FUSION-C

M S X-DOS C L i b r a r y
by Eric Boez & Fernando Garcia

Quick Manual for FUSION-C v1.2

Revision – 2019 September, 6

The FUSION-C Library is free, it took hours and days to complete it, and bring it to the community. Thus, if you make some change, or add functions for your own use, think it can be useful to others MSX coders, please sharing your work.

You can, for example, send us your code, your new functions, your ideas. They will be included in future version of the library, for the benefit of every one.

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
You must provide : **the name of the creator and attribution parties, the title of the material**, a copyright notice, a license notice, a disclaimer notice, and a link to the material

ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

Notices:

You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.

No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

Fusion-C makes use or is distributed with external tools.

Sound FX Editor

AYFX Editor by Shiru.
https://shiru.untergrund.net/software.shtml

Source code and updates can be sound here :
https://github.com/Threetwosevensixseven/ayfxedit-improved

Binary to hexadecimal converter

Bin2froh By Ming Chen
Source code and updates can be found here :
https://sourceforge.net/projects/bin2froh

MSX Disk image manager

MSX DiskImage by ?

Disk-Manager by Lex Lechz
http://www.lexlechz.at

DSKTOOL v1.3 by Ricardo Bittencourt, Tony Cruise 2010, Natalia Pujol 2017
Updates and source code can be found here
https://github.com/nataliapc/MSX_devs

Disk to Rom converter

Disk2Rom 0.8 by Vincent van Dam

MSX-DOS

MSX-DOS 1.03 By Microsoft
MSX-DOS 2.30 By ASCII

Emulator

OpenMSX
Updates and information can be found here : https://openmsx.org

Sc2 GraphXConvertor 0.5
by Eric Boez & Leandro Xorreia

RLEWbCompressor 1.0b
BY ERIC BOEZ & AORANTE

For their help or contribution

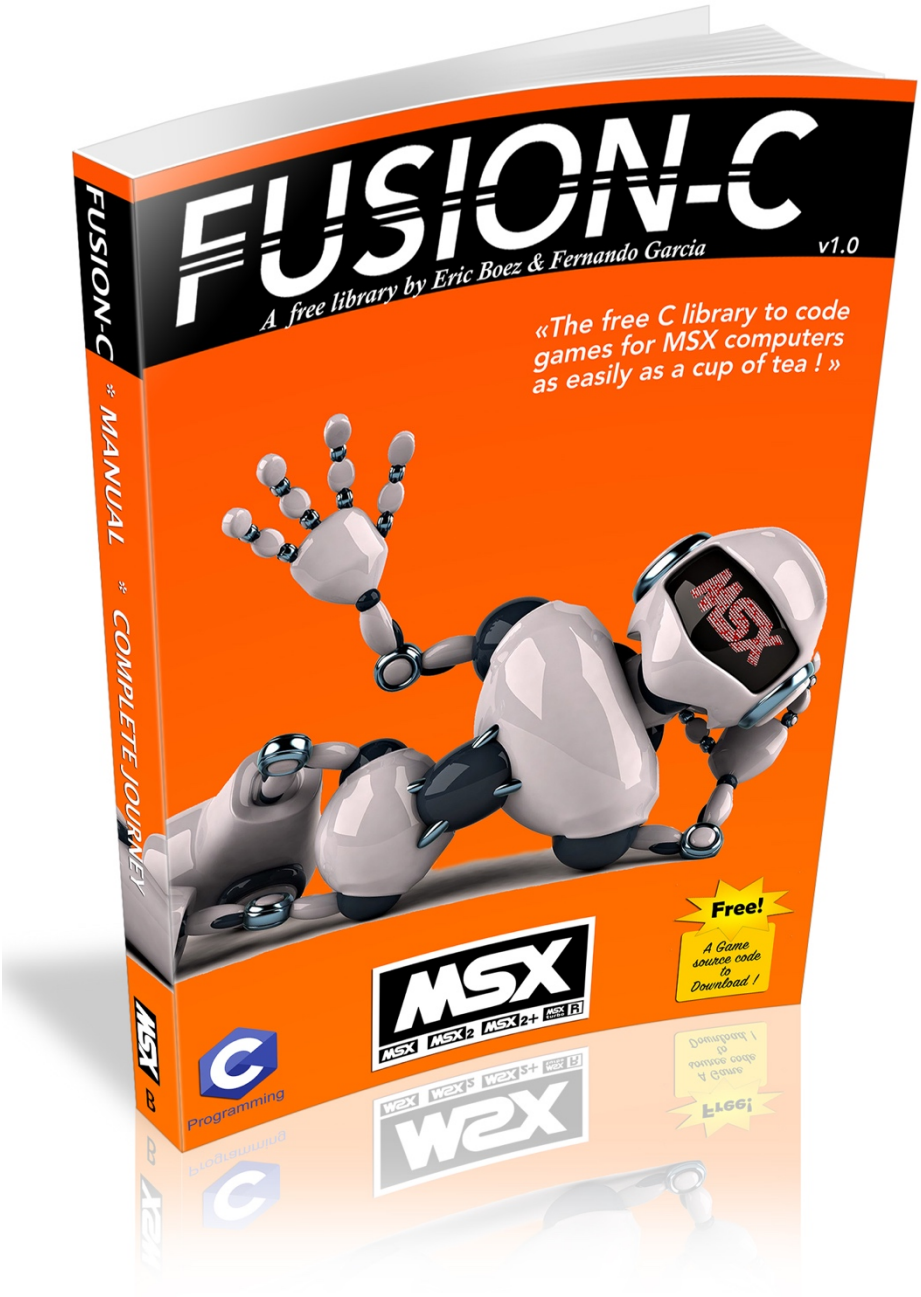
I want to thank:

Sylvain Cregut
Mvac7/303bcn
Nester Soriano
T.Hara
Jorge Torres Chacón
GDX
Javi Lanvandeira

This document is a quick manual, describing all functions of FUSION-C.

If you need more information about how to install the tools chains, and to automatize the compilation process, or if you need MSX hardware information to make your games, please consider to buy the "complete journey" manual on **Amazon.**
Go to Amazon and just do a search: "**MSX FUSION-C LIBRARY"** or search for **ERIC BOEZ**, you will find a big 290 pages book about **FUSION-C.**
Consider buying this book to encourage our work for the MSX.

Content of the FUSION-C , Complete journey:



FUSION-C Complete Journey. NEW VERSION 1.1 available
Available on Amazon for 12,99 €


Bz prod.
game studio'

What is « FUSION-C » ?

Fusion-C is a C library with which you can program software and games for MSX computers under MSX-DOS.

Face the fact there is no real complete and documented tools for the C development on MSX, I decided to reshape and to complete all SDCC code for MSX, available here and there to something coherent and easy to use.

FUSION-C is a C library dedicated to program in C for MSX Computers (MSX1, MSX2, MSX2+ and MSX Turbo-R), it uses the cross C compiler SDCC.

The name FUSION was chosen because this Library is based on some other Libraries, or part of libraries found separately here and there without any homogenization. FUSION-C is also composed with new source code, new commands and routines that will make your life easier in MSX programming.
One of the basis of FUSION-C comes from « Solid-C ». It was a MSX compiler and a C library for MSX created by Ego Voznessenski in 1995-1997.

In 2015 an unknown MSX user partially port the Solid-C library to the SDCC compiler, some part of this job is included in FUSION-C.

In 2006 T. HARA a Japanese developer who worked on the one chip MSX made some functions and routines for the SDCC compiler. Some of its routines are included in this Library.
Other routines comes from the work of Néstor Soriano, Jorge Torres Chacon, MVAC7, Eric Boez, and Fernando Garcia.

SDCC (for Small Device C Compiler) is an optimized Standard C compiler that can produce Zilog Z80 code for multiple Z80 based computers. It's a reliable tool, still in evolution and well documented. The compiler includes a standard C library partially compatible with the MSX. Actually we recommend using the SDCC version 3.6.0

The purpose is to bring to MSX users, tools and documentations to code programs and games in C.

With this aim in mind, with Fernando Garcia we completed the library with graphical functions, music and sound abilities, and the possibility to use the full content of the MSX Memory thru the MSX-DOS 2 memory mapper. We hope you will enjoy coding for MSX computers.

What's new in version 1.2

Again, errors have been corrected, bugs, syntax problems, faults, everything that has been reported has been corrected. Anyway there are still a lot of typos.

In this version MSX2 Graphics functions have been updated, and optimized, also new functions have been added.

Added functions :

HMMM (vdp_graph2.h)
Vram to Vram copy function for MSX2 graphic modes

LMMC (vdp_graph2.h)
Ram to Vram copy function with logical operation

YMMM (vdp_graph2.h)
Vram to Vram Copy function, only on Y axe

HMMV (vdp_graph2.h)
High speed VDP filled rectangle drawing

LMMV (vdp_graph2.h)
High speed VDP filled rectangle drawing with logical op

MouseReadTo (msx_fusion.h)
A simple way to read mouse direction and mouse buttons

Itoa (msx_fusion.h)
An integer to string conversion function

StrReverse (msx_fusion.h)
Reverse order of letters inside a string

BoxFill (msx_fusion.h)
Simple way to draw a filled rectangle

BoxLine (msx_fusion.h)
Simple way to draw rectangle lines

Some functions have been removed because they were in duplicate :

- **Rect** function was removed because it’s the same as **BoxLine**

Fixed or rewritten source code's functions :

RleWBToRam and **RleWBToVram**
MouseRead
LMMM (\!/ variables order changed)
Line (Now Works on Screen 7)
Point (Now Works on Screen 7)
Pset (Now Works on Screen 7)

New source code examples where added to the example directory.

What's new in version 1.1

Many errors have been corrected, bugs, syntax problems, faults, everything that has been reported has been corrected. But it is still possible that there are still a lot of typos.

More than that, many functions have been completely rewritten, for more speed and more compatibilities in your programs.
In addition, 22 new added functions complete FUSION-C with new possibilities like, mouse support, interrupt handler, a real timer, the ability to write and read sectors of disks and more ...

Added functions :

-MouseRead (msx_fusion.h)
Read MSX mouse coordinates attached to a joystick port

-SetRealTimer (msx_fusion.h)
-RealTimer (msx_fusion.h)
To use a real timer in your programs

-CovoxPlay (msx_fusion.h)
Play Sample sounds to a attached covox module

-SC2Circle (vdp_circle.h)
-SC2FilledCircle (vdp_circle.h)
Draw circles in Screen mode 2

-VDPLinesSwitch (msx_fusion.h)
Enable MSX2 screen modes to display 212 or 192 horizontals lines

-RleWBToVram (msx_fusion.h)
-RleWBToRam (msx_fusion.h)
Decompress RLEWB compressed data to RAM or VRAM

-GetDiskParam (io.h)
-GetDiskTrAddress (io.h)
-SetDiskTrAddress (io.h)
-SectorRead (io.h)
-SectorWrite (io.h)
Enable you to write, read directly sectors of disk devices

-CopyRamToVram (msx_fusion.h)
-CopyVramToRam (msx_fusion.h)
Functions to copy memory block to ram or to vram

-**PutText** (msx_fusion.h)
Enhanced version of previous PutText function, with logical operator

-**GetVramSize** (msx_fusion.h)
Function to return the VRAM size of a MSX Computer

-**SetBorderColor** (msx_fusion.h)
Set the screen border color

-**InitInterruptHandler** (msx_fusion.h)
-**EndInterruptHandler** (msx_fusion.h)
-**SetInterruptHandler** (msx_fusion.h)
Functions to activate and control an interrupt handler

Some functions have been removed because they were in duplicate :

-**WaitForKey** idem as WaitKey (msx_fusion.h)
-**KeyboardHit** idem as Inkey (msx_fusion.h)
-**Getcon** idem as Getche (msx_fusion.h)

Other updates :
Added long support to printf

Fixed or rewritten source code's functions :

-getche.s
-vdp_graph2.s
-setdate.s
-inkey.s
-waitkey.s
-getche.s
-joystickRead.c
-triggeread.c
-killekeybuffer.c
-fillvram.c
-crt0_msxdos.s

Many other bugs in source codes were removed to prevent overwriting of C variables

Errors in definitions fixed in msx_fusion.h

-StrToLower renamed into CharToLower
-StrToUpper renamed into CharToLower
-keyboardRead renamed into KeyboardRead
-EnableInterupt renamed into EnableInterrupt
-DisableInterupt renamed into DisableInterrupt

Installing the Tools Chain

What is fundamental nowadays, is the ability to code for MSX computers with modern tools. That's why with FUSION-C, you will not need an old computer or old Operating System to make your game for the MSX. You will just need to install some free tools on your Windows's PC, or Linux Machine, and even on your actual Mac OS X Apple 's computer.

