME	The name of the file. Eight characters. (input)
ASSOC	The associated variable for the file. (input)
MODE	I/O mode. (input) = 0 unformatted I/O ≠ 0 formatted I/O
ADJ	File adjustment option. (unused)
DEL	File delete option. (unused)

Subroutines called:

DEFILE
DEFDA

Pseudo-Code:

```
if unformatted I/O (mode = 0)
  then begin
       SIZ ← rec size in words * 4 bytes per word
       call DEFILE to do DEFINE FILE for unformatted I/O
       end
  else begin
       SIZ ← rec size in bytes
       call DEFILE to do DEFINE FILE for formatted I/O
       end
call DEFDA to do file definition using SIZ
return
```

E. DLETE

Program Title: DLETE

Version: A

Date: 7/9/77

Author: Linda Lallement

Documented by: Linda Lallement

Language: 370 Assembler

Implemented on: IBM-370

Purpose:

This subroutine searches for a file on the users disk. If it is found, it is deleted. The search is performed by the FSSTATE system macro instruction. The equivalent of the CMS ERASE statement is performed by the FSERASE system macro instruction.

Entry point:

DLETE (N, NAME, FOUND)

Arguments:

N	Device number on which to search for file. (unused)
NAME	Name of file to be deleted. Eight characters. (input)
FOUND	Completion code. (output) = 0 file not found = 1 file found and deleted

Subroutines called:

None

Pseudo Code:

```
load name of the file into the parameter list
if file exists
  then begin
       delete the file
       if successful
           then begin
                set return code to 1
                return
                end
       end
set return code to 0
return
```

Parameter List:

```
ID              DS      0D
NAME            DC      CL8' '          File name
TYPE            DC      CL8'SIF'        File type
                DC      CL8'A'          File mode
```

F. FORM

Program Title: FORM

Version: A

Date: 7/7/77

Author: Linda Lallement

Documented by: Linda Lallement

Language: FORTRAN IV

Implemented on: IBM-370

Purpose:

Given the numeric hour and minute of the day, this routine returns a formatted version of the time of day. Format: ØØØHH:MM

Entry point:

FORM (HOUR, MIN, LDATE, *)

Arguments:

HOUR	Numerical representation of the hour of the day. (input)
MIN	Numerical representation of the minute of the hour. (input)
LDATE	Formatted version of the hour and minute of the day. Format: ØØØHH:MM. (output)
*	Error return taken if hour or minute are not valid numbers.

Subroutines called:

None

Pseudo Code:

```
if hour < 0 or hour > 24
    then error return

HR ← character representation of hour from the table

if minutes < 0 or minutes > 60
    then error return
```

```
MN ← character representation of minutes from the table
LDATE(1) ← blank
LDATE(2) ← blank
LDATE(3) ← blank
LDATE(4) ← HR(1)
LDATE(5) ← HR(2)
LDATE(6) ← colon
LDATE(7) ← MN(1)
LDATE(8) ← MN(2)
return
```

G. FSTAT

Program Title: FSTAT

Version: A

Date: 9/18/77

Author: Linda Lallement

Documented by: Linda Lallement

Language: 370 Assembler

Implemented on: IBM-370

Purpose:

This routine determines whether or not a file exists on the user's disk. It assumes a file type of SIF. If the file is found, the record length and number of records in the file are returned in the labeled common block FILSIZ.

```
COMMON /FILSIZ/RECLEN, NREC
```

The search is performed by the FSSTATE system macro instruction.

Entry point:

```
FSTAT (N, NAME, FOUND)
```

Arguments:

N	Device number. (unused)
NAME	File to be found. Eight characters. (input)
FOUND	Completion code. (output) = 0 file not found = 1 file found

Subroutines called:

None

Pseudo Code:

```
load the name of the file into the parameter list
if the file exists
    then begin
        place the record length in the common block
        place the number of records in the common block
        completion code ← 1
        end
    else completion code ← 0
return
```

Parameter List:

```
ID              DS      0D
NAME            DC      CL8' '          File name
TYPE            DC      CL8'SIF'        File type
                DC      CL8'A'          File mode
```

H. INUNIT

Program Title: INUNIT

Version: A

Date: 10/30/77

Author: Linda Lallement

Documented by: Linda Lallement

Language: FORTRAN IV

Implemented on: IBM-370

Purpose:

This routine determines the unit number associated with a particular file. If no unit number is currently assigned to a particular file, one is assigned. A maximum of 5 unit numbers can be used in one program. Unit numbers will range from 11 to 15.

