# MIKADOS

## System Operation and Utility Guide

Dansk Data Elektronik ApS
26 august 1981

Author: Claus Tøndering

Table of Contents

## 1. Introduction.

This manual describes the basic facilities and concepts of the MIKADOS operating system.

The MIKADOS system runs on the SPC/1 and ID-7000 micro computers of Dansk Data Elektronik ApS.

The acronym MIKADOS is a concatenation of MIK and DOS (with an A inserted for ease of pronunciation). MIK (MIkrodatamatorienteret Korutinestyresystem - Danish for: Micro Computer Oriented Coroutine Operating System) is the name of a small real-time process monitor created by Bodil Schrøder at the university of Copenhagen in the mid 1970s. This monitor has been expanded and enhanced by Dansk Data Elektronik so that it now functions as a Disk Operating System (hence: DOS).

The version of MIKADOS described herein is dated 14.07.1981. Older or newer versions may show deviations from this system; however, these deviation will normally be of minor importance.

The purpose of this manual is

- to provide the MIKADOS user with a knowledge of how to run the various MIKADOS utility programs.

- to provide the Pascal and assembly language programmer with the knowledge necessary to take advantage of the facilities of MIKADOS. The assembly language programmer should, however, also acquaint himself with the MIKADOS Advanced Programming Guide.

Dansk Data Elektronik ApS reserves the right to change the specifications in this manual without warning. Dansk Data Elektronik ApS is not responsible for the effects of typographical errors and other inaccuracies in this manual, and cannot be held liable for the effects of the implementation and use of the structures described herein.

## 2. Basic Concepts.

The MIKADOS operating system is a multi-tasking multi user system. This implies that several users may run on the same computer simultaneously, and that one user may have several programs running simultaneously.

### 2.1. Memory layout of a single-user system.

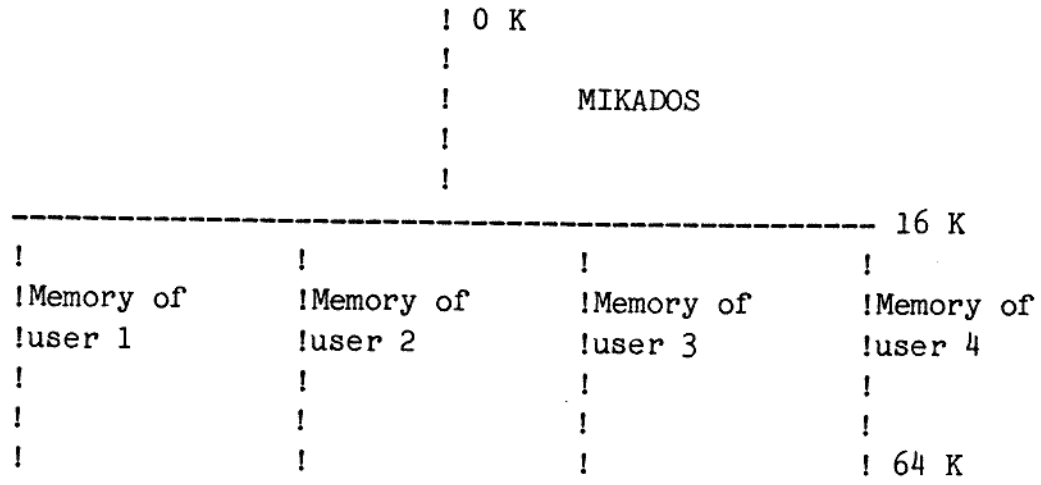
The 8085 micro processor, which is the processor used in the SPC/1 computer, has an address space of 64 K bytes. This means that the processor can have at most 64 K bytes of memory attached to it. This is the case in the single-user system. The memory is devided thus:

0K - 16K: MIKADOS.
16K and up: User program area.

### 2.2. Memory layout of a multi-user system.

In a multi-user system the first 16K of memory containing the MIKADOS operating system is common to all users.

The remaing 48K of address space memory is divided into several so-called banks, so that memory addresses 16K-64K of user number 1 are different from memory addresses 16K-64K of user number 2 etc. This creates a tree-structure in memory:




Thus each user has access to the MIKADOS system but not to each other's memory.

## 2.3. The MIKADOS bank.

In general 16K of memory is not sufficient to hold the operating system. Therefore an extra memory bank of 16K is normally installed in parallel to the user memory banks. This is the so-called MIKADOS-bank; it contains parts of MIKADOS that need not lie in the root of memory.

## 2.4. Running a program.

When a program is started on the computer it is assigned to a memory bank. This is normally the bank of the user starting the program, but it need not be so. In this bank the program takes up a certain amount of memory, which is then reserved for that program until it terminates. No other program is allowed to run in the same memory locations of that bank until the first program is terminated.

An attempt to start another program in the same memory locations of the same bank is allowed, but the program start request will be queued, and not until the first program terminates will the second program start.

## 2.5. Background Banks.

It is possible to generate a MIKADOS system supporting memory banks that do not have terminals attached to them. Such so-called background memory banks may, for example, be used to execute programs that do not communicate much with the issuing terminal. In this way the terminal may be freed to run other programs.

## 3. System Startup.

Each SPC/1 computer is equipped with one or more minidiskette drives. These drives, apart from being available in a running MIKADOS system, are used for bootstrapping the operating system.

When the computer is switched on the minidiskette drives should be empty. Then a minidiskette labelled 'SYSTEM DISC' is inserted into one of the minidiskette drives, and this will cause the MIKADOS system to be loaded into the computer.

This system disk, also called the boot disk, may be a 90 K, a 280 K, or a 560 K minidiskette. However, on older computers only 90 K minidiskettes may be used.

Before the operating system is loaded the computer memory is tested. A printout of the available memory is shown on one of the terminals connected to the system. Only if no errors are found in memory will the MIKADOS operating system be loaded.

