FUSION-C

S D K

C PROGRAMMING FOR M S X

by Eric Boez & Fernando Garcia

Complete Journey for FUSION-C v1.3

(FUSION-C v1.0, v1.1, 1.2, v1.3)

EBSoft
Edition

EBSoft Edition

Book written by Eric Boez

ISBN: 9781730828614

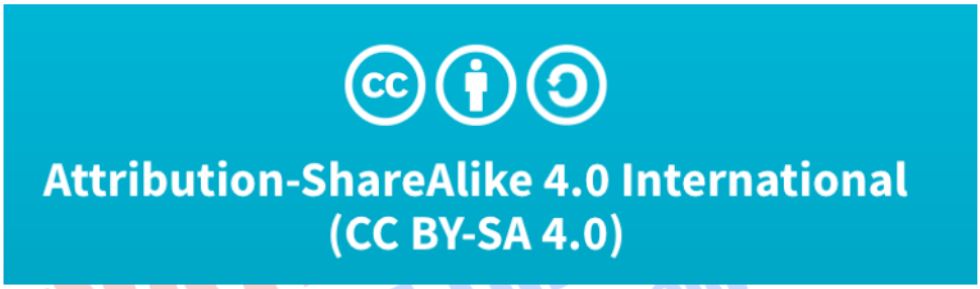
Book revision – October 17, 2021

SDK revision: R21010

« C is Future of MSX »
Jorge Torres Chacón

FUSION-C is free, it took many hours of work to create this library. If you like FUSION-C, and you want to change features, to add more, share your work with the community. 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 everyone.

FUSION-C is free software; it comes without any warranty.
You can use, modify, share it under the terms of
Creative Commons CC BY_SA 4.0 license



What means this licence?
Share: You can copy and redistribute the material in any medium or format
Adapt: You can remix, transform, and build upon the material for any purpose, even commercially.

Your obligations for using this library inside your own software:
You must, quote the original name of the library, the authors, and provide a download link to this original software.

The licensor cannot revoke these freedoms as long as you follow the license terms.

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 permission 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.

AYFX Editor by Shiru.
https://shiru.untergrund.net/software.shtml
Source code and updates:
https://github.com/Threetwosevensixseven/ayfxedit-improved

Bin2froh by Ming Chen
Source code and updates:
https://sourceforge.net/projects/bin2froh

MSX DiskImage by <unknown author>

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

Vortex Tracker II by Sergey Bulba

DSKTOOL v1.3 by Ricardo Bittencourt, Tony Cruise, Natalia Pujol
Source code and updates:
https://github.com/nataliapc/MSX_devs

Hex2Bin 2.5 **by** Jacques Pelletier
https://sourceforge.net/projects/hex2bin/

Disk2Rom 0.8 by Vincent van Dam

Sprite SX by 303bcm

OpenMSX v17.0 by openMSX Team
Source code and updates:
https://openmsx.org

Sc2GraphXConvertor v0.5 by Eric Boez & Leandro Xorreia

RLEWbCompressor 1.0b by Eric Boez & Aorante

BitBuster v1.2 by Team Bomba

Image to Sprite Editor by Sylvain Cregut

Sprite Path Editor by Sylvain Cregut

MSX-DOS v1.03 by Microsoft

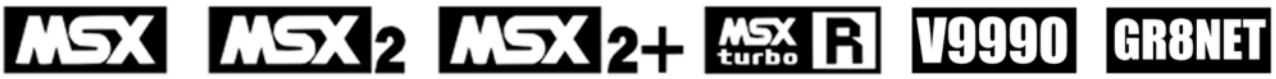
MSX-DOS v2.30 by ASCII

Other contributions
Illustrated pictures by Eric Boez
Humorous illustrations by Carali

For their help or contribution

I want to thank:

Sylvain Cregut
Mvac7/303bcn
Nestor Soriano
T.Hara
Jorge Torres Chacón
GDX
Grauw
Javi Lavandeira
PingPong
Pasi Kettunen
The openMSX dev crew
Dolphin_Soft
Oduvaldo Pavan Junior



What is « FUSION-C »?