I propose you a suite of tools I find pleasant and functional, but you are free to choose the tools you want.

Here the tools you will need :
- **Your actual computer** (Windows, Mac So or Linux) *
- **Open MSX Emulator**
- **SDCC** : Small Device C Compiler *
- **Hex To bin tool** *
- **Sublime Text** code editor.
- **The FUSION-C library** *
- **Compilation script**

** mandatory*

The code editor I'm using is Sublime Text. It is a very nice and professional code editor. It can be freely used. It is available on the three major operating systems.

I'm using the openMSX emulator because it's the more accurate MSX emulator, and can be remote by Shell/Dos commands.

The Hex2Bin tool is needed because SDCC do not generate a binary file you can execute on MSX. Hex2bin will do the job!

Download files

Download the « **SDCC Package** » for your operating system:
https://sourceforge.net/projects/sdcc/files/
(Choose Version 3.6.0)

Download « **Sublime Text 3** », for your operating system:
http://www.sublimetext.com/3

Download the latest version of « **Open MSX** » emulator:
https://openmsx.org/

Download the « **Hex2Bin** » tool:
https://sourceforge.net/projects/hex2bin/

Finally, download the « **FUSION-C** » Library from :
http://www.repro-factory.com
(it's totally free)

www.repro-factory.com/

On www.repro-factory.com, you can also download a game made with FUSION-C and the full source code, and also some others useful archives … ☺

Setting your working folder

Content of the Fusion-C Package Folder

Working Folder
- **dsk**
  - command.com
  - msxdos2.sys
- **fusion-c**
  - **examples**
  - **header**
  - **include**
  - **lib**
  - **source**
    - **lib**
- **openMSX**
- **Tools**
- Makefile
- test.c
- compil.bat

dsk : Destination folder of compiled programs, ready to be tested with OpenMSX
fusion-c : library main folder.
examples: Source codes examples
header: all header files needed to declare functions in your code
include: binaries folder used by compiler
lib: this folder contains fusion-c dynamic library
source: this contains sources codes
lib: contains all library source code and library compilation script
openMSX: MSX Emulator folder
tools: contains useful tools, and information

Make file: Compilation script for MacOS / Linux
Compil.bat : Compilation script for windows
test.c : test program

The Fusion-C Library comes with source codes, tools, examples, scripts, inside a ready to use «working folder».

Copy the « **working folder** » as is, where you want on your computer, and start to work from here.
All necessary tools and scripts will start from the use «working folder».

The example program « **test.c** » is an example ; once compiled, the MSX-DOS executable **« test.com** » file, will be copied in the « **dsk »** folder, and the OpenMSX emulator will be started with this folder as target.

Customize the SDCC Default Library

The SDCC Library comes with all standard libraries and functions of C.
Some of them, are not compatible with the MSX: GETCHAR, PUTCHAR and PRINTF functions are not working with MSX, that's why we need to replace these default functions by their equivalent for MSX-DOS.

The FUSION-C provide KONAMIMAN's GETCHAR, PUTCHAR, and PRINTF functions adapted to the MSX-DOS.

Once you have installed SDCC you must delete the original functions from the SDCC's Z80 Library.

On Mac OS :
Open your Shell/Terminal and go to the default Z80 library folder:
/usr/local/share/sdcc/lib/z80/

Type these commands:

```
> cp z80.lib z80.save
> sdar -d  z80.lib printf.rel
> sdar -d  z80.lib sprintf.rel
> sdar -d  z80.lib vprintf.rel
> sdar -d z80.lib putchar.rel
> sdar -d z80.lib getchar.rel
```

Now we have a saved copy of the original z80.lib file, and the original functions that don't work are removed from the library we will use.

On Windows :
Open your Dos window and browse to the default Z80 library folder :
C:\Program Files\SDCC\lib\z80\

Type these commands:

```
> copy z80.lib z80.save
> sdar -d  z80.lib printf.rel
> sdar -d  z80.lib sprintf.rel
> sdar -d  z80.lib vprintf.rel
> sdar -d z80.lib putchar.rel
> sdar -d z80.lib getchar.rel
```

The New and working MSX's functions, GETCHAR, PUTCHAR and PRINTF are included in the FUSION-C Library.

If one day you will need the **vprint or sprint** commands check the source file of these commands in the source folder of the SDCC library, and include them in your own source code.

Configuring the MSX you will use with OpenMSX

OpenMSX emulator is fully customizable, you can choose the MSX machine you will use with your programs, and also add expansions and peripherals to emulate.

At the end of the compilation process, the compilation Script will launch the emulator (if it doesn't already started). The MSX Machine it will start is describe in a config file you can find in **./Working Folder/openMSX/emul_start_config.txt**

The default content of this file is :

```
machine Philips_NMS_8255
ext msxdos2
ext gfx9000
bind F12 cycle videosource
plug joyporta mouse
plug printerport simpl
diska dsk/
```

Thus, this mean the emulator will launch an *MSX2 Model Philips NMS 8255*, with :
- *MSXDOS2 expansion*
- *GFX9000 graphic expansion*
- *Simpl/Covox module* attached to the printer port
- Mouse connected to joystick port A
- First disk drive is emulate in /*DSK* computer folder
- *the F12 Key* of your keyboard permut the screen between the *MSX Screen* and the *GFX 9000 Screen*.

By modifying this script you can add yourself other expansions like memory, music module etc …
Check the openMSX documentation : https://openmsx.org/manual/user.html

Content of the FUSION-C library

The Library is composed of multiple functions and routines we will describe them in the next chapters.

Functions are dispatched in some header files:

ayfx_player.h	:	AYFX, Sound FX player
io.h	:	File I/O functions
msx_fusion.h	:	Most of the needed functions are here
psg.h	:	PSG Sound functions
pt3replayer.h	:	PT3 Music replayer functions
rammapper.h	:	MSX-DOS 2 Memory Mapper
vdp_circle.h	:	Drawing circles on graphic screens
vdp_graph1.h	:	Graphic functions for MSX1
vdp_graph2.h	:	Graphic functions dor MSX2
vdp_sprites.h	:	Sprite related functions

Optional header files

newTypes.h	:	Definition of old school variables
vars_msxBios.h	:	Definition list of BIOS 's routines
vars_msxDos.h	:	Definition list of MSX-DOS 's routines
vars_msxSystem.h	:	Definition list of System variables

In version 1.2 we added support of the GFX 9000 graphic cartridge (Beta version)

g9klib.h	:	V9990 GFX9000 Support functions

and Gr8NET TCPIP Cartridge

gr8net-tcpip.h	:	TCPIP Funciton for the Gr8NET cartridge

MSX FUSION
[MSX_FUSION.H]

Console Functions

Print	**CONSOLE**
void Print(char *text)	
Prints ****text*** string on a text screen mode Supports escape sequences : **\a (0x07)** - Beep **\b (0x08)** - Backspace. Cursor left, wraps around to previous line, stop at top left of screen. **\t (0x09)** - Horizontal Tab. Tab, overwrites with spaces up to next 8th column, wraps around to start of next line, and scrolls at bottom right of screen. **\n (0x0A)** - Newline > Line Feed and Carriage Return (CRLF) Note: CR added in this Lib. **\v (0x0B)** - Cursor home. Place the cursor at the top screen. **\f (0x0C)** - Form feed. Clear screen and place the cursor at top. **\r (0x0D)** - CR (Carriage Return) **\" (0x22)** - Double quotation mark **\' (0x27)** - Single quotation mark **\? (0x3F)** - Question mark \\ **(0x5C)** - Backslash	

PrintNumber	**CONSOLE**
void PrintNumber (unsigned int num)	
Prints the ***num*** number supplied in parameter to a text screen (console).	

PrintFNumber	**CONSOLE**
void PrintFNumber(unsigned int num, char emptyChar, char length)	
Prints a number ***num*** to a text screen mode with formatting parameters ***emptyChar*** : (32=' ', 48='0', etc.) ***length*** : 1 to 5	

PrintChar	**CONSOLE**
void PrintChar(char c)	
Prints the character ***c*** to console (or to a text screen mode)	

PrintDec	**CONSOLE**
void PrintDec(int num)	
Prints signed integer ***num*** from -32768 to 32767 to a text screen mode	

InputChar	**CONSOLE**
char InputChar()	
Reads a character from console, and return it.	

InputString	**CONSOLE**
int InputString(char *dest, int len)	
Gets a string from console input and store it inside ****dest*** pointer variable. - ***dest*** : is a pointer where to store string - ***len*** : is the maximum length of the string. User can enter len-2 chars. Max length is 253 chars Returns the length of the string	

Getche	**CONSOLE**
char Getche()	
Reads and displays character from console. Returns the readed character.	

Locate	**CONSOLE**
void Locate(int x, int y)	
Sets console's cursor to ***X & Y*** coordinates	

printf	**CONSOLE**
int printf (const char *fmt, ...)	

Prints formatted text data to console (text Screen mode).

Example :
```
int nb=19;
char src[4]="bag";
printf("\n\r I have %d lollipops in my %s ",nb,src);
```

Supported format specifiers:

%d or %i: signed int%ud or %ui: unsigned int

%x: hexadecimal int

%c: character

%s: string

%%: a % character

%l: signed long

%ul: unsigned long

%lx: hexadecimal long

PrintHex	V1.2	**CONSOLE**
void PrintHex (unsigned int num)		
Prints the hexadecimal representation of the integer ***num***, on the text screen mode		

CheckBreak	V1.2	**CONSOLE**
int	**CheckBreak(void)**	
Checks the CTRL-BREAK in the MSX-DOS console. Return 0 if not pressed, or -1 if pressed		

PutCharHex	V1.2	**CONSOLE**
void	**PutCharHex(char c)**	
Prints the hexadecimal representation of the char ***num***, on the text screen mode		

Miscellaneous Functions

Cls	MISCELLANEOUS
void Cls(void)	
Clears console or any screen mode	

KeySound	MISCELLANEOUS
void KeySound(char n)	
Enables or disables Key Sound. ***n*** value must be : 0 : Disables Key Sound 1 : Enables Key Sound	

FunctionKeys	MISCELLANEOUS
void FunctionKeys (char n)	
Shows or hides Function Keys on a Basic text screen mode. ***n*** value must be : 0 : to disable 1 : to enable.	

ReadMSXtype	**MISCELLANEOUS**
char ReadMSXtype(void)	
Reads and returns the MSX type. Returned values : 0 : MSX 1 1 : MSX 2 2 : MSX2+ 3 : MSX Turbo-r	

ChangeCap	**MISCELLANEOUS**
void ChangeCap(char n)	
Changes the state of the Cap Led. ***n*** value must be : 0 : Disables Cap Led 1 : Enables Cap Led	

Screen	**MISCELLANEOUS**
void Screen(unsigned char mode)	
Sets display to specified screen mode. ***mode*** can be a valid screen mode number, from 0 to 8	

Beep	**MISCELLANEOUS**
void Beep(void)	
Plays a beep sound.	

RealTimer	*v1.1*	**MISCELLANEOUS**
unsigned int RealTimer(void);		
Reads and returns the real clock timer of the MSX computer. Timer is incremented by 1 on each VDP comlete screen draw. Depending on the computer's screen refresh rate. This can be 50 times per second for a PAL MSX Computer or 60 times per second on a Japanese MSX Computer. On MSX2, 2+ and MSX-Turbo-r the screen's refresh rate can be adjusted manually with VDP50Hz(void) and VDP60Hz(void) commands.		

SetRealTimer	*v1.1*	**MISCELLANEOUS**
void SetRealTimer (unsigned int value);		
Sets the Real Clock timer of the MSX computer to a specific ***value*** between 0 and 65535.		

CovoxPlay	*v1.1*	**MISCELLANEOUS**
void CovoxPlay(char page, unsigned int start_address, unsigned int length, char speed)		
Plays PCM audio sample (Wav type) stored in Vram thru Covox/simpl module. The PCM audio sample must be stored in the MSX Vram. With a MSX2 you can store, up to 128KB of Sample if you are using all the VRAM memory. Ideally you can store Sample in VRAM Page 2 of screen mode 8 Parameters : ***page*** is the VRAM page where the sample is stored ***start_address*** represents the first VRAM address of the Sample ***length*** is the length of byte you want to play ***speed***, represents the playing speed. More this number is high more the playing is slow.		