Entry point:

INUNIT (IDAT, FILNM, *)

Arguments:

IDAT	Unit number of file. (output)
FILNM	File name. Eight characters. (input)
*	Error return if table is full.

Subroutines called:

None

Pseudo Code:

```
for I = 1 to 5
    begin
        if UTAB(I) = filename
            then begin
                unit ← I + 10
                return
                end
    end
for I = 1 to 5
    begin
        if UTAB(I) = blanks
            then begin
                UTAB(I) ← filename
                unit ← I + 10
                return
                end
    end
error return
```

6. EXAMPLE

This chapter presents an example of the use of KANDIDATS at Kansas State University. The example assumes that the necessary files are available on the user's file manager space or data set. The information to be entered by the user will be typed in small letters. The prompts or responses made by the machine will be typed in large letters. Comments are to the side. 'R;' is the CMS prompt. Once the terminal has been connected to the 370 through phone lines or turned on, the user strikes the attention key (or the shift 0 on the COMPUTEK). The machine responds with:

```
VM/370 ONLINE LFJ359 QSYOSU

logon cms                                          begins session
ENTER ACCOUNT NUMBER
[*] [*] [*] [*] [*] [*] [*] [*]                    user types in
ENTER PASSWORD                                     accounting information
[*] [*] [*] [*] [*] [*] [*] [*]                    in masked out
ENTER SS NUMBER                                    areas
[*] [*] [*] [*] [*] [*] [*] [*] [*]
LOGON AT 16:02:16 CST FRIDAY 02/26/78
CMS V2 PLC 13 - 2/13/76 20:36
FORMATTING DISK 'A'.
R;
osretr * text                                      Begin to retrieve
                                                   files that comprise
R;                                                 KANDIDATS. This may
                                                   take some time. The
                                                   user may wish to logoff
                                                   and logon again later
                                                   or put the machine to
                                                   sleep. Once a message
                                                   is received that the
                                                   files are in the
                                                   virtual reader:
read *
  "files read are listed"
R;

def stor as 512K                                   Re-configure machine
CP ENTERED; DISABLED WAIT PSW                      to at least 512K.

CP
ipl cms
CMS V3 PLC
```

```
load kand01 ibfcnt bdexl dummy

R;

genmod kand01                                  The load module
                                               should now
                                               exist on the user's
                                               disk.

erase * text                                   Erase unnecessary
R;                                             files.
erase load map
R;

fmretr kan exec                                Retrieve the
R;                                             necessary files
fmretr kanmsg sif                              from the user's
R;                                             file manager space.
fmretr kanexp sif
R;
fmretr tek exec
R;

fmretr picexp sif                              Retrieve any image
R;                                             files to be worked with.
                                               In this example, we
                                               will use the image
                                               file PICEXP.

expand kanmsg sif                              Expand any files
R;                                             that are compressed.
expand kanexp sif
R;
expand picexp sif
R;
kan                                            Execute EXEC file
                                               that begins KANDIDATS.
IS THE TERMINAL THE COMPUTEK?  (YES/NO)
yes
TEK EXEC                                       Changes line and
COMPUTEK LINE EDITING                          character delete
                                               characters.
DO YOU WISH TO HAVE CHECKPOINTING?
(YES/NO)
no

DO YOU WISH MESG TURNED OFF?  (YES/NO)
yes
```

```
KANEXP SIF
KANMSG SIF
KAND01 MODULE
```

If these files are not on the user's disk, an error will occur here.

```
KANDIDATS -- 77DECEMBER4
```

Now under the control of KANDIDATS

```
#:
```

Prompt for command

```
disply term <- dpa picexp
```

Displaying example picture.

```
IMAGE NUMBER TO DISPLAY (1-3)
    ENTER 0 TO STOP (13) -
ØØ0
```

PICEXP is assumed to have 3 images

```
#:

exsif
```

Enter examine SIF file subsystem.

```
  EXAMINE V2B 12/27/77
A
```

Subsystem prompt for 'A' file.

```
open dp picnew
```

Open a new file called PICNEW.

```
    -FILE PICNEW NOT FOUND-OPEN NEW FILE
                                ("Y"/"N")?

y
  IDENT(1) --
0
  IDENT(2) --
0
  IDENT(3) --
0
  IDENT(4) --
0
  IDENT(5) --
0
  IDENT(6) --
```

Values for the IDENT array must be entered.