The system disk contains three files: MIKM, LOAD2, and DDEANSYS, each of type 0 (for a discussion of the file system see section 6). The first two files contain the programs that test the memory, the last file contains the operating system itself.

## 4. Operator Communication.

The operator communication module of the MIKADOS operating system enables the console operator to start the execution of named programs. The operator may pass parameters to the programs started from the console. The operator may also break (abort) running programs.

Before entering a command, the operator must press the key labelled 'ESC' or 'ESCAPE'. The MIKADOS system responds by showing a '>' as shown below:

```
>
```

The operator may now enter the desired command.

### 4.1. The Program Start Command.

This command instructs the system to load into memory and start execution of a named program. Optionally the command may contain a parameter string to be passed to the program. The format of the command is as follows (the parentheses are not part of the command, but indicate that the text within them is optional):

```
>programname(:Pn(:type))(*bank)(,parameters)
```

where

programname is the name of the program to be executed. If the program name contains small letters these are converted to capitals; this, however, is not true for older versions of MIKADOS.

Pn (optional) is the disk identification (see section 6.2) of the disk where the program module can be found. If this parameter is not specified the system assumes that the program module should be loaded from the system disk (P1). Note that Pn must be specified if type is specified. The P may be given as a small letter; this, however, is not true for older versions of MIKADOS.

type (optional) is the type of the program file. This will in most cases be 1. If type is omitted, 1 is assumed.

bank (optional) is the number of the bank where the program should run. If bank is not specified, the bank associated with the issuing terminal is assumed.

Even though the program may be executed in the memory bank of another user, the program will allways communicate with the issuing terminal.

Note: A user may have programs in different banks each communicating with the same terminal. If, however, one of these programs runs in direct input mode (see the MIKADOS Advanced Programming Guide section 4.2.5 or the Pascal Standard Assembler Package manual), none of the other programs should request terminal input.

Earlier versions of MIKADOS do not support this facility.

parameters (optional) is a string that is passed unchanged to the new program using a dynamic memory area. This implies that no translation of small letters into capitals take place here; many programs, however, require that the parameters be given as capital letters.

Examples:

The operator command

```
>ASM,PROGR,L,X
```

initiates execution of the ASM program of type 1 residing on disk P1. The program will run in the bank of the issuing terminal, and the parameter string passed to the ASM program is 'PROGR,L,X'.

The operator command

```
>ASM:P2*3,PROGR,L,X
```

works as the above command, except that ASM is loaded from disk P2 and run in memory bank number 3.

## 4.2. The Break Command.

This command breaks the execution of a running program, provided that the program tests the so-called break flag.

The format of the command is

```
>.BR
or
>.br
```

The latter form is not available in older versions of MIKADOS.

Associated with each terminal in the system is a "break flag". The value of this flag is normally zero, but each time the user enters the break command, the value of the flag is increased by one.

It is the duty of the running program to inquire into the value of the break flag. In assembly language this may be done using the BRKTS routine. In Pascal this may be done by testing the value of EOF(OUTPUT).

The execution of a Pascal program will stop unconditionally if the value of the break flag is 2 (i.e. the user has entered >.BR twice).

Note: Pressing a function key on terminals having such, sets the break flag to the value of the key depressed. Thus a running program that tests the value of the break flag will be aborted if a function key has been pressed. Pressing a function key also disables the unconditional stop of Pascal programs on a double >.BR.

## 4.3. The Status Command.

This command is available only on systems with a MIKADOS bank installed (see section 2.3). It is not available on older MIKADOS sytem versions.

## 4.3 The Status Command

In Mikados versions dated 01.10.1982 the status command no longer exists. Trying to issue an >.ST will result in the error message 'ILLEGAL COMMAND' from the Mikados system.

The function performed by the status command is instead implemented in the program ST situated on the system diskette. If the system diskette is disk no. 3 the ST program is activated by

```
>ST:P3
```

The program ST may be copied to P1 (or any other disk). As ST is a Pascal program both ST of type 1 and P should be copied.

The format of the command is

>.ST
or
>.st

This command outputs a description of the hardware configuration supported by the system. Information about the operating system itself is also given.

# 5. The Input/Output System.

This chapter describes how to handle I/O to and from terminals and printers. I/O on disk files is described in chapter 6.

## 5.1. The I/O devices.

Input is taken from the keyboard of one of the terminals attached to the computer.

Output is directed to the screen of one of the terminals or to one of the printers.

When a program is started it is assigned to the terminal from which it was started. Thus terminal I/O normally takes place with that terminal unless unorthodox ways of making I/O are used. Thus we may simplify our ideas by saying:

Input is always taken from 'the' terminal keyboard.

Output is directed to one of the physical output devices:

- 'The' terminal screen.
- One of the printers.

From the program the user does not directly access the physical I/O devices, but rather has access to three logical devices:

- The console device.
- The list device.
- The error device.

Pascal programs cannot access the error device.

Assembly language I/O is described in detail in the MIKADOS Advanced Programming Guide.

In Pascal an input statement like

```
READ(xxxxx) or READ(INPUT,xxxxx)
```

takes input from the console device. An output statement like

WRITE(xxxxx) or WRITE(OUTPUT,xxxxx)

directs output to the console device, whereas

WRITE(LIST,xxxxx)

directs output to the list device.

The logical I/O devices are assigned to the physical I/O devices in the following manner:

If a program is started thus:

>PROG

its console, list, and error devices will all be the terminal.

If a program is started thus:

>L,PROG

its console and error devices will be the terminal, but the list device will be the printer (in a multi-printer system it will be printer number 1).

If a program is started thus:

>A,PROG

its console device will be the terminal, but the list and error devices will be the printer (in a multi-printer system it will be printer number 1).

In a multi-printer system L and A may be replaced by L1, L2, L3, L4, or A1, A2, A3, A4 for printer number 1, 2, 3, and 4, respectively.

Note: L and A are themselves programs.