RleWBToRam *v1.1*	**MISCELLANEOUS**
void RleWBToRam (unsigned int *RamSource, unsigned int *RamDest)	
Decompress RLEWB data to Ram. ***RamSource*** is the pointer address where RLEWB data are stored in RAM ***RamDest*** is the pointer address you want to put uncompressed data n RAM Example : RleWBToRam(&RleData[0],&dest[0]); Note : The RLEWB Compressor command line tool is provided in the "Tools" folder	

RleWBToVram *v1.1*	**MISCELLANEOUS**
void RleWBToVram(unsigned int *RamAddress, unsigned int VramAddress);	
Decompress RLEWB data directly to Vram. ***RamAddress*** is the pointer address where RLEWB data are stored in RAM ***VramAddress*** is the address where you want to start to put uncompressed data, this address must be between 0 and 65535. If you want to uncompress to another VRAM Page, use the **SetActivePage** function to set the VRAM page. Example : RleWBToVram (&rledtata[0],0); Note : The RLEWB Compressor command line tool is provided in the "Tools" folder	

Joystick & mouse functions

JoystickRead	JOYSTICK
char JoystickRead(char joyNumber)	

Reads and returns state of a joystick.

JoyNumber must be :
0 : Keyboard's Arrow keys
1 : Joystick port 1
2 : Joystick port 2

Returned values:
0=inactive

1=up	2=up & right
3=right	4=down & right
5=down	6=down & left
7=left	8=up & left

TriggerRead	JOYSTICK
char TriggerRead(char TriggerNumber)	

Reads and return state of a joystick button.
TriggerNumber must be :
0 – space key
1 – button 1 joystick port A
2 – button 1 joystick port B
3 – button 2 joystick port A
4 – button 2 joystick port B
Note, there is no second button for the keyboard.

Returned values:
0 if button is inactive
255 if button is active

MouseRead	v1.1	**MOUSE**

unsigned int MouseRead(Int MousePort);

Reads and returns the mouse offsets of the mouse connected at MousePort.

The function returns the X offset and the Y offset into a 16bits value.
Decoding this value must be done like this:

```
Mouse_offset=MouseRead(MousePort2);
Xoffset=Mouse_offset >> 8;
Yoffset=Mouse_offset & 0xFF;
```

MousePort is refering to joystick port 1 or port 2, it is defined as a predefined macroYou lust use only :
- **MousePort1** or **MousePort2** as parameters.

Once you have decoded X offset and Y offset you can move a sprite object over the screen like this :

```
mx=mx-Xoffset;
my=my-Yoffset;
PuSprite(1,2,mx,my,15);
```

Note : If **MouseRead** returns **65535** as offset, this means no mouse is connected to the defined MousePort.

MouseReadTo V1.2 **MOUSE**
void MouseReadTo(unsigned char MousePort, MOUSE_DATA *md);
Reads and returns the mouse offsets, and the 2 buttons states of the mouse connected in MousePort. The function is using this pre-defined structure typedef struct { signed char dx; signed char dy; unsigned char lbutton; unsigned char rbutton; } MOUSE_DATA; You must declare this structure before using this function, like this : **static MOUSE_DATA mb;** ***MousePort*** must be 1 or 2 depending on the mouse port you want to read. returned values goes to the structure variables. According to the previous structure declaration, you will receive data inside mb.dx mb.dy mb.lbutton mb.rbutton **lbutton and lbutton are set to 0 when pressed** Code example : **MouseReadTo(1,&mb)**

Keyboard Functions

GetKeyMatrix	KEYBOARD
char GetKeyMatrix(char line)	

Returns the value of the specified ***line*** from the keyboard matrix. Each line provides the status of 8 keys. The state of the key is :
1 = not pressed
0 = pressed

bit	7	6	5	4	3	2	1	0
0	7 &	6 ^	5 %	4 $	3 #	2 @	1 !	0)
1	; :	] }	[{	\ |	= +	- _	9 (	8 *
2	B	A	acent	/ ?	. >	, <	`	' "
3	J	I	H	G	F	E	D	C
4	R	Q	P	O	N	M	L	K
5	Z	Y	X	W	V	U	T	S
6	F3	F2	F1	CODE	CAPS	GRAPH	CTRL	SHIFT
7	RET	SEL	BS	STOP	TAB	ESC	F5	F4
8	Right	Down	Up	Left	DEL	INS	HOME	SPACE
9	NUM4	NUM3	NUM2	NUM1	NUM0	NUM/	NUM+	NUM*
10	NUM.	NUM,	NUM-	NUM9	NUM8	NUM7	NUM6	NUM5

KillKeyBuffer	KEYBOARD
void KillKeyBuffer(void)	

Clears the key buffer

Inkey	KEYBOARD
unsigned char Inkey(void)	

Checks keyboard for a key pressed. Returns the ASCII code of the key, or 0 if no key pressed.

WaitKey	**KEYBOARD**
unsigned char WaitKey(void)	
Waits for a key pressed. Returns the ASCII code of the key.	

KeyboardRead	**KEYBOARD**
unsigned char KeyboardRead(void)	

Returns the state of some MSX Keyboard keys, into a 8bits value :
if a bit is active (1) the key is pressed. It can report 2 keys pressed at the same time

bit	7	6	5	4	3	2	1	0
	Right	Down	Up	Left	DEL	INS	HOME	SPACE

I/O Port Functions

OutPort	I/O PORT
void OutPort(unsigned char port, unsigned char data)	
Sends a 8 bits ***data*** value to a MSX ***port***	

InPort	I/O PORT
unsigned char InPort(unsigned char port)	
Reads from a MSX ***port***	

OutPorts	I/O PORT
void OutPorts(unsigned char port, unsigned char *p_data, unsigned char count)	
Sends ****p_data*** to a MSX ***port***.	

VDP Functions

VDPstatus	**VDP**
unsigned char VDPstatus(unsigned char vdpreg)	
Reads VDP Status register ***vdpreg*** and returns the value. The Interrupt is disabled before reading, and enabled after reading.	

VDPstatusNi	**VDP**
unsigned char VDPstatusNi(unsigned char vdpreg)	
Reads VDP Status register ***vdpreg*** and returns the value. The Interrupt is not modified by this function.	

VDPwriteNi	**VDP**
void VDPwriteNi(unsigned char vdpreg, unsigned char data)	
Writes ***data*** to the VDP register ***vdpreg***. The Interrupt is not modified by this function.	

VDPwrite	**VDP**
void VDPwrite(unsigned char vdpreg, unsigned char data)	
Writes **data** to the VDP register ***vdpreg***. The Interrupt is disabled before writing, and enabled after writing.	

IsVsync	**VDP**
unsigned Char IsVsync()	
Detects Vblank state. Returns 1, if true.	

IsHsync	VDP
unsigned Char IsHsync()	
Detects HSynch state. Returns 1 if true.	

Vpeek	VDP
unsigned char Vpeek(unsigned int address)	
Reads and returns a byte from Vram Memory ***address***	

Vpoke	VDP
void Vpoke(unsigned int address, unsigned char data)	
Writes the byte ***data*** in Vram at Memory ***address***	

VpokeFirst	VDP
void VpokeFirst(unsigned int address)	
Sets first Vram ***address*** for multiple Vpokes. Use VpokeNext after this instruction.	

VpokeNext	VDP
void VpokeNext(unsigned char data)	
Writes the byte ***data*** to the Vram Memory address sets by VpokeFirst and increments address for next use of this instruction.	

VpeekFirst	VDP
unsigned char VpeekFirst(unsigned int address)	
Sets first ***address*** for multiple Vpeek. Use VpeekNext after this instruction.	

VpeekNext	**VDP**
unsigned char VpeekNext()	
Reads and returns a byte from the Vram Memory address set by VpeekFirst, and increments address for next use of VpeekNext instruction	

Width	**VDP**
void Width(char n)	
Sets the width of a text screen mode, ***n*** must be between 1 and 80	

SetColors	**VDP**
void SetColors(int ForeCol, int BackgrCol, int BorderCol)	

Sets, foreground ***ForeCol*** color, background ***BackgrCol*** color and border color ***BorderCol***. Supports MSX2's screen mode like Screen8 ; colors can be a number between 0 and 255.

MSX1 colors palette :

- TRANSPARENT : 0x00
- BLACK : 0x01
- MEDIUM_GREEN : 0x02
- LIGHT_GREEN : 0x03
- DARK_BLUE : 0x04
- LIGHT_BLUE : 0x05
- DARK_RED : 0x06
- CYAN : 0x07
- MEDIUM_RED : 0x08
- LIGHT_RED : 0x09
- DARK_YELLOW : 0x0A
- LIGHT_YELLOW : 0x0B
- DARK_GREEN : 0x0C
- MAGENTA : 0x0D
- GRAY : 0x0E
- WHITE : 0x0F

SetBorderColor	**VDP**
void SetBorderColor(unsigned char BorderCol)	
Sets the screen border color ***BorderCol*** of the screen. It uses same colors as *SetColors* function.	

SetDisplayPage	**VDP**
void SetDisplayPage(unsigned char page)	
Sets the ***page*** of the screen mode which is displayed to the screen.	

SetActivePage	**VDP**
void SetActivePage(unsigned char page)	
Sets the ***page*** of the screen mode which receive modifications	

SetScrollH	**VDP**
void SetScrollH(int n)	
Uses Hardware horizontal scrolling of the MSX2+ and MSX Turbo-R. ***n*** *is the number of pixels to scroll, it can be positive or negative.*	

SetScrollV	**VDP**
void SetScrollV(int n)	
Uses Hardware vertical scrolling of the MSX2, MSX2+ and MSX Turbo-R. ***n*** is the number of pixels to scroll (positive or negative).	

HideDisplay	**VDP**
void HideDisplay(void)	
Hides the screen display (Black screen)	

ShowDisplay	**VDP**
void ShowDisplay(void)	
Shows the screen display if it was previously hidden.	

FillVram	**VDP**
void FillVram(int Startaddress, char value, int length)	
Fills the Vram from ***Startaddress*** with ***length*** bytes.	

PutText *v1.1*	**VDP**
void PutText(int X, int Y, char *str, char LogOp);	
Prints the text string ****str*** to graphic screen (mode 3 to 8) at position ***X, Y*** ***LogOp*** represents the Logical Operation used when printing text. It can be used to print text with a transparent background. (See possible values in MSX2 Graphics Chapter)	

VDP50Hz	**VDP**
void VDP50Hz(void)	
Switches the MSX2 VDP to 50 Hz Pal Mode	

VDP60Hz	**VDP**
void VDP60Hz(void)	
Switches the MSX2 VDP to 60 Hz NTSC Mode	

VDPLinesSwitch	*v1.1*	**VDP**
void VDPLinesSwitch(void)		
Switches the MSX2 VDP Chip to 192 or 212 vertical lines. By default MSX2 screen modes 5 to 8 are 256 x 212 pixels, but you can force the VDP to show 212 x 192 pixels.		

CopyRamToVram	*v1.1*	**VDP**
void CopyRamToVram(void *SrcRamAddress, unsigned int DestVramAddress, unsigned int Length);		
Copy a Ram Memory Block to a Vram Address. - ***SrcRamAddress***, must point to a Memory address or variable. - ***DestVramAddress***, must be a valid VRAM Address - ***Length***, is the length of the Memory Block to Copy		

CopyVramToRam	*v1.1*	**VDP**
void CopyVramToRam(unsigned int SrcVramAddress, void *DestRamAddress, unsigned int Length);		
Copy a Vram Memory Block to a Ram Address. - ***SrcVramAddress***, must be a valid VRAM Address **-** ****DestRamAddress***, must point to a memory address, or variable. - ***Length***, is the length of the Memory Block to Copy		

GetVramSize	*v1.1*	**VDP**
unsigned char GetVramSize(void);		
Returns the Vram size of the MSX Computer. Returned values can be, 16, 64, 128.		

