

GenRef  
v1.00

MDOS Reference guide.

Utility Library

(C) Copyright 1989  
J. Paul Charlton  
**ALL RIGHTS RESERVED**

---

## UTILITY CONTENTS

---

	Page
Utility Overview.....	1
Calling Utility Functions.....	1
Validate Time.....	2
Read Time.....	3
Set Time.....	4
Validate Date.....	5
Read Date.....	6
Set Date.....	7
Julian Date.....	8
Day of Week.....	9
Parse Filename.....	10
Load Task.....	13
Fork Task.....	14

---

## UTILITY OVERVIEW

---

The memory management routines in MDOS are provided to aid a programmer in writing applications which are larger than the 64 Kbytes directly addressable by the CPU's 16 address lines. They also serve the purpose of providing each task with it's own private address space, separate from other the memory accessible to other tasks.

---

## CALLING UTILITY FUNCTIONS

---

The MDOS utility functions must be called from within a machine code program running as a task under MDOS. You pass arguments to the utility functions using only a few registers of your program's workspace.

The MDOS utility functions are invoked from a machine code program when software trap number zero (XOP 0) is called with a library number of 9. The calling program's R0 must contain the opcode of the routine within the utility library which is to be performed. The following code fragment will return the day of the week to the calling task.

```
                LI        R0,7
                XOP       @NINE,0
                MOV       R1,@WEEKDA
*   ...
WEEKDA         DATA     0           day of the week (1-7):(Sun-Sat)
*   ...
NINE           DATA     9
*   ...
```

---

**VALIDATE TIME**

---

**Function**                    This operation is used to check the time stored in the clock chip for validity. It insures that the minutes and seconds are in the range 0:59, and insures that the hours are in the range 0:23.

**Parameters**                R0                = 0 (opcode)

**Results**                    EQ status

**Parameter description**

EQ status                    The equal status bit will be set if the time is valid, allowing you to perform a "JEQ time\$ok" right after the software trap.

---

**READ TIME**

---

**Function** This operation reads the time of day from the clock chip, and places it into your string buffer as a formatted string, with colons between the hours, minutes, and seconds.

**Parameters** R0 = 1 (opcode)  
R1 = buffer

**Results** Buffer contains time string "HH:MM:SS".

**Parameter description**

Buffer The buffer address you pass for the string is a 16-bit address within your task's linear address space. The buffer must be ten characters long, and the address you pass is the address of the second character in the buffer.

On return, the first character of the buffer (offset 0) will contain a length byte. The next eight characters, starting at the address you specified, will contain the formatted time string. The last character in the buffer (offset 9) will contain a zero byte, for a null terminated string.

---

**SET TIME**

---

**Function** This operation will set the clock chip using the time in the formatting string which the calling task passes as an argument.

**Parameters** R0 = 2 (opcode)  
R1 = string

**Results** EQ status

**Parameter description**

String The address you pass for the string is a 16-bit address within your task's linear address space. The address you pass is the address of the second character in the buffer (the first text character in the string.)

The first character in the string buffer must be a length byte, giving the number of text characters in the string. Any leading spaces in the string will be ignored.

The text of the string must have the following format:

[h]h:[m]m[:[s][s]]

EQ status The equal status bit will be set if the time string is valid, allowing you to perform a "JEQ time\$ok" right after the software trap. The clock chip is not altered unless the EQ status has been returned.

---

**VALIDATE DATE**

---

**Function** This operation is used to check the date stored in the clock chip for validity. It insures that the month is in the range 1:12, the day of the month is the range 1:MAX\_DAYS[month], the year is in the range 0:99, and that the day of the week based on the month-day-year in the clock chip agrees with the day of the week stored in the clock chip itself.

**Parameters** R0 = 3 (opcode)

**Results** EQ status

**Parameter Description**

EQ status The equal status bit will be set if the date is valid, allowing you to perform a "JEQ date\$ok" right after the software trap.

---

**READ DATE**

---

**Function** This operation reads the date from the clock chip, and places it into your string buffer as a formatted string, with a dash between the month, day, and year.

**Parameters** R0 = 4 (opcode)  
R1 = buffer

**Results** Buffer contains date string "mm-dd-yy".

**Parameter Description**

Buffer The buffer address you pass for the string is a 16-bit address within your task's linear address space. The buffer must be ten characters long, and the address you pass is the address of the second character in the buffer.