A type and bank code may not be entered on a program started using the L or A programs. The program started by the L or A programs will run in the same bank as that program.

closed within the same < and >. Thus in order to erase the screen and position the cursor to column 10 line 20 and write the text ´BEER´ there, the following Pascal statement should be used: WRITE(´<XC1020>BEER´). If the statement WRITE(´<X><C1020>BEER´) were used, the screen would be erased and the text ´<C1020>BEER´ would be output in the upper left corner. Or, as another example, to increase the line distance on a Qume printer by 1/8" the Pascal statement WRITE(LIST,´<VVVVVVVVVVVV>´) (there are 12 Vs) should be used.

If a command string contains characters that do not apply to the specific output media, these characters are ignored.

To output the character ´<´ be sure to include an empty control sequence in the buffer. WRITE(´<´) will output nothing, whereas WRITE(´<><´) will output ´<´.

Note that many of these control sequences are unnecessary when writing Pascal programs. The <S> command may be replaced by the WRITE procedure, the <Cxxyy> command may be replaced by the GOTOXY procedure, the <X> command may be replaced by the CLEARSCREEN or the PAGE procedure. In general it improves program clarity to use these standard procedues instead of the control sequences.

### 5.2.1. Console Driver.

The MIKADOS console driver supports a variety of terminals, the main terminal line however is a set of Lear Siegler compatible terminals, such as Soroc 120, Soroc 140, Televideo 920C, or DTI Genius 7.1.

On the above terminals an output string may contain characters with bit 7 set to 1 or 0. The character will be output in normal video if bit 7 is 0; if, however, bit 7 is 1 the character will be output in inverse or low video, depending on the terminal type.

The output string may start with a control character sequence as described above.

The control character sequences are:

<Cxxyy> Position cursor to (xx,yy). xx and yy must be given as

two-digit numbers.
Example: <C4012> means position cursor to column 40 line 12.

<D> Move cursor to bottom left hand corner of screen.

<S> Omit the final line feed and carriage return after outputting the buffer.

<U> Move cursor up one line.

<X> Erase screen.

<Z> Erase to end of line.

### 5.2.2. Printer Driver.

The MIKADOS printer driver currently supports the following printer types:

Serial matrix printers: Anadex DP8000, Swiss Matrix, Tally MT1602, Itoh 8300R, Facit 4540, and Texas 810 and 825 (other serial matrix printers may be used).

Serial daisy wheel printers: Diablo 630 (tractor, friction, or sheet feed), Tec FP-1500-25R.

Parallel daisy wheel printers: Qume Sprint 3 (with tractor or sheet feed).

Parallel line printer: Data 100 (other Centronics compatible parallel line printers may be used).

The printer drivers, except for the Qume 3 driver, supervise the time spent waiting for an output operation to be completed. If an output request has not been completed after 30 seconds, an error message will be output on the terminal from where the user program was initiated. The printer driver automatically seeks to perform the output operation again, and the error message is removed when the operation is completed.

The output string may start with a control character sequence as described above.

The control character sequences are (control sequences are only valid for the printers mentioned explicitly):

Serial Matrix Priters:

| | |
|---|---|
| <E> | Eject page. |
| <Fnn> | Set page size to nn lines, where nn is an integer, not necessarily of two digits (valid only on Texas 810 and 825 printers). |
| <G> | Start expanded output (not valid on Texas 810 printers). |
| <I> | End expanded output (not valid on Texas 810 printers). |
| <J> | End compressed output (not valid on Anadex, Facit, or Swiss printers). |
| <K> | Start compressed output (not valid on Anadex, Facit, or Swiss printers). |
| <N> | No final line feed is output after printing the buffer. Thus a subsequent output operation will overprint the string containing the <N>. |
| <S> | Omit the final line feed and carriage return after printing the buffer. |
| <T> | Horizontal tab. |
| <X> | Eject page (same as <E>). |
| <Y> | Vertical tab. |

Serial Daisy Wheel Printer:

| | |
|---|---|
| <A> | Start proportional spacing (only Diablo printers). |
| <B> | End proportional spacing (only Diablo printers). |

| | |
|---|---|
| <E> | Eject page. |
| <Fnn> | Set page size to nn lines, where nn is an integer, not necessarily of two digits. |
| <Hnn> | Set character width to nn/120 ", where nn is an integer, not necessarily of two digits. |
| <Lnn> | Set left margin to column number nn, where nn is an integer, not necessarily of two digits. |
| <N> | No final line feed is output after printing the buffer. Thus a subsequent output operation will overprint the string containing the <N>. |
| <Pnn> | Set pica size to nn, where nn is 10, 12, or 15. This gives a printout with 6 lines per inch, 72 lines per page and 10, 12, or 15 characters per inch, respectively. |
| <Q> | Start automatic justification of right margin. This justification will be removed by the <W> or <*> commands. (Only Diablo printers.) |
| <Rnn> | Set right marign to column number nn, where nn is an integer, not necessarily of two digits. |
| <S> | Omit the final line feed and carriage return after printing the buffer. |
| <Vnn> | Set line height to nn/48 ", where nn is an integer, not necessarily of two digits. |
| <W> | Terminate all Word Processing options (only Diablo printers). |
| <X> | Eject page (same as <E>). |
| <+> | Start underline output. |
| <-> | End underline output. |

Note that + and - have to be part of the initial control character sequence. In order to print

This has been underlined

the follwing three strings should be output:

<S>This has been
<+S>under
<->lined

<*> Reset the printer (only Diablo printers).

For a detailed description of the concepts of proportional spacing, right justification, Word Processing options, and printer reset, the reader is referred to the Diablo printer manual.

The initial setting of, for example, pica size is set by switches on the Diablo printer.

Parallel Daisy Wheel Printer:

<E> Eject page.

<H> Increment character width by 1/120 ".

<h> Decrement character width by 1/120 " if possible.

<M> Move left margin one character to the right.

<m> Move left margin one character to the left if possible.