Fusion-C is a C library with which you can program software and games for MSX computers under MSX-DOS, (MSX1, MSX2, MSX2+ and MSX Turbo-R), it uses the cross C compiler SDCC (Small Device C Compiler).
Face the fact there is no real complete and documented tools for the C development on MSX, With the major help of Fernando Garcia, I decided to reshape and to complete all SDCC code for MSX that were available here and there, to something coherent and easy to use.

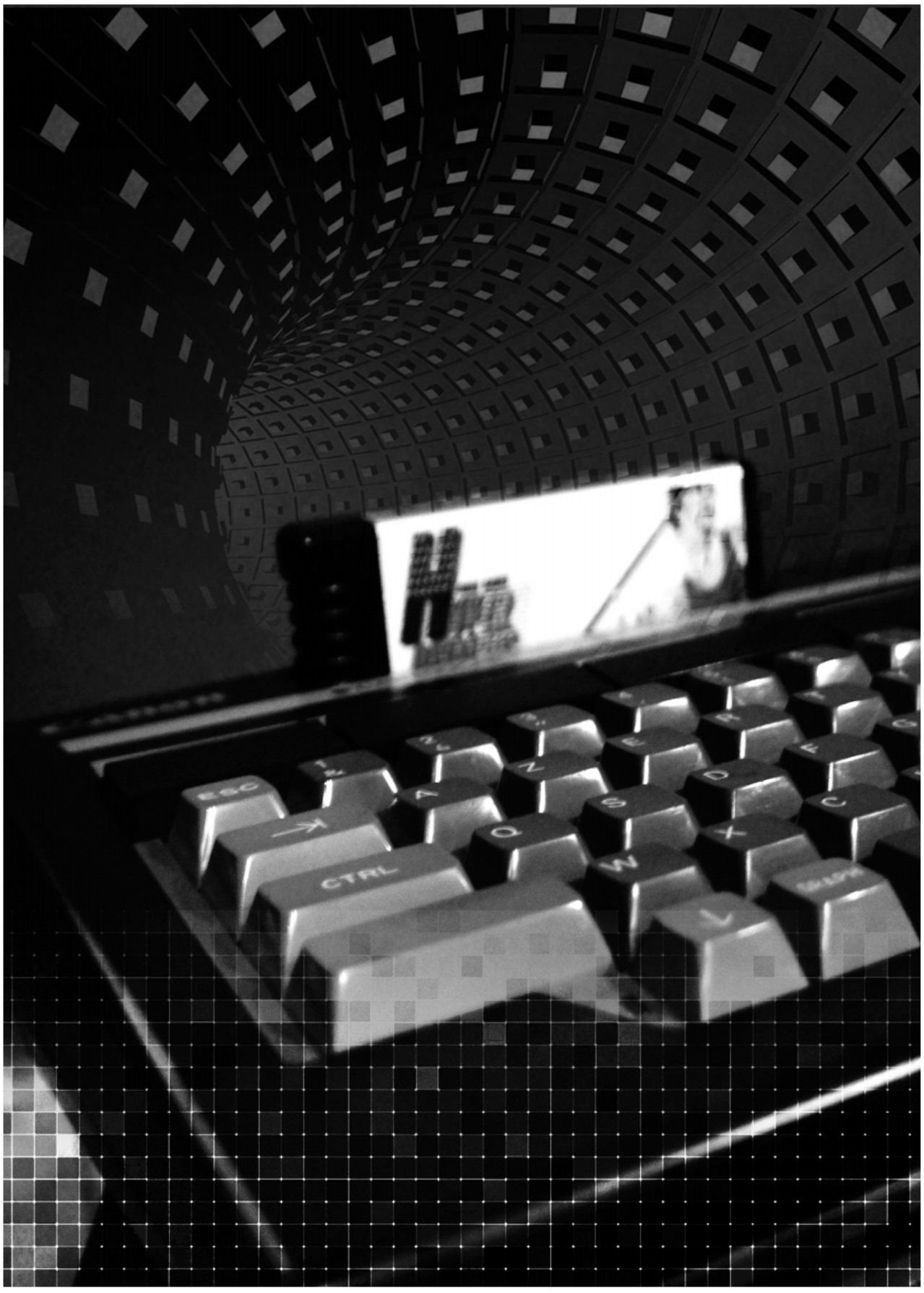
The name "FUSION" was chosen because this library was originally based on some other Libraries, or part of libraries found separately here and there without any homogenization. FUSION-C is also composed with new source code, new commands and routines that will make your life easier in MSX programming.
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, portions of this code are 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 his routines are included in this Library. Other routines comes from the work of Néstor Soriano, Jorge Torres Chacon, MVAC7, Eric Boez, and Fernando Garcia.