On return, the first character of the buffer (offset 0) will contain a length byte. The next eight characters, starting at the address you specified, will contain the formatted date string "mm-dd-yy". The last character in the buffer (offset 9) will contain a zero byte, for a null terminated string.



---

**SET DATE**


---

**Function** This operation will set the clock chip using the date in the formatting string which the calling task passes as an argument.

**Parameters** R0 = 5 (opcode)  
R1 = string

**Results** EQ status

**Parameter description**

string The address you pass for the string is a 16-bit address within your task's linear address space. The address you pass is the address of the second character in the buffer (the first text character in the string.)

The first character in the string buffer must be a length byte, giving the number of text characters in the string. Any leading spaces in the string will be ignored.

The text of the string must have the following format:

[m]m/[d]d[/[y][y]] [m]m-[d]d[-[y][y]]

EQ status The equal status bit will be set if the date string is valid, allowing you to perform a "JEQ date\$ok" right after the software trap. The clock chip is not altered unless the EQ status has been returned.

---

**JULIAN DATE**

---

**Function**                This operation performs the function of a perpetual calendar, and will work on any date after January 1st, 1 AD.

**Parameters**            R0            = 6 (opcode)  
                         R1            = month  
                         R2            = day  
                         R3            = year

**Results**                R1,R2        = julian date

**Parameter description**

Year                    This must be the full year, like "1989", not "89", for the year in which this documentation was written.

Julian date             This is the number of days since January 1st, 4712 B.C.

---

**DAY OF WEEK**

---

**Function** Returns the day of the week, from one (Sunday) to seven (Saturday).

**Parameters** R0 = 7 (opcode)

**Results** R1 = weekday

**Parameter description**

Weekday This is a sixteen bit integer with a value from >0001 (Sunday) to >0007 (Saturday).

---

**PARSE FILENAME**

---

**Function** This operation will convert a logical filename descriptor to a physical filename descriptor recognized by the Device Service Routines. For disk devices, the conversion may depend on the drive currently set for the task and the current subdirectory on the drive (depending on the ambiguity left in the name by the calling program.)

It is useful when you wish to make your application program independent of which device it was loaded from or when your application must ask a user for a filename.

**Parameters**

R0	= 8 (opcode)
R1	= logical name
R2	= physical name
R3	= alias flag

**Results**

R0	= delimiter
R1	= error code
EQ	status

**Parameter description**

Logical name This is the address of the first character in the string to be converted to a physical device name.

At first, the name is compared to the names of all character devices recognized by MDOS. If the string matches the name of any of the character devices, the string will be copied without modification to the specified string output buffer.

There are three separators recognized by this routine as part of a disk path name: COLON ":", PERIOD ".", and BACKSLASH "\". If the first separator found before a terminating delimiter is a PERIOD, the entire string will be copied without modification to the specified string output buffer.

The following characters indicate the end of the input string if they are not contained inside of double-quote marks: SPACE, COMMA, SLASH, SEMICOLON. The NIL character (>00) always terminates the input string, even if there are unmatched double quotes in the string. These characters are referred to as "terminal characters".

The terminal characters along with the three separator characters are known as "delimiters".

Remaining filenames are parsed as follows:

Part A, drive alias. All characters parsed during this phase are ignored in subsequent phases of the parsing.

caller R3	alias flag <>0	-> null string
"volume:"	+ non-terminal	-> "WDS.volume."
"volume:"	+ terminal character	-> "volume"
"n:"	+ terminal character	-> "alias"
"n:"	+ non-terminal, "\", or "."	-> "alias."
all others		-> "alias."

Part B, current directory. Using the characters remaining in the input string after Part A.

"\"	is first character	-> null string
current dir	is NULL	-> null string
"."	+ terminator	-> "CURDIR"
".."	+ terminator	-> "PARENTDIR"
".\"	+ non-delimiters	-> "CURDIR."
"..\\"	+ non-delimiters	-> "PARENTDIR."
all others		-> "CURDIR."