<N> No final line feed is output after printing the buffer. Thus a subsequent output operation will overprint the string containing the <N>.

<S> Omit the final line feed and carriage return after printing the buffer.

<V> Increment line height by 1/96 ".

<v> Decrement line height by 1/96 " if possible.

Note: The resolution of the line height is 1/48 ", wherefore only every other <V> or <v> has any effect.

<X> Eject page (same as <E>).

The initial setting is 12 characters per inch, 6 lines per inch, and a 12 " page length.

If any character in the output buffer has bit 7 set to 1, that character is printed underlined.

Parallel Line Printer:

<E> Eject page.

<G> Expanded output.

<S> Omit the final line feed and carriage return after printing the buffer.

<X> Eject page (same as <E>).

<Y> Vertical tab.

## 5.3. Input line editing.

When the user is typing on the terminal several commands are available to edit the input string. These commands are:

RETURN line feed and carriage return is echoed back to the terminal. The input operation is terminated.

ESC same as RETURN. The user program may check if the input operation was terminated by RETURN or ESC. In assembly language programs this is checked by inspecting bit 4 of the completion code, in Pascal programs this is checked using the EOF(INPUT) function.

Left-arrow The cursor is moved one character position left if possible, allowing the user to rewrite a possibly erroneous character.

| | |
|---|---|
| Right-arrow | The cursor is moved one character position right. |
| Up-arrow | A space is inserted at the cursor position. |
| Down-arrow | The character at the cursor position is deleted. |
| HOME | All characters to the right of the cursor are deleted (replaced by blanks). |
| RUBOUT or DELETE | All characters in the line are deleted (replaced by blanks). The cursor is moved to the start of the line. |
| TAB | The cursor is advanced to the next relative buffer position divisible by 8. |

Note that if the input item is a character string (which is always the case in an assembly language program) the length of the string does not include any blank chacters to the right of the last printable character in the buffer. If however the input buffer consists only of blanks, the length of the string is assumed to be the character position at the time the RETURN key was pressed.

## 5.4. Printer reservation.

As mentioned in section 5.1 the L and A programs will reserve the printer for the user and assign the list and possibly the error device to the printer.

The user may wish to reserve and release the printer dynamically during program execution. This can be done using the subroutines mentioned in the following sections.

### 5.4.1. Pascal Printer Reservation Subroutines.

These subroutines are all assembly language routines that should be linked to the Pascal interpreter when they are used. Please refer to the MIKADOS Pascal User´s Guide for information about how this is done.

#### 5.4.1.1. LISTPR

The function LISTPR returns the number of the printer currently reserved by the calling program. The value zero is returned if no printer has been reserved.

The function should be declared in the following manner in the Pascal program:

```
FUNCTION LISTPR: INTEGER;
EXTERNAL;
```

#### 5.4.1.2. GETPR

The function GETPR reserves a printer whose number is specified as a parameter. The value TRUE is returned if the reservation was successful. The value FALSE is returned if the reservation for some reason could not be made. Normally the reason will, of course, be that the printer is reserved by another user, but error conditions such as an illegal printer number are also possible. If the reservation of the printer is successful the list device will be assigned to that printer.

The function should be declared in the following manner in the Pascal program:

```
FUNCTION GETPR(PRINTER: INTEGER): BOOLEAN;
EXTERNAL;
```

#### 5.4.1.3. FREEPR

The procedure FREEPR releases the printer - if any - reserved by the calling program and assigns the list device to the terminal.

The procedure should be declared in the following manner in the Pascal program:

```
PROCEDURE FREEPR;
EXTERNAL;
```

### 5.4.2. Assembly Language Printer Reservation Subroutines.

See the MIKADOS Advanced Programming Guide section 9.1.

# 6. General Remarks About the File System.

The MIKADOS file system gives the user easy access to disk storage on the computer.

This chapter contains an introduction to the MIKADOS file system, describing the general disk layout. For a description on how to use the file system, the Pascal programmer is referred to the Pascal User´s Guide, whereas the assembly language programmer is referred to the MIKADOS Advanced Programming Guide.

## 6.1. File System Structure.

The information that can be handled by the MIKADOS system is stored on disk drives. There may be one or more disk drives in a system, and disk drives of different types may be intermixed.

A disk drive contains one or more physical disks. The disks, some of which may be removable, are used for storing information. The physical disks are mapped onto logical disks during system generation. A logical disk is a disk seen from the programmer´s point of view. Normally, physical and logical disks are identical, but an installation may choose to subdivide a physical disk into several logical disks. Currently it is not possible to combine physical disks into one logical disk.

A logical disk (hereafter referred to as ´a disk´) consists of a file catalog and a file area. The maximum number of files that can be accomodated in the file catalog is specified during system generation. A special scheme (hashing) ensures fast file lookup even if the catalog is nearly full.

A file consists of a primary file, which is the original file constructed by the create operation, plus 0 to 60 extents. All extents are of the same size as the original file. Any attempt to extend a file past the 60th extent is rejected.

The size of a file is specified in sectors. The user must be aware that a sector as seen from the MIKADOS system always consists of 256 bytes even though the hardware manuals may say otherwise. Further, the

user should note that the first 32 bytes of the first sector of the primary file and of each extent are used for system information and not accessible to the user. Except for its effect on the useable file size this restriction is transparent to the user.

The first sector of every disk is called the label sector. In this sector various information about the disk is stored. Included in this information are the so-called disk label and disk type. These two items are character strings of 10 and 5 characters, respectively. They do not play any role in the file system, but various utility programs may inspect and modify them. In Comal a disk is identified by its label. The user may use these strings to give a name to the disk.