Type Functions

IsAlphaNum	TYPE
int	**IsAlphaNum(char c)**
Checks if the supplied value ***c*** is alpha or numerical : A..Z or a..z or 0..9	

IsAlpha	TYPE
int	**IsAlpha(char c)**
Checks if the supplied value ***c*** is alpha : A..Z or a..z	

IsAscii	TYPE
int	**isAscii(char c)**
Check if the supplied value ***c*** is an ASCII character : - !..~	

IsCtrl	TYPE
int	**IsCtrl(char c)**
Checks if if the supplied value ***c*** is an unprintable control symbol	

IsDigit	TYPE
int	**IsDigit(char c)**
Checks if the supplied value ***c*** is a digit: 0..9	

IsGraph	TYPE
int	**IsGraph(char c)**
Checks if the supplied value ***c*** is a graph representation	

IsLower	**TYPE**
int IsLower(char c)	
Checks if the supplied value ***c*** is a lower char	

IsUpper	**TYPE**
int IsUpper(char c)	
Checks if the supplied value ***c*** is an upper char	

IsPrintable	**TYPE**
int IsPrintable(char c)	
Checks if the supplied value ***c*** is a printable char	

IsPunctuation	**TYPE**
int IsPunctuation(char c)	
Checks if the supplied value ***c*** is a punctuation sign	

IsSpace	**TYPE**
int IsSpace(char c)	
Checks if the supplied value ***c*** is a space	

IsHexDigit	TYPE
int IsHexDigit(char c)	
Checks if the supplied value ***c*** is a hexadecimal digit	

IsPositive	TYPE
Int IsPositive(int c)	
Checks if the supplied value ***c*** is positive. Return -1 if negative, 1 if positive, 0 if value is null.	

IntToFloat	TYPE
Float IntToFloat(int c)	
Returns a float value of the supplied value ***c***	

IntSwap	TYPE
void IntSwap(int *a, int *b)	
Swaps the content of two Integer variables.	

String Functions

CharToLower	**STRING**
char	**CharToLower(char c)**
Returns the supplied value ***c*** to the lowercase	

CharToUpper	**STRING**
char	**CharToUpper(char c)**
Returns the supplied value ***c*** to the uppercase	

StrCopy	**STRING**
void	**StrCopy(char *dst, char *src)**
Copy string from ****src*** to ****dst***	

NStrCopy	**STRING**
void	**NStrCopy(char *dst, char *src, int n)**
Copy string from ****src*** to ****dst*** with no more than *n* characters	

StrConcat	**STRING**
void	**StrConcat(char *dst, char *src)**
Concatenates string ****src*** at the end of ****dst***	

NStrConcat	**STRING**
void	**NStrConcat(char *dst, char *src, int n)**
Concatenates ***n*** characters from the string ****src*** at the end of ****dst***	

StrLen	**STRING**
int StrLen(char *string)	
Returns length of the ****string***	

StrCompare	**STRING**
int StrCompare(char *s1, char *s2)	
Compares two strings ****s1*** and ****s2***, Returns -1 if (s1<s2), 0 if (s1=s2), 1 if (s1>s2)	

NStrCompare	**STRING**
int NStrCompare(char *s1, char *s2, int n)	
Compares the ***n*** first characters two strings ****s1*** and ****s2***. Returns -1 if (s1<s2), 0 if(s1=s2), 1 if (s1>s2)	

StrChr	**STRING**
int StrChr(char *string, char c)	
It returns 1 if ***c*** is found in ****string***, otherwise returns -1	

StrPosStr	**STRING**
int StrPosStr(char *s1, char *s2)	
it looks for the string ****s2*** inside string ****s1*** and returns position of first character of ****s1***, or returns -1 if not found	

StrSearch	**STRING**
int StrSearch(char *s1, char *s2)	
Returns the first occurrence of any character from ****s2*** in the string ****s1*** Returns -1 if not found	

StrPosChr	**STRING**
int StrPosChr(char *string, char c)	
Returns the position of ***c*** inside ****string***, or -1 if not found	

StrLeftTrim	**STRING**
void StrLeftTrim(char *string)	
Removes left spaces inside ****string***	

StrRightTrim	**STRING**
void StrRightTrim(char *string)	
Removes right spaces inside ****string***	

StrReplaceChar	**STRING**
void StrReplaceChar(char *string, char c, char new_c)	
Replaces all chars ***c*** by ***new_c*** in string ****string***	

StrReverse V1.2	**STRING**
char* StrReverse(char *str)	
This function reverse the order of the chars inside the string ****str*** The new string is returned as a string chars	

Itoa	V1.2	**STRING**
char*	**Itoa(int num, char* str, int base)**	
This function convert an the integer ***num*** to a string of chars ****str.*** The new string is return as a string of chars, and must be declared before using this function. ***base*** indicates which base you want to convert to : 8, 10, 16		

Memory Functions

Poke	**MEM**
void Poke(unsigned int address, char data)	
Writes a byte ***data*** to Memory ***address***	

Pokew	**MEM**
void Pokew(unsigned int address, unsigned int data)	
Writes a 16 bits value ***data*** to Memory ***address*** (2 Bytes will be used in memory)	

Peek	**MEM**
Unsigned char Peek(unsigned int address)	
Reads and returns a byte value from Memory ***address***	

Peekw	**MEM**
Unsigned int Peekw(address)	
Reads a 16 bits value from Memory ***address***. (Will peek 2 bytes.). Returns the value to Int variable	

MemChr	**MEM**
char *MemChr(char *adr, char c, int n)	
Returns pointer to char in n bytes of adr, or NULL if not found	

MemFill	**MEM**
void MemFill(char *adr, char c, int n)	
Fills the Ram with a byte ***c***, from ***adr*** to ***adr***+n	

MemCopy	**MEM**
void MemCopy(char *dst, char *src, int n)	
Copy ***n*** bytes from ****src*** to ****dst***	

MemCopyReverse	**MEM**
void MemCopyReverse(char *dst, char *src, int n)	
Copy ***n*** bytes from ****src*** to ****dst***, but process will start from the end address (src + n) to the start address (src)	

MemCompare	**MEM**
int MemCompare(char *s1, char *s2, int n)	
Compares ***n*** bytes of ****s1*** and ****s2***, returns -1 if (s1<s2), 0 if (s1=s2), 1 if (s1>s2)	

MMalloc	**MEM**
void *MMalloc(unsigned int size)	
SDCC version of malloc, memory right below the code (heap_top=length of program+few bytes) should be free of data or code loaded after at runtime unsigned char *heap_top ;	

ReadTPA	**MEM**
Unsigned Int	**ReadTPA(void)**
Returns the TPA address of the current MSX-DOS running system. (Only available for MSX-DOS)	

ReadSP	**MEM**
Unsigned Int	**ReadSP(void)**
Reads the SP register, and returns the current lower address of the Stack. The Stack is inside the TPA zone. Stack is growing and decreasing while a program is running. In any case, the Stack address is the ultimate address you can use.	

Interrupt Functions

EnableInterrupt	**INTERRUPT**
void	**EnableInterrupt(void)**
Enables the Interrupt base system. May be useful in some circumstances	

DisableInterrupt	**INTERRUPT**
void	**DisableInterrupt(void)**
Disables the Interrupt base system. May be useful in some circumstances	

Halt	**INTERRUPT**
void	**Halt(void)**
Same as the Halt Asm Function. Wait for Interrupt.	

Suspend	**INTERRUPT**
void	**Suspend(void)**
Suspend Z80	

InitInterruptHandler *v1.1*	**INTERRUPT**
void	**InitInterruptHandler(void)**
Initialization of the Interrupt handler used by FUSION-C. Call this function at the beginning of your program.	

EndInterruptHandler *v1.1*	**INTERRUPT**
void	**EndInterruptHandler(void)**
If your program returns to MSX-DOS, it's important to end the Interrupt handler properly. Call this function just before the Exit to MSX-DOS.	

SetInterruptHandler *v1.1*	**INTERRUPT**
void	**SetInterruptHandler(char (*p_handler)(void));**

Sets the function of your program the interrupt handler process will call at each interruption.
The called function must be a function without any parameters, but it can make use of global variables.
Sets the function of your program the interrupt handler process will call at each interruption.The called function must be a function without any parameters, but it can make use of global variables.

Code example :

```
static char my_interrupt( void ) {
if( IsVsync() == 0 ) return 0;
  count++;
  return 1;
}

void main (void)

{
InitInterruptHandler();
SetInterruptHandler( my_interrupt );
...
}
```

PSG Functions

InitPSG	PSG
void InitPSG(void)	
Initialization of the PSG (Use this function before sending data to PSG). All registers will be set to 0, and stops all noises and sounds.	

PSGread	PSG
unsigned char PSGread(unsigned char psgreg)	
Reads data from <psgreg> PSG register	

PSGwrite	PSG
void PSGwrite(unsigned char psgreg, unsigned char data)	
Writes data to <psgreg> PSG register	

MSX-DOS File I/O Functions

Predefined macros & structures

```
typedef struct {
  unsigned char       drive_no ;
  unsigned char       name[8] ;
  unsigned char       ext[3] ;
  unsigned int        current_block ;
  unsigned int        record_size ;
  unsigned lon        file_size ;
  unsigned int        date ;
  unsigned int        time ;
  unsigned char       device_id ;
  unsigned char       directory_location ;
  unsigned int        start_cluster_no ;
  unsigned int        last_access_cluster_no ;
  unsigned int        cluster_offset ;
  unsigned char       current_record ;
  unsigned long       random_record ;
} FCB ;



typedef struct {
  unsigned char       name[8] ;
  unsigned char       ext[3] ;
  unsigned char       attribute ;
  unsigned char       undel_char ;
  unsigned char       reserve[9] ;
  unsigned int        time ;
  unsigned int        date ;
  unsigned int        start_cluster_no ;
  unsigned long       file_size ;
} FCB_DIR ;
```

```
typedef struct {
  unsigned char       drive_no ;
  FCB_DIR             dirinfo ;
} FCB_FIND ;
```

Return code :

FCB_SUCCESS	**0x00**

Dir attributes :

FCB_DIR ::attribute	
FCB_ATTR_READONLY	**0x01**
FCB_ATTR_HIDDEN	**0x02**
FCB_ATTR_SYSTEM	**0x04**
FCB_ATTR_VOLUME	**0x08**
FCB_ATTR_DIR	**0x10**
FCB_ATTR_ARCHIVE	**0x20**

fcb_open	**FILE I/O**
unsigned char fcb_open(FCB *p_fcb)	
Opens a file. Returns 0 if success	

fcb_create	**FILE I/O**
unsigned char fcb_create(FCB *p_fcb)	
Creates a file on media. Returns 0 on success	

fcb_close	**FILE I/O**
unsigned char fcb_close(FCB *p_fcb)	
Closes a file previously opened. Returns 0 on success	

Fcb_read	**FILE I/O**
unsigned int fcb_read(FCB *p_fcb, void *p_buffer, int nsize)	
Reads ***nsize*** bytes from opened file, and sends read data to ****p_buffer*** . ****p_fcb*** is the opened file handler.	

Fcb_write	**FILE I/O**
char fcb_write(FCB *p_fcb, const void *p_buffer, int nsize)	
Write *nsize* bytes to an opened file, from ****p_buffer*** ****p_fcb*** is the opened file handler.	

fcb_find_first	**FILE I/O**
char fcb_find_first(FCB *p_fcb, FCB_FIND *p_result)	
Finds first entry in directory	

Fcb_find_next	**FILE I/O**
char fcb_find_next(FCB_FIND *p_result)	
Finds next entry in directory	

MSX-DOS Functions

Predefined macros & Structures

```
typedef struct {
  unsigned int af, bc, de, hl, ix, iy, Cf, Zf ;
} REGS ;

typedef struct {
  int hour ;          /* Hours 0..23 */
  int min ;           /* Minutes 0..59 */
  int sec ;           /* Seconds 0..59 */
} TIME ;

typedef struct {
  int year ;          /* Year 1980…2079 */
  int month ;         /* Month 1=Jan..12=Dec */
  int day ;           /* Day of the month 1…31 */
  int dow ;           /* On getdate() gets Day of week 0=Sun…6=Sat */
} DATE ;
```

GetDate **MSX_DOS**

void GetDate (DATE *date)

Gets the current date, and sends it to ****date*** structure

Code example, get and show date :

```
DATE __at (0x21F0) dt ;
getdate(&dt) ;
putdec(dt.year) ; putch('.');
putdec(dt.month) ; putch('.');
putdec(dt.day) ; putch(' ');
```

GetTime	**MSX_DOS**
void GetTime (TIME *time)	
Gets the current time, and send it to ****time*** structure *Code example, get and show date :* TIME __at (0x2100) tm ; GetTime(&tm) ; PrintDec(tm.hour) ; putch(‘ :’); PrintDec (tm.min) ; putch(‘ :’); PrintDec (tm.sec) ;	

SetDate	**MSX_DOS**
int SetDate (DATE *date)	
Sets the system date thru the ****date*** structure. Returns 0 if valid.	