The purpose is to bring to MSX users, tools and documentation 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.

More than a year after the release of the first version of FUSION-C, the library has evolved a lot. Other actors from the MSX scene joined me to advance the library. I also reworked a lot of the original codes to improve them and add new functions.

SDCC (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

We hope you will enjoy coding for MSX computers with Fusion-C.


ESC
CTRL

What's new in version 1.3

The Fusion-C's *folder **./source/lib*** has been reorganized, as most of the source codes.
A ***_Source_Code_Catalog.pdf*** file has been created in order to easily find where to find the source code of a function.
The "*Working Folder*" folder has been **renamed** to "*WorkingFolder*", the content of the *fusion-c/example* folder has been reorganized, and the examples reworked. A lot of source code where rewrote and optimized for speed performance.

MAJOR INCOMPATIBILITY
Because some functions were removed in FUSION-C 1.3 your code made for FUSION-C 1.2 or FUSION-C 1.1 may not be 100% compatible with this new release.
Please check the removed functions section.
Major problems may come from the suppression of HMCM and HMCM_SC8.
HMCM & HMCM_SC8 were replaced by LMCM5 and LMCM8. They have the same use as old ones, but the last parameter was removed because it was useless. To fix any issue, rename old HMCM and HMCM_SC8 by LMCM5 and LMCM8, then remove the last parameter of the old function (OP).

New AYFX Stand alone driver.
New PT3 + AYFX driver.
New Pattern rotation and Pattern Flip functions.
New Fast Sprite Management
New Fast Paint and Fast Pset functions
New Debug possibilities with openMSX Debugger
New Interrupt handler management
New development with MSX Hard-Drive support

- Renamed "Working Folder" to "WorkingFolder"
- Introducing Fusion-C Environment variables (msx_fusion.h)

_SpriteOn
_SpriteSize
_SpriteMag
_ActivePage
_DisplayPage
_VDPFreq
_VDPLines
_ForgroundColor
_BackgroundColor
_BorderColor
_ScreenMode
_SpritePatternAddr
_SpriteAttribAddr
_SpriteColorAddr
_WidthScreen0
_WidthScreen1
_FusionVer
_FusionRev

- added functions to the library and to manual/book:

SC2BoxLine (Replacing Rect)	(vdp_graph1.h)
SC2BoxFill	(vdp_graph1.h)
BitReturn	(msx_fusion.h)
BitReset	(msx_fusion.h)
Vsynch	(msx_fusion.h)
SetScrollMask	(msx_fusion.h)
SetScrollDouble	(msx_fusion.h)
JoystickReadTo	(msx_fusion.h)
StrToLower	(msx_fusion.h)
StrToUpper	(msx_fusion.h)
TurboMode	(msx_fusion.h)
SpriteOverlap	(vdp_sprites.h)
SpriteOverlapId	(vdp_sprites.h)
Pattern8RotationRam	(vdp_sprites.h)
Pattern8RotationVram	(vdp_sprites.h)
Pattern16RotationRam	(vdp_sprites.h)
Pattern16RotationVram	(vdp_sprites.h)
Pattern8FlipRam	(vdp_sprites.h)