## 6.2. File Identification.

A file is identified by a file name, a file type code, and a disk identification. Although any character is allowed in the file name, many system programs require that only capital letters and digits be used. It is good programming practice to follow this convention. The file name is supplemented by a file type code which must be an alphanumeric character. The file type code is used together with the file name to uniquely identify a file on a disk, that is, several files may have the same file name but different type codes (for example, source file, relocatable file, and executable program file). Except for type code N the user may select his type codes freely. However, certain systems programs (for example, editor and assembler) assume that certain file types are assigned specific type codes, wherefore the following conventions are used:

| | |
|---|---|
| K | - source text. |
| R | - relocatable program. |
| digit | - executable (linked) program. |
| P | - Pascal P-code file. |
| B | - Comal binary file. |
| T | - Word processing tab and margin file. |
| N | - Nullfiles (see section 6.3). |

The disk identification is used to point out the logical disk where the file is located. The disk identification always has the form 'Px' where x is an alphanumeric character. The legal disk identifications vary among installations and are defined at system generation time.

Note that two files with the same name and type may exist on different disks in the same system.

## 6.3. Nullfiles.

A special type of files are the so-called nullfiles. A nullfile is a file that contains nothing but a 16 bit number.

Nullfiles take up no space in the file area of the disk.

Only assembly language programmers can use nullfiles, and a discussion of the subject is given in the MIKADOS Advanced Programming Guide.

## 6.4. Fixed and Variable Length Record Files.

The file system supports two different record structures within files: fixed length records and variable length records. A file may not contain records of both types.

In a fixed length record file the record length is defined when the file is opened. All records in the file have the same length. Record boundaries are not recorded in the file but are determined using only the specified record length. The main advantage of this record structure is that it permits direct and fast access to any record in the file using the position file subroutine. In special applications the assembly language programmer may take advantage of the fact that the record length may vary from one open operation to the next, and that the record length may be altered in the DCB (Data Control Block, see the MIKADOS Advanced Programming Guide) while the file is open. These features should be used with great care. The record length must not exceed 2560 bytes.

In Pascal fixed length record files are declared as FILE OF xxx, where xxx is a type other than CHAR or any equivalent of CHAR. A file is positioned to a specific record using the SEEK procedure, and I/O is performed using the GET and PUT procedures.

In a variable length record file each record carries information about

its own length. The length indicators require a 2-byte overhead per record in the file, as there is an initial and final byte, each specifying the length of the record. Variable length record files must be read or written sequentially, that is, access to record no. n is possible only after reading the preceding n-1 records. The main advantage of this file type is a better utilization of disk space because no record occupies unneccessary space. The assembly language programmer should note that the 'record length' parameter needed by certain file system subroutines should be set to any nonzero value when operating on variable length record files. The length of any record in the file must not exceed 136 bytes.

In Pascal variable length record files are declared as TEXT or FILE OF CHAR. I/O is performed using the READ, READLN, WRITE, and WRITELN procedures.

A variable length record in a file corresponds to a line on the terminal or printer.

# 7. File System Utility Programs.

This chapter describes a number of file system utility programs used to perform various functions in the MIKADOS file system.

## 7.0.1. General Remarks.

All file system utility programs are conversational. Entering ESC (escape) as a response to a question asked by a file system utility program will terminate execution of that program immediately.

## 7.0.2. Specifying a Master File Identification.

Several file system utility programs require the specification of a master file identification (file name and type) to identify one or more files on a disk. The comparison between the master file identification and an actual file name and type takes place on a character-by-character basis:

1. a '*' matches any character.

2. a lower case alphabetic character matches any character except the corresponding upper case character. This works only if the file name specification contains at least one '*'. It may not be used as a type specification.

3. any character not covered by the above rules matches only that particular character.

A match between the master file identification and an actual file identification occurs if rules 1 and 3 are not violated by any of the 9 characters in the file identification, and if rule 2 when appropriate is not violated in at least one case.

Examples:

| Master file name | type | result of comparison |
|---|---|---|
| ******** | * | all files match |

| | | |
|---|---|---|
| LINK | 0 | only file LINK of type 0 matches (if it exists) |
| LIN***** | * | all files whose name start with 'LIN' match |
| lin***** | * | all files except those whose name start with 'LIN' match |
| Ab*D**g | R | certain relocatable files match (note: The 8th character of master file name is a blank), e.g. ABCD - matches<br>ABCDEFG - does not match<br>AHCDEFG - matches<br>AHCDEFGH - does not match |

## 7.0.3. File System Error Codes.

An error code will be reported by the utility program, if a file system error occurs. The error codes are listed in Appendix A.

## 7.1. PLINI - Initialize a Disk.

This program is used to initialize (clear) a new disk cartridge or floppy disk. The program initializes and optionally reads back all disk sectors and returns an appropriate error message if one or more sectors are found to be defective.

The program may be used to remove all files from a disk, and to change the disk label and type.

Calling sequence (user input is shown underlined):

```
>PLINI
Enter disk drive identification P2
Address of first available sector is: 0022 07
Enter/change disk label SYSTEM5.2
Enter/change disk type SYST*
Enter/change date of last back-up 17.06.1981
Label sector output to disk. Initialization desired (Y/N)? Y
Read-back desired (Y/N)? N
```

The answer to the first question identifies the physical disk drive (P2 in the above example) whose cartridge/floppy disk is to be initialized.

The following steps in the conversation permit the user to initialize/change the label, type and date of last back-up in the label sector (track 0, sector 0) of the disk. Note that the COMAL system disregards the last two characters of the label (characters 9 and 10). Also note that the 5th and last character of the type should be a '*' if the disk is a master disk and '0' if the disk is a back-up disk (required by PKOPI). The date entered in response to the 4th question is not checked for validity by PLINI.

If the user answers No to the 5th question, only the specified label changes are output to the disk after which PLINI terminates. If the user answers Yes, the disk is initialized (cleared). Read-back may be quite time consuming, and is recommended only if the user suspects the disk of being defective. If the specified disk is on a Pertec disk drive, readback is performed automatically by the MIKADOS disk driver. Requesting PLINI to read-back in this case is thus merely a waste of time.

The user may stop an ongoing initialization at any time by entering a >.BR command on the console from which PLINI was started.