SetTime	**MSX_DOS**
int SetTime (TIME *time)	
Sets the system time thru the ****time*** structure. Returns 0 if valid.	

IntDos	**MSX_DOS**
void IntDos(void)	
Calls a MSXDOS System routine thru the REGS Structure : REGS *rr = _REGs() ; Must be Set before calling intdos(). Parameters must be given by dedicated structure.	

IntBios	**MSX_DOS**
void IntBios(void)	
Calls MSX BIOS System routine thru the REGS structure : REGS *rr = _REGs() ; Must be Set before calling intbios(). Parameters must be given by dedicated structure.	

Exit	**MSX_DOS**
void exit(char N)	
Exits from C program, and go back to MSX-DOS. *N* represent MSX DOS error number you want to show when exiting. 0 means no error.	

Turbo-r Functions

GetCPU	**TURBO-R**
char GetCPU (void)	
Gets the CPU Mode of the R800 processor. Returned values : 0 : Z80 mode 1 : R800 Rom Mode 2 : R800 Ram Mode	

ChangeCPU	**TURBO-R**
void ChangeCPU (char n)	
Changes the CPU Mode of the R800 processor. ***n*** must be : 0 : Z80 mode 1 : R800 Rom Mode 2 : R800 Ram Mode	

PCMPlay	**TURBO-R**
void PCMPlay(int start, int length)	
Plays a PCM sound stored in the Vram ***start*** : must be the Vram Address of the beginning of the PCM Sound ***length*** : is the length in bytes of the PCM sound to play.	

File I/O
[IO.H]

Functions to manipulate files with MSX-DOS 1 & MSX-DOS 2
Offer more possibilities than conventional functions.

Predefined macros & Structures

```
SEEK_SET      0
SEEK_CUR      1
SEEK_END      2

O_RDONLY      0
O_WRONLY      1
O_RDWR        1
O_CREAT       0x39
O_EXCL        0x04
O_TRUNC       0x31
O_APPEND      0x41
O_TEMP        0x80



typedef struct {
  unsigned char drive ;              // 0 : default drive
  unsigned char filename[11] ;    // 8+3 for extension, as « MYPROG  PRG »
  unsigned int block ;
  unsigned int record_size ;
  unsigned long file_size ;
  unsigned int date ;
  unsigned int time ;
  unsigned char device ;
  unsigned char dir_location ;
  unsigned int top_cluster ;
  unsigned int lastacsd_cluster ;
  unsigned int clust_from_top ;
  unsigned char record ;
  unsigned long rand_record ;
  unsigned char none ;             //+1 byte
} FCBstru ;                        // 38 bytes


typedef struct {
   FCBstru fcb[8] ;
} FCBlist ;
```

```
extern   FCBlist *FCBs( void ) ;

extern   int  _io_errno ;

// Structure that will receive Get Disk Parameters data (MSX-DOS2)
typedef struct {

      char DriveN;                  // Physical drive number (1=A: etc
       int SectorSize;              // Sector size (always 512 currently)
      char SectorPerCluster;        // Sectors per cluster (non-zero power of 2)
      int NumberReservedSector;     // Number of reserved sectors (usually 1)
      char NumberFatCopy;           // Number of copies of the FAT (usually 2)
      int NumberRootDirEntries;     // Number of root directory entries
      int TotalLogicalSectors;      // Total number of logical sectors
      char MediaDescriptionByte;    // Media descriptor byte
      char NumberSectorsPerFat;     // Number of sectors per FAT
      int FirstRootSectorNumber;    // First root directory sector number
      int FirstDataSectorNumber;    // First data sector number
      int MaximumCluster;           // Maximum cluster number
      char DirtyFlag;               // Dirty disk flag
      char VolumeId[4];             // Volume id. (-1 => no volume id.)
      char Reserved[8];             // Reserved (currently always zero)

} DSKPARAMS;
```

DiskLoad	**IO**

int DiskLoad
(char* filename, unsigned int address, unsigned int run_address)

Loads binary file from disk to RAM.
- ****filename*** : is 11 chars DOS1 for FCB : Example « MYFILE01BIN »
- ***address*** : where to load the first byte
- ***run_address*** : if not 0, then where to CALL after loaded

Returns 0 on success

GetOSVersion	**IO**
int GetOSVersion()	
Returns OS version : - 1 for MSX DOS 1 - 2 for MSX DOS 2 - 0 if not initiated Global Variables changed after calling : **_os_ver** : is set with MSX-DOS Kernel version **_mx_ver** : is set with MSXDOS2.SYS version number	

Open	**IO**
Char Open(char *file_name, int mode)	
Opens a file. Return an INT value as file Handler, or -1 if error **mode** can be - O_RDONLY : read only - O_WRONLY : write only - O_RDWR : read and write - O_CREAT : creation - O_EXCL : - O_TRUNC : - O_APPEND : append - O_TEMP : File operation errors are sent to the variable : **_io_errno**	

Create	**IO**
Int Create (char *file_name)	
Creates a file, named as ****file_name***	

FCBlist	IO
FCBlist *FCB = FCBs()	
Mandatory when reading or writing files with MSX DOS 1. (Or MSX DOS 2 In compatibility mode) FCBLIST will give a file handler. Example of use : char sbuf[10] ; // Set a 10 bytes buffer FCBlist *FCB = FCBs() ; // FCB initialization int fH ; // Set a file handler variable fH = open(« TEST0001.SC8 », O_RDWR) ; // open file for read read(fH, sbuf, 10) ; // Read 10 bytes to 74 close(fH) ; // Close file	

Close	IO
Int Close (int fH)	
Closes a file, previously opened. ***fH*** is the file handler that must be provided as argument.	

Read	IO
Int Read (int fH, void *buffer, unsigned int nbytes)	
Reads **nbytes** bytes from a file definied by ***fH*** handler to ****buffer***. The file must be previously opened.	

Write	**IO**
Int Write(int fH, void *buffer, unsigned int nbytes)	
Writes ***nbytes*** bytes from ****buffer*** to a file defined by ***fH*** handler. The file must be previously opened.	

OpenAttrib	**IO**
Int OpenAttrib(char *file_name, int mode, int attr)	
Only with MSX DOS 2. Opens a file (is same as the open function) (Function not tested!)	

CreateAttrib	**IO**
Int CreateAttrib(char *name, int attr)	
Only with MSX DOS 2. Creates a file. - attr is same as the open function. (Function not tested!)	

GetCWD	**IO**
Int GetCWD(char *buf, int bufsize)	
Gets directory of A : to buffer - ****buf*** is buffer pointer - ***bufsize*** is the buffer size	

GetDisk	**IO**
Int GetDisk(void)	
Gets and returns current drive number.	

SetDisk	**IO**
void SetDisk(int diskno)	
Sets ***diskno*** as current drive number.	

Ltell	**IO**
Int Ltell (int fH, long address)	
Reads file position, returns 0, or error in**: _io_error**. ***Address*** is like 0xABCD, will operate as 4-bytes long value	

Lseek	**IO**
Int Lseek(int fH, long address_value, int ot)	
Sets file position, returns 0, or error in **_io_error**. **ot** must be : - SEEK_SET : 0 - SEEK_CUR : 1 - SEEK_END : 2	

Remove	**IO**
Int Remove(char *filename)	
Removes file ****filename*** from directory. Returns 0 on success, or error in **_io_error**	

ChangeDir	**IO**
Int ChangeDir(char *path)	
sets current path. Only available with MSX DOS 2 Returns 0, or error **in _io_error**	

FindFirst	**IO**
Int FindFirst(char *wildcard, char *result, int attr)	
Finds files or folders by wildcard as « *.COM », « ???? », etc. Returns 0 on success, or error in _io_error. Provide 0 or attributes for MSXDOS2. n=findfirst(« *.* »,sbuf,0) ; for(;!n ;){ cputs(sbuf) ; cputs(sn) ; n=findnext(sbuf) ; }	

FindNext	**IO**
Int FindNext(char *result)	
See findfirst function. Find next occurrence.	

MakeDir	**IO**
int MakeDir(char *folderName)	
Creates folder. Only available with MSX DOS 2 Returns 0, or error in : _io_error	

RemoveDir	**IO**
int RemoveDir(char *folderName)	
Removes a folder. (Only with MSX DOS 2). Returns 0, or error in : **_io_error**	

_tell	**IO**
unsigned long _tell(int fH)	
Reads and return file position from FCB Only for MSX DOS 1	

_seek	**IO**
void _seek(int fH, long pos, int ot)	
Sets file position from FCB Only for MSX DOS 1 ot must be - SEEK_SET - SEEK_CUR - SEEK_END	

_size	**IO**
unsigned long _size(int fH)	
Returns the file size from FCB Only for MSX DOS 1	

GetDiskParam *v1.1*	**IO**
unsigned char GetDiskParam(DSKPARAMS *info, unsigned char Drive)	
After Initialization of the DSKPARAMS Structure the call to this instruction will get all parameters of the disk drive. It may be a floppy Drive or any other MSX-DOS 2 's storage drive. This instruction only works with MSX-DOS2. See the DSKPARAMS Structure at the beginning of this chapter for details.	

SetDiskTrAddress *v1.1* **IO**
void SetDiskTrAddress(unsigned int *address);
Sets the Memory ****address*** where the data will be transferred, when a Reading Sectors instruction is called or when a Writing function is called. Most of the time a sector size is 512 bytes long (See result of a GetDiskParam), thus if you want to store at least one sector, the ***address** pointer must point to a 512 bytes variable area (minimum).

GetDiskTrAddress *v1.1* **IO**
unsigned int GetDiskTrAddress(void);
Returns the Memory address where the data are transferred, when a Reading Sectors instruction is called, or when a writing sector instruction is called.

SectorRead *v1.1* **IO**
unsigned char SectorRead(unsigned int SectorStart, unsigned char drive, unsigned char NbSectors);
Reads ***NbSectors*** Sectors from ***SectorStart*****,** on the specified ***drive*****.** Data are sent to the memory address set by SetTrAddress instruction. Returns 0 if no error.

SectorWrite *v1.1* **IO**
unsigned char SectorWrite(unsigned int SectorStart, unsigned char drive, unsigned char NbSectors);
Write ***NbSectors*** Sectors from ***SectorStart*****,** on the specified ***drive*****.** Sectors are written with the data stored at the memory address set by the instruction SetTrAddress instruction. Returns 0 if no error.

MSX1 GRAPHICS
[VDP_GRAPH1.H]

MSX1 Graphic and draw functions.

Predefined macros & Structures
// filling mode **NO_FILL 0x00** **FILL_ALL 0xFF** **/* structure to set/get color of 8 pixels */** **typedef struct {** **int col ; // color number 0..15 for pixels of pattern** **int bg ; // background color number 0..15** **} pxColor ;**

SC2WriteScr	**VDP_GRAPH1**
void SC2WriteScr(unsigned int addr_fromPalettes, unsigned int addr_fromColours)	
Writes RAM to VRAM (2x6144 bytes) Use with Screen mode 2 or 3	

SC2ReadScr	**VDP_GRAPH1**
void SC2Read_Scr(unsigned int addr_toPalettes, unsigned int addr_toColours)	
Reads VRAM to RAM(2x6144 bytes) Use with Screen mode 2 or 3	

ReadBlock	**VDP_GRAPH1**
void ReadBlock (int X, int Y, int dx, int dy, unsigned int addr_toPalettes, unsigned int addr_toColours)	
VRAM => RAM (copy block to memory) (X,Y) – left upper corner of screen position to copy dx,dy – count of columns and rows of pixels So, the block (X,Y)-(X+dx-1,Y+dy-1) will be copied. X,dx should be 0,8,16,24,32,… 8*n because complete 8-pixel patterns will be copied Requires 2 memory blocks size of (dx/8)*dy	

WriteBlock	**VDP_GRAPH1**
void WriteBlock(int X, int Y, int dx, int dy, unsigned int addr_fromPalettes, unsigned int addr_fromColours)	
RAM => VRAM (puts from memory to screen) (X,Y) – where to put on screen	

Get8px	**VDP_GRAPH1**
int Get8px(int X, int Y)	
Gets byte of 8-pixels at (X,Y)	

Get1px	**VDP_GRAPH1**
int Get1px(int X, int Y)	
Gets pixel of 8-pixels at (X,Y). Returns 0 if not set.	