Pattern16FlipRam	(vdp_sprites.h)
Sprite8FlipVram	(vdp_sprites.h)
Pattern16FlipVram	(vdp_sprites.h)
PatternRotation	(vdp_sprites.h)
PatternHFlip	(vdp_sprites.h)
PatternVFlip	(vdp_sprites.h)
SpriteFollow	(vdp_sprites.h)
LMCM5	(vdp_graph2.h)
LMCM8	(vdp_graph2.h)
VDPLINE	(vdp_graph2.h)
TurboMode	(msx_fusion.h)
Paint (New Fast Paint for MSX2)	(vdp_paint.h)
SetPaintBuffer	(vdp_paint.h)
SetColor	(msx_fusion.h)
fVDP	(vdp_graph2.h)
Polygon	(vdp_graph2.h)
Rkeys	(msx_fusion.h)
Fkeys	(msx_fusion.h)
ReadKeyboardType	(msx_fusion.h)
VDPalternate	(msx_fusion.h)
VDPinterlace	(msx_fusion.h)
SetExpandVDPcmd	(msx_fusion.h)
SetScreen10	(msx_fusion.h)
SetScreen12	(msx_fusion.h)
SetTransparent	(msx_fusion.h)
SetAdjust	(msx_fusion.h)
fPutSprite	(msx_fusion.h)
InitVDPInterruptHandler	(msx_fusion.h)
EndVDPInterruptHandler	(msx_fusion.h)
PT3FXInit	(ayfx_player.h
PT3FXPlay	(pt3replayer.h)
PT3FXRout	(pt3replayer.h)
CovoxPlayRam	(msx_fusion.h)
FcbDelete	(msx_fusion.h)
CallSub	(msx_fusion.h)
SaveScreenBoot	(msx_fusion.h)
SetRamDisk	(msx_fusion.h)

- Fixed error or rewritten functions:

SC2Pset (500 % Faster than previous version)	(vdp_graph1.h)
SC2Line (500 % Faster than previous version)	(vdp_graph1.h)
SC2Point	(vdp_graph1.h)
SetScrollH	(vdp_graph1.h)
SetScrollV	(vdp_graph1.h)
SetSpriteColors	(vdp_sprites.h)
YMMM Fixed definition last parameter removed	(vdp_graph2.h)
GetKeyMatrix	(msx_fusion.h)
InitInterruptHandler	(msx_fusion.h)
EndInterruptHandler	(msx_fusion.h)
Lseek	(io.h)
MakeDir	(io.h)
ChangeDir	(io.h)
RemoveDir	(io.h)
InitFX	(ayfx_player.h)
PlayFX	(ayfx_player.h)
UpdateFX	(ayfx_player.h)
StopFX	(ayfx_player.h)
PT3Init	(pt3replayer.h)
PT3Play	(pt3replayer.h)
PT3Rout	(pt3replayer.h)
PT3Mute	(pt3replayer.h)

- Removed functions:

SC2Rect	Replaced identically by	SC2BoxLine

DosCLS	Replaced identically by	Cls
SC5SpriteColors	Replaced identically by	SetSpriteColors
SC8SpriteColors	Replaced identically by	SetSpriteColors
HMCM	Replaced by LMCM5	(vdp_graph2.h)
HMCM_SC8	Replaced by LMCM8	(vdp_graph2.h)
WriteScr	**Totally removed**	(vdp_graph2.h)
ReadScr	**Totally removed**	(vdp_graph2.h)
KeyboardRead	Replaced by	Rkeys
SetInterruptHandler	**Totally removed**	(msx_fusion.h)
TestFX	Totally remove	
FreeFX	Totally remove	
IntDos	Replaced by	CallDos
IntBios	Replaced by	Callbios