## 7.2. KATLG - Output Disk Catalog.

Outputs the catalog and label information of a specified disk to the list device.

Calling sequence (user input is shown underlined):

```
>KATLG
Enter disc identification: P2
Options available:
                    O - ordinary (all base files)
                    F - full (base files plus extents)
                    X - extended (as F, plus purged files)
                    N - nullfiles only
                    Q - specific files only
                    S - label information only
                    C - catalog consistency check
Enter option: F
```

or

>KATLG,Px(,option)

where Px is replaced by the identification of a disk drive and 'option' is replaced by one of the possibilities mentioned in the above menu (O, F, X, N, Q, S, or C). If no option is specified, 'O' (ordinary) is assumed.

The options have the following meaning:

O - ordinary; for each file the following is listed: file name, file type, extent code (always A with this option), track and sector address of first sector in base file, number of sectors in base file (= number of sectors in each extent), record length specified when file was created, and number of file extents.

F - full; same as 'O', except that file extents are also listed (number of file extents is not shown for file extents).

X - extended; same as 'F', except that purged files are also listed (first character in file name replaced by '*') and the so-called catalog group number for the file catalog entry is listed.

N - nullfiles; for each nullfile the following information is listed: nullfile name, and nullfile information (address).

Q - question; print information about files matching a specified master file name and type (see section 7.0.2). After selection of this option the program starts the following conversation (sample user responses shown underlined):

```
Enter master file name: P*******
Enter master file type: K
Enter extent code (usually blank): _
Enter starting track number: 0000
```

If no extent code is entered, information about extents is not printed. Otherwise information about all extents with an extent code greater than or equal to the specified character is output. The primary file has extent code A, the first extent has extent code B, and so on.

The starting track number indicates the lowest starting address for file and file extents for which information is shown.

S - short form; with this option only label information is output.

C - check catalog; with this option it is possible to check that every file in the catalog does indeed exist on the disk and that every file on the disk has an entry in the catalog. The prime purpose of this is to check and maybe restore a possibly corrupted disk. After selection of this option the program starts the following conversation (sample user responses shown underlined):

```
Enter M-ain catalog, F-ile chain or B-oth: B
Enter R-eport only or A-sk for corrective action: R
```

The first question requires the operator to specify which of the two possible checks should be carried out: Checking that every entry in the catalog corresponds to an existing file and/or checking that every file has an entry in the catalog.

The second question requires the operator to specify whether only an error report should be generated or the program should ask for corrective action when it encounters an error. The corrective action will normally be either deleting the catalog entry/the file or creating a catalog entry.

Except when the N or C option has been specified, the program outputs label information for the disk in question, after a request has been satisfied. The information is output using the following format:

```
First unused track/sector: 0022 0008. Number of unused sectors: 00145.
Disk type: PLA1*. Date of last backup: 17.06.1981. Disk label: PP1      .
Number of purged sectors: 00020.
```

The number of purged sectors is output only if a complete scan of the

whole disk has been performed, that is, only in connection with options O, F, X, and Q.

Information about nullfiles is output only with the N option. The O, F, X, and Q options will not print any nullfile information.

The user may stop an ongoing catalog listing at any time by entering a >.BR command on the console from which KATLG was started.

## 7.3. PKOPI - Disk Copy.

Copies all MIKADOS file information from a source disk to a destination disk. The source and destination disks must be of the same type, for example two 90 K mini floppy disks (otherwise FCOPY should be used).

Calling sequence (user input is shown underlined):

```
>PKOPI
Source disk: P1
Destination disk: P5
```

Before copying starts the user is asked to confirm the request:

```
Source disk type: DIS1*. Destination disk type: DIS10
Do you still want copying? Y/N Y
```

If the first four characters of the source and destination disk types (see section 7.1) are not identical, a warning will be output ('Disk labels do not match'). If the both disks are master disks (5th character of type is a '*'), a warning will be output ('Destination disk should not be copied onto'). If the source disk is a backup disk and the destination disk is a master disk, and the disk types are otherwise identical, a warning will be output ('WARNING - attempt to copy from backup onto original'). The user may choose to ignore the warnings and force a copy despite the warnings.

The type of the destination disk is not changed by PKOPI. The date of the last backup is changed on both the source and the destination disks to the current MIKADOS date.

Note that some of the information in the label sector (track 0, sector

0) of the destination disk is not changed by PKOPI. Also note that in order to save time the unused parts of a disk are not copied. This means that a disk which has not been initialized by the MIKADOS system may not be correctly copied.

After a copy has been completed the user may initiate a new copy.

It is recommended to write protect the disk drive where the source disk (the disk copied from) is inserted as an extra precautionary measure.

The user may stop an ongoing copy at any time by entering a >.BR command on the console from which the program was started.

## 7.4. FCOPY - Copy File(s).

Copies all files (including nullfiles) matching a specified master file name and type from one disk to another.

Calling sequence (user input is shown underlined):

```
>FCOPY(,options)
Enter source disk identification: P2
Enter destination disk identification: P3
Enter master file type: R
Enter master file name: ********
```

where ´options´ is replaced by one or more of the characters C, R, and F. The options have the following meaning:

C - contiguous; the primary file size of the destination file is made so large that all of the file contents will fit within the primary file (no extents necessary).

R - rename; before each file is copied the user is asked to enter the name of the destination file on the console.

F - formatted; the files are considered sequential and copied record by record until an end-of-file mark or length indicator mismatch (MIKADOS error code 19) is encountered. This option is used primarily to regain space occupied by deleted records in type K and R files. This, however, gives no advantage if option C is specified.

The master file identification (name and type) is used to select the files that are copied. All file identifications on the source disk are compared to the master file identification. If a match is detected then the file is copied.

After a file has been copied its name is output on the console device.