Set8px	VDP_GRAPH1
void Set8px(int X, int Y)	
Sets whole byte of 8-pixels at (X,Y)	

Set1px	VDP_GRAPH1
void Set1px(int X, int Y)	
Sets pixel of 8-pixels at (X,Y)	

Clear8px	VDP_GRAPH1
void Clear8px(int X, int Y)	
Clears byte (sets=0) of 8-pixels at (X,Y)	

Clear1px	VDP_GRAPH1
void Clear1px(int X, int Y)	
Clears pixel (sets=0) of 8-pixels at (X,Y)	

GetCol8px	VDP_GRAPH1
void GetCol8px(int X, int Y, pxColor *C)	
Get color of 8-pixel pattern at (X,Y) See the predefined structure	

SetCol8px	VDP_GRAPH1
void SetCol8px(int X, int Y, pxColor *C)	
Sets new color in (X,Y) for 8-pixel pattern typedef struct { int col ; // color number 0..15 for pixels of pattern int bg ; // background color number 0..15 } pxColor ;	

SC2Point	VDP_GRAPH1
int SC2Point(int X, int Y)	
Gets color of pixel at (X,Y), (same for 8-pixel pattern) Use with Screen mode 2 or 3	

SC2Pset	VDP_GRAPH1
void SC2Pset(int X, int Y, int color)	
Puts pixel in (X,Y), sets color of whole 8-pixel pattern Use with Screen mode 2 or 3	

SC2Line	VDP_GRAPH1
void SC2Line(int X, int Y, int X2, int Y2, int color)	
Draws a line from (X,Y) to (X2,Y2), with color, Does not change background color Use with Screen mode 2 or 3	

SC2Rect	VDP_GRAPH1
void SC2Rect(int X1, int Y1, int X2, int Y2, int color, int OP)	
Draws a rectangle from top left corner (X,Y) to bottom right corner (X2,Y2). Use color, with filling operation OP OP can be - NO_FILL - FILL_ALL Use with Screen mode 2 or 3	

SC2Paint	VDP_GRAPH1
void SC2Paint(int X, int Y, int color)	
Paints for small screen regions. Slow Function! Use with Screen mode 2 or 3	

SC2Draw	VDP_GRAPH1
void SC2Draw(char *drawcommands)	
Remake of BASICs « draw » with original commands (except A,X) syntax	

MSX2 GRAPHICS

[vdp_graph2.h]

MSX2 Graphic and draw functions.

Predefined Structures and macros

Logical fill for rectangle and circle

```
FILL_ALL        0xFF
NO_FILL         0x00
```

Palette

```
typedef struct {
    unsigned char color ;   // color number 0..15
    unsigned char R ;       // 0..7   red brightness
    unsigned char G ;       // 0..7   green brightness
    unsigned char B ;       // 0..7   blue brightness
} ColRGB ;

typedef struct {
  ColRGB rgb[16] ;
} Palette ;
```

fLMM variable structure :

```
typedef struct {
  unsigned int X ;      // source X (0 to 511)
  unsigned int Y ;      // source Y (0 to 1023)
  unsigned int X2 ;     // destination X (0 to 511)
  unsigned int Y2 ;     // destination Y (0 to 1023)
  unsigned int DX ;     // width (0 to 511)
  unsigned int DY ;     // height (0 to 511)
  unsigned char s0 ;    // set to 0, dummy 1st empty byte sent to chip
  unsigned char DI ;    // set to 0 (b), works well from left to right
  unsigned char LOP ; // 0 to copy (a), Logical+Operation
} MMtask ;
```

Logical operations

When graphic commands are called, various logical operations can be done between the source data and the destination data. Here how logical operation are working.

SC : is for source color code **DC :** is for destination color code

Param.	**Name**	**action**
0	LOGICAL_IMP	DC=SC
1	LOGICAL_AND	DC=SC*DC
2	LOGICAL_OR	DC=SC+DC
3	LOGICAL_XOR	DC=$\overline{SC}$*DC+SC*$\overline{DC}$
4	LOGICAL_NOT	DC=$\overline{SC}$
8	LOGICAL_TIMP	if SC=0 then DC=DC else DC=SC
9	LOGICAL_TAND	if SC=0 then DC=DC else DC=SC*DC
10	LOGICAL_TOR	if SC=0 then DC=DC else DC=SC+DC
11	LOGICAL_TXOR	if SC=0 then DC=DC else $\overline{DC}$=SC*DC+SC*$\overline{DC}$
12	LOGICAL_TNOT	if SC=0 then DC=DC else DC=$\overline{SC}$

vMSX **VDP_GRAPH2**

Int vMSX(void)

Check MSX VDP version.
Returns 1 for MSX1, or 2 for MSX2

WriteScr **VDP_GRAPH2**

void WriteScr(unsigned int addr_from)

Writes data from RAM ***addr_from*** to VRAM screen
Only work with screen 5 (page 0) mode. This function copies 0x6A00 bytes to the VRAM

ReadScr	VDP_GRAPH2
void ReadScr(unsigned int addr_to)	
Reads VRAM screen (page 0) to RAM memory address ***addr_to***. Only works with screen 5 mode. This function copies 0x6A00 bytes to RAM Use it with Screen5 mode	

SetSC5Palette	VDP_GRAPH2
void SetSC5Palette ((Palette *) mypalette)	

Sets the screen 5 color palette with « ***mypalette*** » structure data.
The predefined « Palette Structure » is composed of 16 lines formatted like this : N, R, G, B
N is the number of the color (0 … 15). R, G, and B are the level of Red, Green or Blue in the final color. R, G and B must be between 0 and 7.
Example :

```
char mypalette[] =
  {
            0, 0,0,0,
            1, 2,1,1,
            2, 6,5,4,
            3, 5,4,3,
            4, 5,5,3,
            5, 6,5,3,
            6, 7,6,4,
            7, 3,2,1,
            8, 7,5,2,
            9, 6,4,2,
            10, 4,3,2,
            11, 6,0,1,
            12, 5,3,2,
            13, 3,3,2,
            14, 3,1,0,
            15, 6,6,6
};
SetSC5Palette((Palette *) mypalette);
```

RestoreSC5Palette	VDP_GRAPH2
void RestoreSC5Palette (void)	
Sets screen 5 color palette to the default palette	

Pset	VDP_GRAPH2
void Pset(int X, int Y, int color, int OP)	
Draws a pixel at ***X,Y*** with the defined ***color*** and logical operation ***OP***	

Point	VDP_GRAPH2
char Point(int X, int Y)	
Reads and return color of the pixel at ***X,Y***	

Line	VDP_GRAPH2
void Line (int X1, int Y1, int X2, int Y2, int color, int OP)	
Draws a line from ***X1,Y1 to X2,Y2*** with the defined ***color*** and logical operation ***OP***	

BoxLine	VDP_GRAPH2
void Rect (int X1, int Y1, int X2, int Y2, int color, int OP)	
Draws a rectangle ***from X1,Y1*** (left upper corner) to ***X2,Y2*** (right bottom corner) with the defined ***color*** and logical operation **OP**. Use **FILL_ALL** as operator to fill the rectangle. Any Other operator will draw an empty rectangle. If you want to draw filled rectangle with operator, use BoxFill function.	

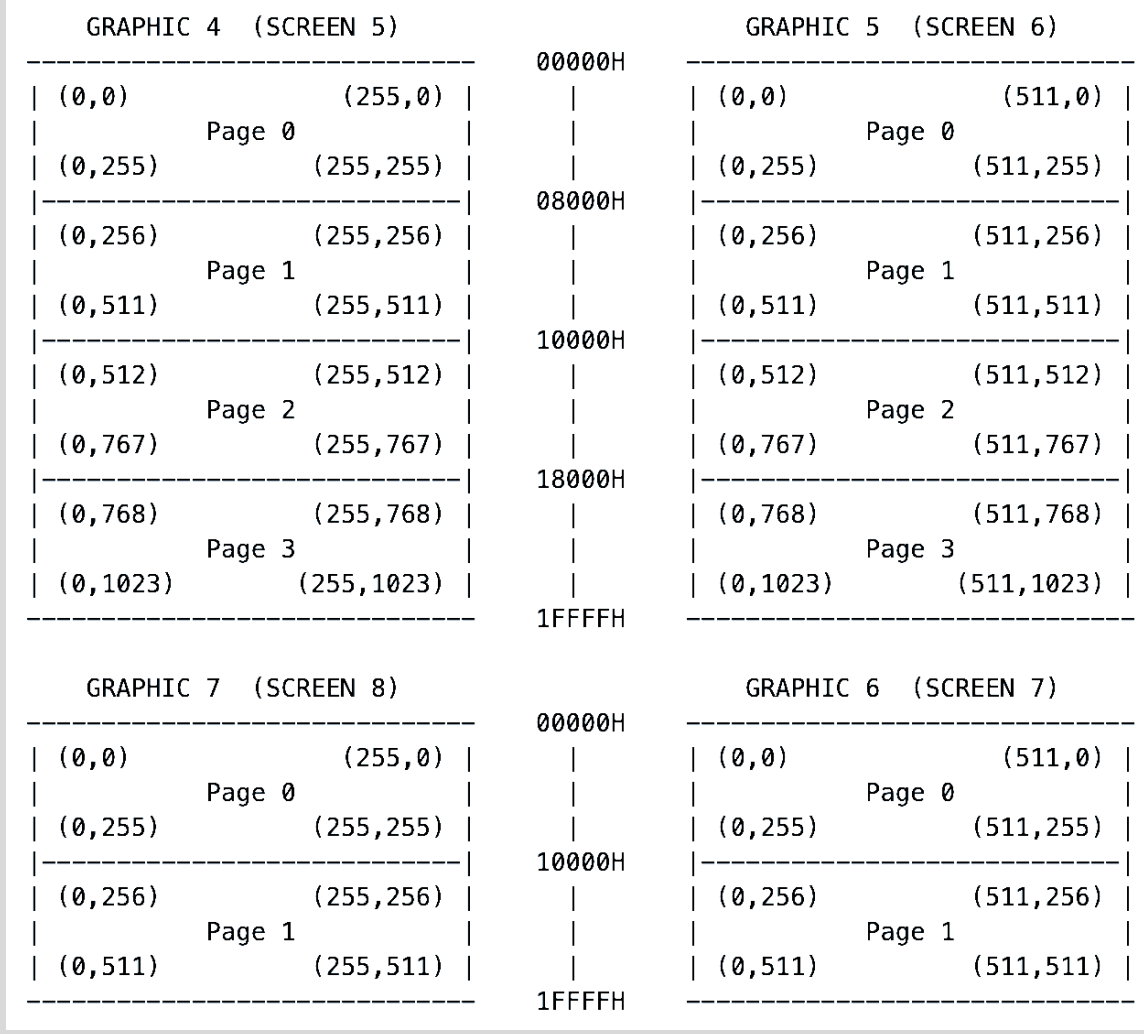
BoxFill V1.2 **VDP_GRAPH2**
void BoxFill (int X1, int Y1, int X2, int yY22, char color, char OP)
Draws a filled rectangle ***from X1,Y1*** (left upper corner) to ***x2,y2*** (right bottom corner) with ***color*** and logical operation **OP**.

Paint **VDP_GRAPH2**
void Paint(int X, int Y, int color, int border_color)
Slow and dummy paint. Paint the area at ***X,Y*** with ***color***, stopping at ***border_color***

Draw **VDP_GRAPH2**
void Draw(char *drawcommands)
Remake of BASICs « draw » with original commands (except A,X) syntax

High speed VDP Commands

VDP High speed commands are using the global system coordinates to address the full 128KB of VRAM



HMMM	V1.2	**VDP_GRAPH2**

void HMMM(int XS, int YS, int XT, int YT, int DX, int DY);

High speed from VRAM to VRAM
Copy the rectangle image starting at ***XS,YS*** *(top left corner of the rectangle)* to the target ***XT,YT*** coordonate. Length and high of the rectangle image are defined by ***DX*** and ***DY.*** No logical operation allowed.