- Renamed functions :

fcb_open	to	FcbOpen
fcb_create	to	FcbCreate
fcb_close	to	FcbClose
fcb_read	to	FcbRead
fcb_write	to	FcbWrite
fcb_find_first	to	FcbFindFirst
fcb_find_next	to	FcbFindNext
SetSC5ColorPalette	to	SetColorPalette
SetSC5Palette	to	SetPalette
RestoreSC5Palette	to	RestorePalette
VDPLineSwitch	to	VDPlineSwitch
PSGRead	to	PSGread
SetVDPWrite	to	SetVDPwrite
SetVDPRead	to	SetVDPread
CovoxPlay	to	CovoxPlayVram

- Added Example :

code examples completely reworked

Installing the Tools Chain - SDK

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 **MacOS** Apple's computer, and even on **Linux**.

The SDK is composed of the following elements

- **OpenMSX Emulator**
  Since Fusion-C v1.3, open MSX is part of Fusion-C SDK

- **OpenMSX Machines System Roms**
  The system roms collection of all MSX machines, and peripherals.

- **SDCC** (Small Device C Compiler)
  The heart of the SDK.

- **Hex To bin tool**
  This tool generates the the binary file you will execute on the MSX.

- **The FUSION-C Library and scripts environnement**
  Fusion-C SDK is composed of a library and multiple script, tools, and code examples

- **Sublime Text code editor.**
  The editor to type your code

The code editor used is **Sublime Text 3**. It is a very nice and professional code editor, it can be freely used and available on the three major operating systems.
If you feel more comfortable with another code editor, you can change to the one you are used to. However, you will have to configure it yourself so that it can launch the compilation script; or start the compilation manually.

Download the necessary files

1- Download the « SDCC 4.0 Package » for your operating system :

Historically I advised to use the SDCC v3.6 version. But today it will no longer work properly on computers with a 100% 64-bit OS, such as MacOS 10.15 Catalina, this is why an upgrade **to SDCC 4.0** is recommended.

https://sourceforge.net/projects/sdcc/files/

2- Download « Sublime Text 3» for your operating system :

Many code editors may be suitable for editing C code. I suggest using the free version of « **Sublime Text 3** ». The installation and the way of working explained in this book, has been designed for this code editor. I suggest you install this editor, to follow the process of this book, and possibly change your code editor later, if you prefer to work with another tool.

https://www.sublimetext.com/3

3- Download « openMSX Machines Rom files »

For copyright reasons, the MSX System Rom Files cannot be provided with OpenMSX, or Fusion-C SDK. You can download the full pack of System Rom files for OpenMSX from here:

http://www.ebsoft.fr/msx/roms/OpenMSX_ROMS.zip

Or

https://download.file-hunter.com/System%20ROMs/Full%20Set%20System%20ROM's%20(OpenMSX).zip

4- Download « Fusion-C SDK »

The latest version of **« Fusion-C SDK »** can be downloaded from one of these 2 official sources.

https://www.ebsoft.fr/shop/fr/19-fusion-c

Or

https://github.com/ericb59

Manual installation for Windows

1 - The Working Folder

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

- Unpack the archive, copy the « **WorkingFolder** » as is, where you want on your computer, that's all! For example, copy it to your Desktop.

2 - Install Sublime Text 3

Install the code editor as any other application.

We need to set up the automatic compilation command. Open Sublime Text, and go to the **Tools's menu**, and choose « **Build System > New Build System** ».

Inside the new opened window enter this text:

```
{
	"cmd": ["$file_path\\fusion-c\\build.bat", "$file_name"]
}
```

Now save this file to:

« "**C:\Users\<*USER NAME*>\AppData\Roaming\Sublime Text 3\Packages\User** »

name it « **sdcc-build.sublime-build** »

Close Sublime Text, and open it again. Go to the **Tools's menu**, choose « **Build System** », now in the list you must see your « **sdcc-build** »; click on it to choose it as the default build system. SDCC is nox able to call the compilation script.