If the destination file does not exist, it is created. If either the file type is 0 through 9 or the C option has been specified, FCOPY will ensure that the destination file is contiguous, that is, has no extents. To accomplish this, FCOPY may have to purge an existing file with the same name and type as the destination file and subsequently create the destination file with the required primary file size.

When FCOPY creates a destination file that is not required to be contiguous, the primary file size and default record length of the source and destination files will be the same.

All files are considered unformatted and copied completely without consideration of the file contents unless the F option is specified.

Destination files are extended by FCOPY as the need arises.

If the destination disk is filled during a copy the uncompletely copied file is deleted from the destination disk and the user is requested to insert a new destination disk.

After all files are copied, FCOPY asks for a new master file name. If no further copying is desired, FCOPY may be terminated by pressing ESC. If a new master file name is entered, the source and destination disks as well as the master file type are considered unchanged, and a new copy operation is initiated. If the request for a master file name is answered by just entering RETURN, FCOPY restarts and the user may enter new values of all FCOPY parameters.

FCOPY may be used to copy all information from one disk to another disk or other disks of a different type.

The user may stop an ongoing copy by entering a >.BR command on the console from which FCOPY was started. FCOPY will then be terminated after copying of the current file has been completed.

## 7.5. KOMPR - Pack a Disk.

Regains the space occupied by purged files on a disk.

Calling sequence (user input is shown underlined):

```
>KOMPR
Enter disk identification: P1
```

During the pack operation the program outputs information about the name, type, extent code, and address of the file extent currently being moved. The pack operation may take more than 20 minutes to complete on a full 5 Mbyte disk.

The pack operation has no influence on file contents. Unused space in sequential (type K and R) files is not recovered by KOMPR (use FCOPY with F option for this purpose).

If the computer or the disk drive fails during the pack operation, the only file that may be destroyed is the file that was being moved when the error occurred. However, experience has shown that in most cases even this file will be ok.

A disk should not be packed if there are any open files on the disk, nor should the user open files while the disk is being packed.

If KOMPR detects an inconsistency between the catalog and file pointers on the disk a message similar to the following will be output:

```
Catalog pointer 002F 08; actual file address 003F 08
Delete file (1), change catalog (2) or terminate KOMPR (3)? :
```

If this message is encountered, the user should not try to pack the disk further, but instead try to save as much of the disk as possible by backing it up onto another disk using FCOPY with master file name and type equal to *'s.

## 7.6. RENAME - Rename a File.

Renames a file of any type on any disk in the system.

Calling sequence (user input is shown underlined):

```
>RENAME
Enter old file name:disk identification (e.g. AB:P1) FILENAME:P1
Enter new file name NEWNAME
Enter file type 1
```

The RENAME program changes the name of the specified file. The file type cannot be altered.

## 7.7. FPURG - Purge File(s).

Purges (deletes) all files (including nullfiles) matching a specified master file name and type from a disk.

Calling sequence (user input is shown underlined):

```
>FPURG
Enter master file name: F****
Enter master file type: *
Enter disk identification: P1
```

The user may not enter a master file identification consisting of 9 *'s (8 for the file name and 1 for the file type); PLINI should be used to clear a disk completely.

Before a file is purged its name is output on the console device.

The space occupied by the purged file(s) is not available for new files before the disk has been packed (see section 7.5).

The FPURG program uses the MIKADOS PURGE call to purge a file.

## 7.8. OPROFIL - Create/Modify Nullfile.

Creates a nullfile, or modifies the value of a nullfile.

Calling sequence (user input is shown underlined):

```
>OPROFIL
Enter file name (may be followed by :disk designator) XXXXXX:P2
Enter address (4 digits required) YYYY
```

In the above example the user creates a nullfile named XXXXXX on disk

P2. The nullfile information value is YYYY (YYYY must be a 4-digit hexadecimal number).

If ´:disk designator´ is not specified, the nullfile is assumed to be on P1.

If the nullfile exists, the system prints the current nullfile value after the ´Enter address´ question and asks whether the current value should be retained or replaced with the new one.

Note that the fourth character of the program name is a zero.

# 8. Other utility programs.

## 8.1. MONITOR - Execute control file.

Executes one or more programs according to commands in a specified source file. This program will run only in computer banks of 32K or more (excluding the 16K used by MIKADOS).

Calling sequence (user input is shown underlined):

>MONITOR,filename

where ´filename´ is the name of a type K (source) file containing the commands to be executed. The file name may be followed by ´:disk identification´.

The monitor commands are:

1. Start program.

   The command line must start with a ´>´ immediately followed by a normal MIKADOS command used to start a program. This command should not include the ´*bank´ option of the MIKADOS start command.

2. Respond to a program console input request with an ASCII string.

   The command line must not start with ´>´ or ´<´. The command line contents are transferred to the program in response to the next console input or update string request. Note that in an update string (edit) request the original string value is ignored. Characters in the command line that do not fit within the program input buffer are ignored.

3. Respond to a program console input request with an ESCape. The command line must start with a ´<´. The remainder of the line is ignored.

4. Terminate MONITOR.

   The command line must start with a ´><´. The remainder of the line is ignored.

Upon encountering an end-of-file condition in the command file, the monitor responds with ESCape to all program input requests; the monitor terminates as soon as the current program has completed.

If the program issues an input request at a time where the next monitor command is a start program or terminate monitor command, the monitor responds with an ESCape.

If a program terminates at a time where the next monitor command is not a start program command, the monitor skips all commands until a start program command or a terminate monitor command is encountered.

Monitor commands are output on the console as they are executed. Start program commands are output with an extra '>' in front, that is, the command line output starts with '>>'. Escape commands are output as (ESCAPE). Otherwise, execution of a program through the monitor as seen from the console is identical to normal program execution.

The command file may be created and updated using the editor or any other program generating a text file.

Example: to copy certain files from one disc to another and assemble them the monitor file should contain

```
>FCOPY
P1
P2
K
FILE1
SOURCE*
<  terminate FCOPY with a simulated ESC
>ASM,FILE1
>ASM,SOURCE2
>ASM,SOURCE4
><
```

Note that the < in line 7 and the final >< are not mandatory.