20 pointers per row.

```
Ø20
    IDENT(7) --                          10 rows per image.
 10
    IDENT(8) --
  0
    IDENT(9) --
  0
    IDENT(10) --
  0
    IDENT(11) --
 51
    IDENT(12) --
 20
    IDENT(13) --                         20 points per subimage.
 20
    IDENT(14) --                         1 row per subimage.
  1
    IDENT(15) --                         10 is the minimum
 10                                      gray tone.
    IDENT(16) --                         60 is the minimum
 60                                      gray tone.
    IDENT(17) --                         1 image.
  1
    IDENT(18) --
  0
    IDENT(19) --                         Data mode is positive
  0                                      integer.
    IDENT(20) --
  0

 IMAGE - 1     BLOCK - 1                 Identify block.
    1    1 -                             Request values for
                                         point at row 1 -
 10                                      column 1.
                                         Give that point a
    1    2 -                             value of 10.
 12                                      User is prompted for
    1    3 -                             values for points of
    .                                    the image with row
    .                                    and column numbers.
    .
    1   20 -
 10

 IMAGE - 1     BLOCK - 2
    1    1 -
 60
    .
    .
    .
    1   20 -
 54
    .
    .
    .
```

```
IMAGE - 1      BLOCK - 10
   1    1 -
50
   .
   .
   .
   1   20 -
45

A                                          Subsystem prompt.

blk                                        User requests to see
                                           the values of the
  ENTER FIRST<,LAST>?                      image as a check of
                                           the input.
1,10                                       Blocks 1-10.

  ***FILE PICNEW *** IMAGE 1
          *** COLUMNS 1 - 20 ***

  --- BLOCK 1 ---

  10   12   . . .

  --- BLOCK 2 ---

  60   . . .                               54
       .
       .
       .

  --- BLOCK 10 ---

  54   . . .

A

done                                       Close the file and
                                           return to KAND01.

#:

stop                                       Exit KANDIDATS.

R;                                         CMS command prompt.

compress picnew sif                        Compress image file
                                           just created.

R;

fmsave picnew sif                          Save that file on
                                           the file manager space.
```

```
R;

logoff                                          User ends CMS
                                                session.

CHECKPOINTING . . .

CONNECT = 00:17:05  TOTCPU = 000:00.75
CONNECT COST $.  CPU COST $.  I/O COST $ .
TOTAL COST $.    ACCOUNT BALANCE $ .
LOGOFF AT 16:19:22  CST FRIDAY 02/26/78
```

7. SUMMARY AND EVALUATION

This report has presented a description of the changes made to KANDIDATS in transporting the system from the University of Kansas PDP-15 to the Kansas State University IBM-370. A description of the KANDIDATS image processing system as it was implemented on the 370 was included in the form of a user's guide and a system programmer's guide.

This chapter will evaluate the KANDIDATS system as a tool for the user wishing to work with digitized multi-image files. It will also consider the KANDIDATS system from the aspect of the ease with which the system can be ported to other machines and operating systems.

7.1 Summary of the System

KANDIDATS is a tool for the user wishing to work with digitized multi-images. The KANDIDATS system provides the user with the ability to perform standard functions on images without an extensive knowledge of programming. In evaluating the adequacy of the system as a tool, two characteristics must be considered: the functional completeness and ease of use of the system.

The functional completeness of the system is its capability to provide the needed functions for processing an image. Through KANDIDATS an image can be created, displayed, combined with other images, and broken down. The history of the processing performed on an image file is automatically maintained for the file. Characteristics of the ground represented by an image may also be kept in the file. The mode of the image may be changed. The image may be quantized (the gray tones of the image mapped onto a finite set of gray tones). KANDIDATS can expand or compress an image. Images may be transformed with gradient operations, spatial clustering and convoluted.

The inexperienced user of computers who has a need or desire to work with digitized images may learn to use the KANDIDATS system. There are several characteristics of the system which make it simple to use. First, the system has a standardized command format. The name of the function is given, followed by the destination if any, an arrow, and finally any sources required. Any additional information needed to process a function is requested from the user by the function. Secondly, there are several commands that can be used by the inexperienced user to gain more information about the use of the KANDIDATS system from the

system itself. These commands are: MESG, SVOC, VOCA, and EXPL. MESG provides general information about the system. SVOC lists the available commands. VOCA gives a brief description of each of these commands while EXPL provides a more detailed description. Finally, KANDIDATS checks input from the user for valid values. If an error does occur, the user is provided with possible correct values for repeating the input.