3 - Install Hex2Bin

Use Windows to navigate to the folder « **WorkingFolder/Tools/Hex2Bin/Hex2bin Windows/** », copy the file « **hex2bin.exe** » and paste it inside the folder « **C:\Program Files\SDCC\bin**»

4 - Add the System Roms to OpenMSX

OpenMSX is pre-installed inside the « **WorkingFolder** », but it is not provided with the MSX machines system ROMs files. Unpack the **OpenMSX_ROMS.zip** you previously downloaded, and copy the « **systemroms** » folder inside **« ./WorkingFolder/openMSX/share/ »**.

The new folder replaces the old one.

5 - Install SDCC 4.0

Use the SDCC setup you previously downloaded. Do not change the default parameters the installer offers. All the SDCC files will be installed inside the folder: « **C:\Program Files\SDCC** ».

Now open a Dos window and type above commands:

```
> CD C:\Program Files\SDCC\lib\z80
> 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
```

6 - Verification

Inside the Dos window type:

```
> sdcc -v
```

SDCC must answer, and show you its version number

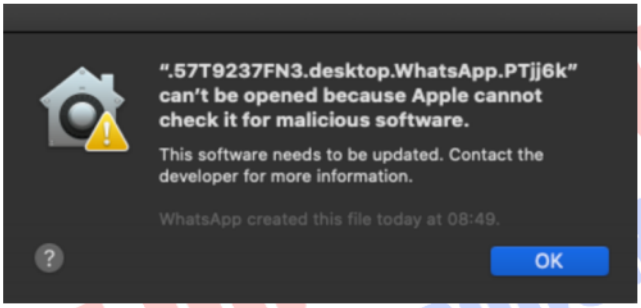
```
> hex2bin
```

Hex2Bin must show you its help page

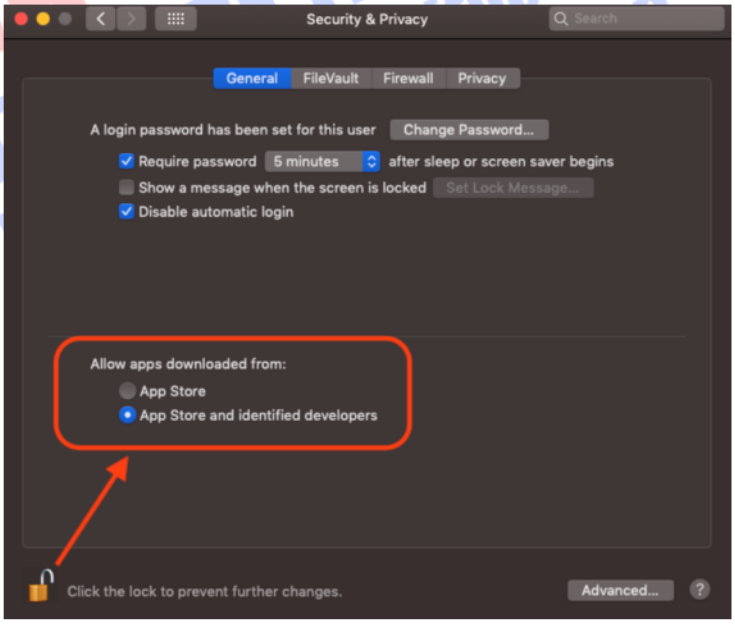
Manual installation for MacOS

0 – ⚠ Note for MacOS Catalina users (10.15 and superior)

We first have to take back control of MacOS Catalina's security system.
Since MacOS CATALINA, (10.15 and superior), Apple has tightened up the security rules on the execution of softwares which are not referenced on the AppStore, or which comes from developers not registered with Apple. This causes a lot of problems for homebrew softwares and for tools that come from the Linux world. If you ever see this window, you know what I'm talking about!