Part C, file specifier. This is the characters remaining in the string after Part A and Part B have been done. These characters are copied into the output buffer until a BACKSLASH, QUOTE, or terminator is found. When a BACKSLASH is found, it is replaced by a PERIOD. When a terminator is found, parsing of the input string is stopped, and the address of the terminator is returned to the caller. When a QUOTE is found, all characters until the next QUOTE or NIL in the input string are copied to the output buffer (Unless two matching QUOTES were adjacent to each other, in which case a single QUOTE character will be placed into the output buffer.)

The resulting string in the output buffer is "PART A" + "PART B" + "PART C", and is returned to the caller.

Phys. name	<p>This is most useful when it specifies the name length byte in a PAB.</p> <p>Note that the physical name can use the same buffer as the logical name, and will simply overwrite the logical name after parsing is complete. As a caller, you must specify the address of the length byte in your string output buffer with this parameter. Before calling the parse routine, you must set the length byte to the maximum length allowed for the output string, which is returned in the form "&lt;len&gt;&lt;chars&gt;&lt;nil&gt;".</p>
Alias flag	<p>This flag must be set to zero for normal processing. If this flag is non-zero, no disk drive alias will be prepended to the output filename. (This feature is used only by the CHDIR command of MDOS at present.)</p>
Delimiter	<p>This is the address of the first character in your input string which wasn't processed during the generation of the filename. Note that this is designed in such a fashion that you generally don't need your own routine to parse filenames which are passed to your program as command line parameters; just call the parse routine, check the delimiter, and call the parse routine again for the next command line parameter.</p>
Error code	<p>This is set to zero if no errors were encountered while parsing the filename. This is non-zero under several conditions: the resulting output string was too long for your buffer, a COLON is the first character in the input string, a drive specified with "n:" does not have an assigned alias, or a directory specifier of the form "." or ".." was not followed by a BACKSLASH or a terminator.</p>

---

**LOAD TASK**


---

<b>Function</b>	This operation will load a chained program image file into memory, and cause it to execute as a task under MDOS. Invocation of the new task will start at address >0400 with a workspace of >F000, and memory windows 0..6 of the task will initially contain the data loaded from the program image.		
<b>Parameters</b>	R0	=	9 (opcode)
	R1	=	physical name
<b>Results</b>	R0	=	error code
	R1	=	child page zero (physical page number)
	>00E8		in child task contains the physical page number of the parent's page zero

**Parameter description**

Phys. name      This is the address of the length byte of a filename stored in the format "<len><chars>".

Error code

0	=	no error, task was loaded
1	=	insufficient memory
2	=	invalid filename
3	=	image file found, with invalid header

**Image file header**

An image file header has the following format, compatible with GenLINK.

byte 0	if >00, last image in chain, otherwise bump filename and load another image in the chain
byte 1	"G" (normal speed) or "F" (use fast memory)
byte 2,3	length of this image file
byte 4,5	load address of this image file
byte 6..len+6	image data bytes

---

**FORK TASK**

---

**Function**

This operation causes the creation of a new task under MDOS. The new task (child task) is an exact copy of the calling task (parent task).

The new task has a virtual memory address space which has one physical memory page for each physical page used by the parent task. If the parent task was using shared memory, the child task can also use the same shared memory and communicate with the parent task.

Further note: The terms "parent" and "child" are used as a convenience in differentiating between the calling task and the newly created task. In MDOS itself, there is no concept of "task tree" or parent-child relationship as there is in some other operating systems. In MDOS, all tasks are peers.

**Parameters**

R0 = 10 (opcode)

**Results**

parent task:

R0 = -1 (error)  
= otherwise, this is the physical page number of the child task's header page.

PC = program execution continues with the instruction after the XOP call. This instruction must be a single word instruction such as a Jump instruction.

child task:

R0 = -1

PC = program execution continues with the 2nd instruction after the XOP call. Note that the instruction after the XOP call must be a single word instruction.

>00E8 in child task contains the physical page number of the parent's page zero



**Parameter Description**

error                      An error code of -1 will be returned to the parent task if there is not enough memory available on your system to create a clone of your task.

**Example code**

```

                                LI      R0,10
                                XOP     @NINE,0
                                JMP     PARENT
CHILD   PRINT   "This is the child speaking..."
                                BLWP    @0      exit
PARENT  PRINT   "Child: be quiet!"
                                BLWP    @0
```