LMMM V1.2 **VDP_GRAPH2**
void LMMM (int XS, int YS, int XT, int YT, int DX, int DY, unsigned char OP)
High speed copy with logical Operation from VRAM to VRAM Copy the rectangle image starting at ***XS,YS*** *(top left corner of the rectangle)* to the target coordinates ***XT,YT***. Length and high of the rectangle image are defined by ***DX*** and ***DY*** ***OP*** must be a standard logical operator If you want to copy a rectangle image from one vram page to another, use the ***YT*** coordonate. For example, if YT>255 you are working on the 2nd VRAM page

fLMMM **VDP_GRAPH2**
void fLMMM(MMMtask *VDPtask)
High speed copy by structure ****VDPtask*** . Same effect as LMMM command but here it is using a structure where you set parameters. First declare the structure: ***MMMtask t ;*** Predefined structure : typedef struct { unsigned int X ; // source X (0 to 511) unsigned int Y ; // source Y (0 to 1023) unsigned int X2 ; // destination X (0 to 511) unsigned int Y2 ; // destination Y (0 to 1023) unsigned int DX ; // width (0 to 511) unsigned int DY ; // height (0 to 511) unsigned char s0 ; // set to 0. 1st empty byte sent to chip unsigned char DI ; // set to 0 (b), works well from left to right unsigned char LOP ; // 0 to copy (a), Logical Operation } MMMtask ;

HMCM	VDP_GRAPH2
void HMCM_SC8 (int X1, int Y1, int X2, int Y2, void *tobuffer, unsigned char OP)	
High speed Copy the rectangle (***X1,Y1***)-(***X2,Y2***) from VRAM to the RAM buffer ****tobuffer***. Only for Screen 5 mode. ***OP*** is a standard logical operator. In screen mode 5 or 7, ***X2-X1*** must be even. Use this function only in screen mode 5 or 7.	

HMCM_SC8	VDP_GRAPH2
void HMCM_SC8 (int X1, int Y1, int X2, int Y2, void *tobuffer, unsigned char OP)	
High speed Copy the rectangle (***X1,Y1***)-(***X2,Y2***) from VRAM to the RAM buffer ****tobuffer***. ***OP*** is a standard logical operator. Use this command only with screen mode 8.	

YMMM	V1.2	VDP_GRAPH2
void YMMM(int XS, int YS, int DY, int NY, int DiRX)		
High speed copy of a part of image from VRAM to VRAM . This only copy the image part to another Y position (***DY***) The rectangle image starting at ***XS,YS,*** and ends at ***255,YS+NY*** if ***DirX*** =0 or ends at ***0,YS+NY*** if ***DiRX***=1 The image block is copied to ***XS, DY*** position. No logical operation allowed.		

HMMC	**VDP_GRAPH2**
void HMMC (void *pixeldata, int X, int Y, int DX, int DY)	
Copy the RAM ****pixeldata*** buffer to Vram ***X,Y*** position ***DX*** is Length of the zone to copy ***DY*** is Height of the zone to copy Use ***Y*** Coordinate > 256 to copy buffer on other vram page. (add 256 to Y to copy to page 1, in screen 8); No logical operation allowed.	

LMMC	V1.2	**VDP_GRAPH2**
void LMMC (void *pixeldatas, int X, int Y, int DX, int DY, unsigned char OP);		
Copy the RAM ****pixeldata*** buffer to Vram ***X,Y*** position with logical operation ***DX*** is Length of the zone to copy ***DY*** is Height of the zone to copy Use ***Y*** Coordinate > 256 to copy buffer on other page. (add 256 to Y to copy to page 1, in screen 8); ***OP*** is a standard logical operator parameter. *In Screen mode 5 or 7, if you want ot use LMMC command, you must previously tranfer data to RAM buffer with HMCM_SC8 instead of HMCM*		

HMMV	V1.2	**VDP_GRAPH2**

void HMMV(int XS, int YS, int DX, int DY, char COLOR)

High speed fillinglmmc of a rectangle box.
Rectangle top left corner is defined ***at XS,YS*** its length is ***DX*** pixels, its height is ***DY*** pixels. The color to use is defined by ***COLOR***
No logical operation allowed.
When working on screen 5 or 7, HMMV will fill 2 horizontal pixels at the same time. The ***COLOR*** variable must be divided into two blocks of 4 bits example ***COLOR l***: ***0bAAAA BBBB*** **.** The left 4 bits will be used for the left pixel color, and the 4 right bits will be used for right pixel.
If you do not need this feature, your ***COLOR*** variable can be calculated by this formula :

```
color =12;                          // use color 12
color=((color << 4) | color);       // Use color 12 for both pixels
```

LMMV	V1.2	**VDP_GRAPH2**

void LMMV(int XS, int YS, int DX, int DY, char COL, unsigned char OP)

High speed fill of a rectangle box, with logical operation.
Rectangle top left corner is defined ***at XS,YS*** its length is ***DX*** pixels, its height is ***DY*** pixels. The color to use is defined by ***COL,*** and the logical operation is ***OP.***

SPRITES

[vdp_SPRITES.h]

SpriteOn	**VDP_SPRITES**
void SpriteOn(void)	
Enables Sprites	

SpriteOff	**VDP_ SPRITES**
void SpriteOff(void)	
Disables Sprites	

Sprite8	**VDP_ SPRITES**
void Sprite8(void)	
Sets sprites size to 8x8 pixels pattern mode	

Sprite16	**VDP_ SPRITES**
void Sprite16(void)	
Sets sprites to 16x16 pixels pattern	

SpriteSmall	**VDP_ SPRITES**
void SpriteSmall(void)	
Sets normal pixel sprite size	

SpriteDouble	VDP_ SPRITES
void SpriteDouble(void)	
Sets double pixel sprite size	

SpriteReset	VDP_ SPRITES
void SpriteReset(void)	
Resets all sprites attributes and patterns	

SpriteCollision	VDP_ SPRITES
char SpriteCollision(void)	
Returns 1 in case of a sprite collision	

SpriteCollisionX	VDP_ SPRITES
char SpriteCollisionX(void)	
Return **X** position of a sprite collision. (Available only for MSX2 and upper)	

SpriteCollisionY	VDP_ SPRITES
char SpriteCollisionY(void)	
Returns **Y** position of a sprite collision (Available only for MSX2 and upper)	

SetSpritePattern	**VDP_ SPRITES**
void SetSpritePattern(char pattern_n, char* p_pattern, char s_size)	
Sets the ***pattern_n*** with ****p_pattern*** data. ***s_size*** is number of line of the pattern.	

PutSprite	**VDP_ SPRITES**
void PutSprite(char sprite_n, char pattern_n, char x, char y, char color)	
Puts the ***sprite_n*** on screen with the defined pattern (***pattern_n***) at position X and ***Y*** and color ***color***. On MSX2 and upper you must define the sprite color with *SC5SpriteColor* or *SC8SpriteColor* functions.	

Sprite32Bytes	**VDP_GRAPH2**
unsigned char *Sprite32Bytes(unsigned int *bindata)	
Uses to define 16x16 patterns. It convert a standard sprite (1-4quadr.) definition 32 char bytes 16 x 16-bit integers as bits (0b1111111100000000	

SC5SpriteColors	**VDP_ SPRITES**
void SC5SpriteColors(int spriteNumber, unsigned char *data)	
On MSX2 Screen mode 5. Defines each line of the sprite ***spriteNumber*** with a specific color, set in ****data***.	

SC8SpriteColors	**VDP_ SPRITES**
void SC8SpriteColors(int spriteNumber, unsigned char *data);	
On MSX2 Screen mode 8. Defines each line of the sprite ***spriteNumber*** with a specific color, set in ****data***.	

CIRCLE
[VDP_CIRCLE]

Graphic functions to draw circle on MSX2 graphic screen.
Can be use in complement of VDP_GRAPH2.H or VDP_GRAPH1.H

CircleFilled	**VDP_CIRCLE**
void CircleFilled(int x0, int y0, int radius, int color, int OP)	
Draws a filled circle. Center of the circle ***at x0, y0***, with a ***radius***, a ***color*** and an Operation Mode ***OP***.	

Circle	**VDP_CIRCLE**
void Circle(int x0, int y0, int radius, int color, int OP)	
Draws a circle. Center of the circle ***at x0, y0***, with a ***radius***, a ***color*** and an Operation Mode ***OP***.	

SC2CircleFilled *v1.1*	**VDP_CIRCLE**
void CircleFilled(int x0, int y0, int radius, int color)	
Only for Screen 2 mode. Draws a filled circle. Center of the circle ***at x0, y0***, with a ***radius***, and a ***color***	

SC2Circle *v1.1*	**VDP_CIRCLE**
void Circle(int x0, int y0, int radius, int color)	
Only for Screen 2 mode. Draws a filled circle. Center of the circle ***at x0, y0***, with a ***radius***, and a ***color***	

MSX-DOS 2 RAM MAPPER
[RAMMAPPER.H]

All necessary functions to be able to use full memory of the MSX computer with Memory Mapper thru the secure functions of MSX-DOS2.

InitRamMapperInfo	**RAMMAPPER**

void InitRamMapperInfo(unsigned char deviceId)

Initialization of the MSX2 Mapper device.
DeviceId must be set to 0x04 to initialize MSX-DOS2 Mapper.
InitRamMapperInfo(4) ;
After Initialization, the structure is set with all data and information about all mappers found.

```
typedef struct {
   unsigned char slot ;
    unsigned char number16KBSegments ;
    unsigned char numberFree16KBSegments ;
    unsigned char numberAllocatedSystem16KBSegments ;
    unsigned char numberUser16KBSegments ;
    unsigned char notInUse0 ;
    unsigned char notInUse1 ;
     unsigned char notInUse2 ;
} MAPPERINFOBLOCK ;
```

Get_PN	**RAMMAPPER**

unsigned char Get_PN (unsigned char page)

Gets and returns Segment Address of a Memory ***page***.
page must be 0,1,2 or 3

Put_PN	**RAMMAPPER**
void Put_PN (unsigned char page, unsigned char segment)	
Sets a specific memory ***segment*** to a page. ***page*** must be 0,1,2, or 3. ***segment*** must be one of the allocated segment	

AllocateSegment	**RAMMAPPER**
SEGMENTSTATUS *AllocateSegment(unsigned char segmentType, unsigned char slotAddress)	
Allocates next available 16Kbytes ram segment and returns information about this segment in the Status structure. typedef struct { unsigned char allocatedSegmentNumber ; unsigned char slotAddressOfMapper ; unsigned char carryFlag ; } SEGMENTSTATUS ; - segmentType must be 0 (Allocation of a user Segment) 1, means System segment - slotAddress must be set to 0 for automatic allocation **AllocateSegment(0,0) ;**	

FreeSegment	**RAMMAPPER**
***FreeSegment(unsigned char segmentType, unsigned char slotAddress)**	
Free an allocated segment	

P S G

[PSG.H]

Extended function to control PSG

Sound	**PSG**
void Sound(char reg, char value)	
Writes a value into a register of PSG (0 to 13)	

SoundFX	**PSG**
void SoundFX(char channel, FX *soundat)	
Plays a FX by a PSG Channel (0,1 or 3). Sound data is from this structure typedef struct { boolean isTone ; boolean isNoise ; unsigned int Tone ; char Noise ; unsigned int Period ; char Shape ; } FX ;	

SetChannelA	**PSG**
void SetChannelA(char channel, boolean isTone, boolean isNoise)	
Enables or disables Tone and Noise channels (0,1 or 2) IsNoise must be 1 or 0	

SilencePSG	**PSG**
void SilencePSG()	
Plays off all three PSG Channels	

GetSound	**PSG**
char GetSound(char reg)	
Reads a PSG Register	

SetTonePeriod	**PSG**
void SetTonePeriod(char channel, unsigned int period)	
Sets Tone Period for any channel (0,1 or 2) period must be between 0 and 4095	

SetNoisePeriod	**PSG**
void SetNoisePeriod(char period)	
Sets Noise Period. Period must be between 0 and 31	

SetEnvelopePeriod	**PSG**
void SetEnvelopePeriod(unsigned int period)	
Sets Envelope Period. Period must be between 0 and 65535	

SetVolume	**PSG**
void SetVolume(char channel, char volume)	
Sets volume of a channel (0,1 or 2) volume must be between 0 and 15, or 16 to activate envelope	

SetChannel	**PSG**
void SetChannel(char channel, boolean isTone, boolean isNoise)	
Mixer. Enables or disables Tone and Noise channels (0,1 or 2) Tone and State must be 0 or 1	

PlayEnvelope	**PSG**
void PlayEnvelope(char shape)	
Plays the sound on channels that have a volume of 16. Envelope shape must be between 0 and 15	

AYFX PLAYER

[ayfx_player.h]

Use the AYFX sound editor to edit and create AYFX Sound Banks.
Before using ayFX Player you must load a sound bank file to memory
(sound banks are .afb file).

InitFX	**ayfx_player**
void InitFX (void)	
Initialization of the ayFX player	

PlayFX	**ayfx_player**
char PlayFX (unsigned char nfx)	
Plays the sound-FX ***nfx*** from the Sound bank.	