Before going any further, I suggest you to take back control of your MacOS 's security system.
Open the **System preference** and go to **Security & Privacy**.



You can see there are only 2 options, we can add a third option to enable the execution of programs coming from anywhere. To do that, open a **Shell/Terminal**'s window and enter this command line:

```
> sudo spctl --master-disable
```

Check the Gatekeeper is disabled by entering this command line:

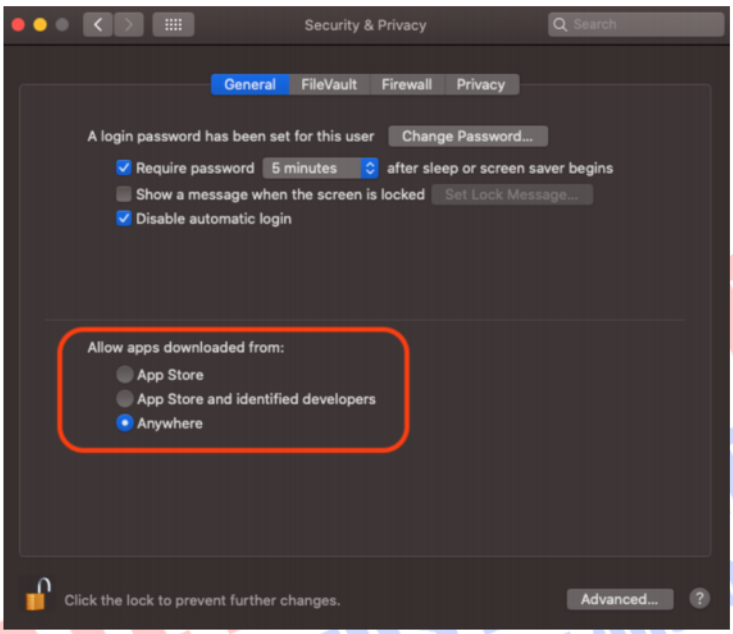
```
> sudo spctl --status
```

The result must be: ***assessments disabled***

Let's take advantage that the terminal is open to allow the FINDER to display the hidden files of the system, that will be useful to us a little later.
Type the above commands:

```
> defaults write com.apple.finder AppleShowAllFiles YES
> killall Finder
```

Now, close the **Security & Privacy** window, and open it again, you must now see the Third option appear.



During the installation of **Fusion-C**, and up to your first compilation, I suggest you to choose this 3rd option, « **Anywhere** ». Once an unidentified authored program has been run, it will continue to run without annoying you, even if you come back to a more restricted security mode, so you can come back to a more secured system when all **Fusion-C**'s components will be started at least one time.

1 - The Working Folder

The Fusion-C Library comes with source codes, tools, examples, scripts, inside a ready to use «**WorkingFolder**»
- Unpack the archive, copy the « **WorkingFolder** » as is, where you want on your computer, that's all! For example, copy it to your Desktop.

2 - Install Hex2Bin

open a **Shell/Terminal'**s window, and type: **cd** [space] *(This means, type the letter 'c', 'd' and 'Space Barre')*
Use the Finder, navigate to the folder « **WorkingFolder/Tools/Hex2Bin/** », you must see a folder named « **Hex2bin-2.5 Mac**». Drag and drop this folder to the **Terminal** window. The full path of the folder must now be written on the Terminal. Press the "**Enter key**"
Now type this command inside the terminal's window:

```
> make install MAN_DIR=/usr/local/share/man/man1
```

This will create the executable and install ***Hex2Bin*** on your system.
If you haven't done previously, we will also take the opportunity to allow the FINDER to show the hidden files stored on the hard-drive… Type the above commands:

```
> defaults write com.apple.finder AppleShowAllFiles YES
> killall Finder
```

3 - Install Sublime Text 3

Open the **.dmg** file you previously downloaded and drag the **Sublime Text application** to your « **Applications** » folder.