The MONITOR program is resident in memory during execution of the control file. Thus, the monitor must be located in a memory area not used by any of the programs started by the monitor. Usually, the monitor resides in addresses A000 - C000, where it is capable of handling most MIKADOS utility programs, which use addresses 4000 - 8000. In certain cases, such as the execution of Pascal programs, it may be necessary to relocate the monitor, for example, move it to 4000 - 6000, with the Pascal program running from 6000 and up. Thus the stan-

dard MONITOR program delivered with a MIKADOS system should not be used to run Pascal programs and compilations.

## 8.2. PROM - Write into a PROM.

Writes into a PROM (Programmable Read Only Memory) of type 2708, 2716, 2516, or 2532. Performs a number of checks on the PROM contents.

Calling sequence (user input is shown underlined):

```
>PROM
Enter file name (enter only RETURN if copying from PROM) PROGFILE:P3
Enter file type 0
Program starting address 0000, ending address 4000, execution starts 0000.
Enter PROM type (2708, 2716, 2516, 2532) : 2708
Enter PROM starting address: 0C00
Enter P for PROM'ing, C for comparison: P
Checking the PROM contents.
* means five write cycles completed >**********<
```

The sequence used to copy one or more PROMs from a master PROM (empty response to first question) is similar to the above sequence.

A correctly erased PROM contains only 1-bits. The PROM program can only alter 1-bits to 0-bits.

If the program detects that the PROM to be programmed has not been correctly erased or that a PROM attempt has not been succesful, one or both of the following informative messages appear:

```
0001 PROM bytes have a 0 bit where a 1 bit is required (i.e.not PROM'able)
0001 PROM bytes have a 1 bit where a 0 bit is required (i.e.may be PROM'able)
```

If a byte is 'not PROM'able' it will not be included in the 'may be PROM'able' count even if it has one or more 1 bits where a 0 bit is required.

After an unsuccesful PROM attempt, the user may make another attempt, erase the PROM and try again, or consider the PROM defective.

To write a 2708 PROM an ID-7011 interface with i/o address 10 must be present in the computer.

## 8.4. SUBMIT - Start a Program on Another Terminal.

It is possible to start a program on another terminal using the SUBMIT program.

The file containing the program to be started must have the type 1.

The format of the command is:

>SUBMIT,n,ppppp:Pi,sssss

where

| | |
|---|---|
| n | is the number of the terminal where the program should run. |
| ppppp | is the name of the file containing the program. |
| Pi | is the disk where the program resides (:Pi may be omitted, in which case P1 is assumed). |
| sssss | is the parameters to the program. |

If the program is started sucessfully, the message

OK

appears on the originating terminal; otherwise an error message is given.

Example:

To get the KATLG program running on terminal 3 printing the contents of P2 the following command should be entered on any terminal:

>SUBMIT,3,KATLG,P2

Here KATLG is assumed to be found on P1.

Note:
The MIKADOS system in itself gives the user a possibility of running a program in another bank, but using his own terminal. This is described in section 4.1. of this manual.

system is started it will (if so specified during system generation) start the START program if this program resides on P1. The START program will in its turn start the specified programs on the specified terminals.

## 8.6. START.

Supplied with most MIKADOS systems is a START program. Running this program allows the user to enter the system date and time. The date will be stored in the label sector of P1.

## Appendix A. MIKADOS Error Codes.

The following error codes may be returned by the MIKADOS system. Some of these codes are relevant to both Pascal and assembly language programmers, others are only relevant to assembly language programmers and the description of these errors may refer to the MIKADOS Advanced Programming Guide.

The errors are grouped according to the part of MIKADOS that may detect a particular error.

### No Error.

| Error code | Explanation |
|---|---|
| 0 | No error |

### File System Errors.

| Error code | Explanation |
|---|---|
| 1 | A file with the specified name does not exist. |
| 2 | A file with the specified name already exists. |
| 3 | No more room on disk. |
| 4 | Illegal record length. |
| 5 | The file is already in use. |
| 6 | The file is not open. |
| 7 | The file is not closed. |
| 8 | Attempt to extend a file more than 60 times. |
| 9 | Illegal file name. |
| 10 | Illegal disk identification. |
| 11 | Attempt to position file to non-existent record. |
| 12 | Attempt to read or write a record that lies partly or completely outside the file boundaries. |
| 13 | The file has not been opened for writing. |
| 14 | The catalog on the disk is full. |
| 15 | Illegal DCB length. |
| 16 | Illegal number of sectors in file. |
| 17 | Illegal file type. |
| 18 | The file name has not been reserved (returned by CLOSE if the file is not open or DCB bombed). |
| 19 | Error in variable record length indicator (record length>136 or initial and final length indicator are not |

| | |
|---|---|
| | the same). |
| 20 | Internal file system error. |

Symbolic Resource Management Error.

| Error code | Explanation |
|---|---|
| 30 | No resource element available. |

Disk Driver Errors.

| Error code | Explanation |
|---|---|
| 40 | Disk not ready. |
| 42 | Hard error on disk. |
| 44 | Disk is write protected. |
| 48 | Illegal track/sector number or illegal buffer length. |
| 50 | Transfer extends past last sector of disk (perhaps because of a corrupted file structure on the disk). |
| 52 | Illegal disk identification. |

Process Management Errors.

| Error code | Explanation |
|---|---|
| 61 | No process control block available. (Each running program and each program that has been started but is not yet running occupies a process control block. Only a limited number of these blocks are available in the system.) |
| 62 | Illegal priority. |
| 63 | No memory available to execute program. |
| 64 | Illegal entry point for program. |
| 65 | Illegal load module file structure (number of extents not zero, number of sectors does not correspond to main storage limits specified). |
| 66 | Address does not specify a process control block. |
| 67 | Illegal operation code. |