UpdateFX	**ayfx_player**
void UpdateFX (void)	
Updates the Sound FX playing. UpdateFX must be use inside the main loop of a game. It update the PSG registers will a sound FX must be played.	

TestFX	**ayfx_player**
char TestFX (void)	
Tests if a sound FX is currently playing. Return 1 if true. If the test is valid, you must Update the playing with UpdateFX.	

FreeFX	ayfx_player
void FreeFX (void)	
Free the Memory used by the ayFX player.	

MUSIC PT3 REPLAYER

[PT3REPLAYER.H]

Use those function to play PT3 music files.

PT3Init	**ayfx_player**
void PT3Init (unsigned char *songAddress , unsigned char loop)	
Initialization of the PT3 replayer. Enter the address where the pt3 file is loaded. loop must be : 0 if you want the music loop continuously	

PT3Play	**ayfx_player**
void PT3Play(void)	
Actualizes the music playing, inside a main loop.	

PT3Rout	**ayfx_player**
void PT3Rout(void)	
Prepares data to be played. (Part of the Playing routine), inside a main loop.	

PT3Mute	**ayfx_player**
void PT3Mute(void)	
Mutes the music.	

The C standard functions
(included in SDCC package)

CTYPE.H	
Int isalnum (int c)	Check whether the given character is alphanumeric
Int isalpha(int c)	Check whether the given character is alphabetic
Int iscntrl(int c)	Check whether the given character is a control character
Int isdigit(int c)	Check whether the given character is decimal digit
Int isgraph(int c)	Check whether the given character has graphical representation
islower(int c)	Check if given character is in lower case
isupper (int c)	Check if given character is in upper case
isprint (int c)	Check if given character is printable
ispunct (int c)	Check if given character is punctuation
isspace (int c)	Check if given character is is a space
isxdigit (int c)	Check if given character is a Hexadecimal digit
isblank (int c)	Check if given character is blank
tolower (int c)	Convert given character to lower case
toupper (int c)	Convert given character to upper case

MATH.H	
sinf(float x)	Return the sine of a radian angle
cosf(float x)	Return the cosine of a radian angle
tanf(float x)	Return the tangent of X
asinf(float x)	Return the arc sine of X radian
acosf(float x)	Return the arc cosine of X
atanf(float x)	Return the arc tangent of X
atan2f(float x, float y)	Returns the arc tangent in radians of y/x based on the signs of both values to determine the correct quadrant.
sinhf(float x)	Return the hyperbolic sine of X
coshf(float x)	Return the hyperbolic cosine of X
tanhf(float x)	Return the hyperbolic tangent of X
expf(float x)	Return the value of x raised to Xth power
logf(float x)	Return the natural logarithm of x
log10f(float x)	Returns the common logarithm (base-10 logarithm) of X
powf(float x, float y)	Returns X raised to the power of Y
sqrtf(float a)	Returns the square root of X
fabsf(float x)	Returns the absolute value of x
frexpf(float x, int *pw2)	The returned value is the mantissa and the integer pointed to by exponent is the exponent. The resultant value is x = mantissa * 2 ^ exponent.
ldexpf(float x, int pw2)	Returns x multiplied by 2 raised to the power of exponent.
ceilf(float x)	Returns the smallest integer value greater than or equal to X

floorf(float x)	Returns the largest integer value less than or equal to X
modff(float x, float * y)	Returns the remainder of X divided by Y

STDIO.H

	Library Variable & Description
1	**size_t** This is the unsigned integral type and is the result of the **sizeof**keyword.
2	**FILE** This is an object type suitable for storing information for a file stream.
3	**fpos_t** This is an object type suitable for storing any position in a file.

	Macro & Description
1	**NULL** This macro is the value of a null pointer constant.
2	**_IOFBF, _IOLBF** and **_IONBF** These are the macros which expand to integral constant expressions with distinct values and suitable for the use as third argument to the **setvbuf** function.
3	**BUFSIZ** This macro is an integer, which represents the size of the buffer used by the **setbuf** function.

4	**EOF** This macro is a negative integer, which indicates that the end-of-file has been reached.
5	**FOPEN_MAX** This macro is an integer, which represents the maximum number of files that the system can guarantee to be opened simultaneously.
6	**FILENAME_MAX** This macro is an integer, which represents the longest length of a char array suitable for holding the longest possible filename. If the implementation imposes no limit, then this value should be the recommended maximum value.
7	**L_tmpnam** This macro is an integer, which represents the longest length of a char array suitable for holding the longest possible temporary filename created by the **tmpnam** function.
8	**SEEK_CUR, SEEK_END,** and **SEEK_SET** These macros are used in the **fseek** function to locate different positions in a file.
9	**TMP_MAX** This macro is the maximum number of unique filenames that the function **tmpnam** can generate.
10	**stderr, stdin,** and **stdout** These macros are pointers to FILE types which correspond to the standard error, standard input, and standard output streams.

Functions	
printf (const char *,...)	Sends formatted output to stdout
vprintf (const char *, va_list)	Sends formatted output to a stream using an argument list.
sprintf (char *, const char *, ...)	Sends formatted output to a string.
vsprintf (char *, const char *, va_list)	Sends formatted output to a string using an argument list.
puts(const char *)	Writes a string to stdout up to but not including the null character. A newline character is appended to the output.
***gets(char *)**	Reads a line from stdin and stores it into the string pointed. It stops when either the newline character is read
getchar(void)	Gets a character (an unsigned char) from stdin.
putchar(char)	Writes a character (an unsigned char) specified by the argument char to stdout.

STDLIB.H

	Library Variable & Description
1	**size_t** This is the unsigned integral type and is the result of the **sizeof**keyword.
2	**wchar_t** This is an integer type of the size of a **wide** character constant.
3	**div_t** This is the structure returned by the **div** function.
4	**ldiv_t** This is the structure returned by the **ldiv** function.

	Lybrary Macro & Description
1	**NULL** This macro is the value of a null pointer constant.
2	**EXIT_FAILURE** This is the value for the exit function to return in case of failure.
3	**EXIT_SUCCESS** This is the value for the exit function to return in case of success.
4	**RAND_MAX** This macro is the maximum value returned by the rand function.
5	**MB_CUR_MAX** This macro is the maximum number of bytes in a multi-byte character set which cannot be larger than MB_LEN_MAX.

Functions	
atof (const char *nptr)	Converts the string pointed to, by the argument *str* to a floating-point number (type double).
atoi (const char *nptr)	Converts the string pointed to, by the argument *str* to an integer (type int).
atol (const char *nptr)	Converts the string pointed to, by the argument *str* to a long integer (type long int).
_itoa(int x, char p*, unsigned char base)	Convert the x integer number to a string pointer, on base 2,10, or 8
_ultoa(unsigned long x, char p*, unsigned char base)	Convert the x unsigned long number to a string pointer, on base 2,10, or 8
_ltoa(long x, char p*, unsigned char base)	Convert the x long number to a string pointer, on base 2,10, or 8
rand(void)	Returns a pseudo-random number in the range of 0 to *RAND_MAX*.
srand(unsigned int seed)	This function seeds the random number generator used by the function **rand**
calloc (size_t nmemb, size_t size)	Allocates the requested memory with requested size and returns a pointer to it. Calloc initialise memory cells to 0.
malloc (size_t size)	Allocates the requested memory and returns a pointer to it
realloc (void *ptr, size_t size)	Attempts to resize the memory block pointed to by ptr that was previously allocated with a call to *malloc* or *calloc*.
free (void * ptr)	Deallocates the memory previously allocated by a call to *calloc, malloc,* or *realloc*.
abs(int j)	Returns the absolute value of x.

STRING.H

	Library Variable & Description
1	**size_t** This is the unsigned integral type and is the result of the **sizeof**keyword.

	Library Macro & Description
1	**NULL** This macro is the value of a null pointer constant.

memcpy (void * dest, const void *src, size_t n)	Copies n characters from src to *dest*.
memmove (void *dest, const void *src, size_t n)	Another function to copy n characters from *str2* to *str1*.
strcpy (char *dest, const char *src)	Copies the string pointed to, by *src* to *dest*.
strncpy(char * dest, const char * src, size_t n)	Copies up to n characters from the string pointed to, by *src* to *dest*.
strcat (char * dest, const char *src);	Appends the string pointed to, by *src* to the end of the string pointed to by *dest*.
strncat(char *dest, const char *src, size_t n)	Appends the string pointed to, by *src* to the end of the string pointed to, by *dest* up to n characters long.
memcmp (const void *s1, const	Compares the first n bytes

void *s2, size_t n)	of *str1* and *str2*.
strcmp (const char *s1, const char *s2)	Compares the string pointed to, by *str1* to the string pointed to by *str2*.
strncmp(const char *s1, const char *s2, size_t n)	Compares at most the first n bytes of *str1* and *str2*.
strxfrm(char *dest, const char *src, size_t n)	Transforms the first **n** characters of the string **src** into current locale and places them in the string **dest**.
memchr (const void *s, int c, size_t n)	Searches for the first occurrence of the character c (an unsigned char) in the first n bytes of the string pointed to, by the argument *str*.
strchr (const char *s, int c)	Searches for the first occurrence of the character c (an unsigned char) in the string pointed to, by the argument *str*.
strpbrk(const char *str1, const char *str2)	Finds the first character in the string *str1* that matches any character specified in *str2*.
strrchr(const char *str, char c)	Searches for the last occurrence of the character c (an unsigned char) in the string pointed to by the argument *str*.
strcspn(const char *str, const char *reject)	Calculates the length of the initial segment of str1 which consists entirely of characters not in str2.
strpbrk(const char *str, const char *accept)	Finds the first character in the string *str1* that matches any character specified in *str2*.
strrchr(const char *s, int c)	Searches for the last occurrence of the character c (an unsigned char) in the string pointed to by the argument *str*.
strspn (const char *str, const char *accept)	Calculates the length of the initial segment of *str1* which consists entirely

	of characters in *str2*.
strstr (const char *haystack, const char *needle)	Finds the first occurrence of the entire string *needle* (not including the terminating null character) which appears in the string *haystack*.
strtok (char *str, const char *delim)	Breaks string *str* into a series of tokens separated by *delim*.
memset (void *s, int c, size_t n)	Copies the character c (an unsigned char) to the first n characters of the string pointed to, by the argument *str*.
strlen (const char *s)	Return the length of a string

TIME.H

	Library Variable & Description
1	**size_t** This is the unsigned integral type and is the result of the **sizeof**keyword.
2	**clock_t** This is a type suitable for storing the processor time.
3	**time_t is** This is a type suitable for storing the calendar time.
4	**struct tm** This is a structure used to hold the time and date.

time(time_t *t)	Calculates the current calendar time and encodes it into time_t format.
gmtime(time_t *timep)	The value of timer is broken up into the structure tm and expressed in Coordinated Universal Time (UTC) also known as Greenwich Mean Time (GMT).
mktime(struct tm *timeptr)	Converts the structure pointed to by timeptr into a time_t value according to the local time zone.
ctime(time_t *timep)	Returns a string representing the local time based on the argument timer.

STDARG.H

	Library Variable & Description
1	**va_list** This is a type suitable for holding information needed by the three macros **va_start(), va_arg()** and **va_end()**.

va_start(marker, first)	This macro initializes **ap** variable to be used with the **va_arg** and **va_end** macros. The **last_arg** is the last known fixed argument being given to the function i.e. the argument before the ellipsis.
va_arg(marker, type)	This macro retrieves the next argument in the parameter list of the function with type **type**.
va_copy(dest, src)	
va_end(marker)	his macro allows a function with variable arguments which used the **va_start** macro to return. If **va_end** is not called before returning from the function, the result is undefined.

Publish and distribute your game

You just finished a game, and you're very proud of it? Do you want to distribute it or sell it around the world?

You can trust REPROFACTORY to distribute your game, and receive your royalties on sales.

With more than 550 customers on 5 continents, you will be assured that your game will be played by dozens of MSX users.

REPROFACTORY can manufacture the cartridges, and offer you to realize a packaging, and deal with sales; or just one of these steps if you wish.

Example of games sold on www.Repro-Factory.com :

Muffie's Tutankham & Conversion Classic...

30,00 €

In Stock

Joust - Arcade

38,00 €

In Stock

Gyruss - Arcade

38,00 €

In Stock

Do not hesitate to contact REPROFACTORY on **www.repro-factory.com**

Contacts

I hope you will enjoy the FUSION-C Library, and you will use it to create new games and tools for the MSX computers.

If you have any questions about FUSION-C please contact me at :

ericb59@ebsoft.fr