7.2 Evaluation of Portability

Portability is the ability to move a software system from one hardware system to another. Several characteristics of KANDIDATS make it a portable system. First, KANDIDATS is written largely in FORTRAN. FORTRAN is widely implemented on computer systems and is a highly standardized language. The major differences between the FORTRAN syntax on the PDP machine and on the IBM-370 are the decrementing DO-loop, the DOUBLE INTEGER type, the use of character string constants and alternate error returns. KANDIDATS does have a few routines which must be written in assembler language. These routines had to be rewritten for the 370. The assembler routines, for the most part, interface with the operating system.

Secondly, KANDIDATS is a highly modular system which increases its portability. Those parts of the system that will not be used on a different machine may be eliminated from the system by stabbing out the necessary routines. For example, the tape routines were not needed for the 370.

The modularity of the system ties in with the third aspect of KANDIDATS that increases the portability of the system. That is, the general structure and readability of the code. Because the code is generally well structured and documented, it is easier to follow and alter. One notable exception to this was the subsystem EXSIF. EXSIF used assigned GOTO's and was largely undocumented.

To make the system more portable, a labeled common area, MACH, was added to KANDIDATS. MACH contains those variables that have machine dependent values. For example, the number of bits per machine

word is in MACH. Calculations that require these values reference the variable rather than the constant. When the system is ported to a new machine, only the values assigned to the variables of MACH must be changed in the block data routine. Another change that would improve the portability of KANDIDATS is to put variables of different types in different common blocks.

The scope of the project of porting KANDIDATS in volume and time is as follows. Approximately 14,000 lines of code were ported to the 370. Of this, approximately 4,000 lines were written specifically for the 370. About 4,000 lines of the remainder actually had to be changed. The final object size of the system is almost 179,400 bytes. Learning the system required about 100 man/hours. The actual changing and testing of the code required 900 man/hours. Documenting the porting of the system encompassed 200 man/hours.

It is recommended that to make KANDIDATS more easily ported to other systems, two things should be done. First, test procedures should be written that can be used to verify the accuracy of the transported routines. Secondly, the code should be changed to ANSII standard FORTRAN. Non-standard parts can be recognized by the PFORT verifier programs [9] which is available at the University of Kansas and at Kansas State University.

BIBLIOGRAPHY

[1] Bryant, William, Robert M. Haralick, Dale Johnson, Gary Minden, Craig Paul, and Amrendra Singh, "KANDIDATS II: KANsas Digital Image DATa System," Technical Report 0920-2, The University of Kansas Center for Research, Inc., September 1976.

[2] IBM Virtual Machine Facility/370: CMS User's Guide, 1976, C 20-1819.

[3] IBM Virtual Machine Facility/370: CMS Command and Macro Reference, 1976, C 20-1818.

[4] Kansas State University Computing Center, CP/CMS Guide, January 1977.

[5] Haralick, Robert M., "Glossary and Index to Remote Sensed Image Pattern Recognition Concepts," Pattern Recognition, Vol. 5, 1973, pp. 391-403.

[6] Kansas State University Computing Center, Technical Bulletin 22, "RREAD and Core for OS/360 FORTRAN IV," July 26, 1971.

[7] Kansas State University Computing Center, "Program Descriptions: DATE, GENDATE, GENGREG, INTIME, TIME," November 1972.

[8] White, A.B., Video Imaging on the Plasma Display Panel, R-677, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, April 1974.

[9] Ryder, B., The PFORT Verifier, Software Practice and Experience, 1974, pp. 359-377.

KANDIDATS:
The Porting of an Image Processing System

Linda J. Lallement
B.S., Kansas State University
1975

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the
requirements for the degree
MASTER OF SCIENCE
Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas
1978

ABSTRACT

This report describes the transporting of the KANDIDATS system from the PDP-15 at the University of Kansas to the IBM-370 at Kansas State University. KANDIDATS is a computerized system that interacts with the user to process digitized multi-images. It was developed by the Image Processing Group of the Remote Sensing Laboratory, University of Kansas. KANDIDATS was transported to the IBM-370 and runs under the Conversational Monitoring System (CMS). It is written in FORTRAN and 370/Assembler. Included in this report are a user's guide and programmer's guide for KANDIDATS as it exists at Kansas State University and a description of the changes made to the system. The changes made to KANDIDATS involve FORTRAN syntactical changes, the writing of the assembler routines, changes due to differences in the two machines, and the addition of new routines.