We need to set the automatic compilation command. Open Sublime Text, and go to the **Tools's menu**, and choose « **Build System > New Build System** »

In the new opened window enter this text:

```
{
    "shell_cmd": "./fusion-c/build.sh $file_name",
    "working_dir": "$file_path"
}
```

Save this file to:

« **/Users/*<YOUR USER NAME>*/Library/Application Support/Sublime Text/Packages/User/**»

name it « **MSX-SDCC.sublime-build** »

⚠ ***Note:*** If the dialog box doesn't show the hidden folders, press keys:

Close Sublime Text, and open it again. Go to the **Tools's menu**, choose « **Build System** ». Now you must see your « **MSX-SDCC** ». Click on it to choose it as the default build system.

4 - Add the System Roms to OpenMSX

OpenMSX is pre-installed inside the « **WorkingFolder** », but it is not provided with the MSX machines system ROMs files. Unpack the **OpenMSX_ROMS.zip** you previously downloaded, and copy the « **systemroms** » folder inside **« ./WorkingFolder/openMSX/share/ ».**

The new folder replaces the old one.

5 - Install SDCC 4.0

Inside a Terminal window, and go to the directory where you previously downloaded the archive. You can do that easily, by typing CD (+space) in the Terminal's window, then drag the **folder** where the archive is and drop it over the Terminal's windows, then press enter.

Extract the files from the archive with this Shell command:

```
> tar xjf sdcc-4.0.0-x86_64-apple-macosx.tar.bz2
```

The filename may change, it depends on the version.

```
> cd sdcc
```

The folder name may change, it depends on the version.

```
> cp -r * /usr/local
```

This will install all necessary files on your system.

Now type the following commands to remove incompatible commands:

```
> cd /usr/local/share/sdcc/lib/z80/
> 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 need to grant new permissions to the compilation script.

Inside the Terminal's window type this command: **chmod +x [space]**

```
> chmod +x
```

with a space at the end, after the 'x' (and do NOT press the enter key yet).

From the finder, open the **« WorkingFolder/fusion-c »** folder and identify the file « **build.sh** » drag it, and drop it to the Terminal's window. The file path and filename should automatically fill in. Click on the Terminal's window and press enter to validate the new permission to the script.

If you want to hide the system files again, and go back to the initial configuration type this

```
> defaults write com.apple.finder AppleShowAllFiles NO
> killall Finder
```

6 - Verification

Inside the Terminal window type:

```
> sdcc -v
```

SDCC must answer, and show you its version number

```
> hex2bin
```

Hex2Bin must show you its help page

SDK Folder Architecture

The «**WorkingFolder**» folder contains all the files of the library, as well as all the necessary tools, such as the emulator, the development tools, the files emulating the floppy disks or hard disk of a real MSX...

Content of the 2 first levels of **WorkingFolder**

```
WorkingFolder
    |_ dsk
        |_dska
        |_dskb
        |_export
        |_hda-1
        |_hda-2
    |_ fusion-c
        |_examples
        |_header
        |_include
        |_lib
        |_source
    |_build scripts
    |_ openMSX
        |_Catapult
        |_codec
        |_ doc
        |_hard-drive
        |_MSX_config
        |_platforms
        |_share
    |_ Tools
            |_ … Various folder and files
    |_ hello.c
```

dsk	Destination folders of compiled programs.
… dska & dskb	Must contain the files you want on floppy drives A and B
… hda-1 & hda-2	These folders must contain the files you want to import to a virtual hard-drive attached to the MSX you are emulating. Hard-drive is separated into 2 partitions, **hda-1** and **hda-2**.
… export	Contains a copy of the hard-drive partition, exported from the emulator
fusion-c	Main Fusion-C folder
… examples	Example source code
… header	Header files used by Fusion-C. All function definitons.
… lib	Main dynamic Library
… source	Source files of all Fusion-C function
openMSX	Main folder of openMSX emulator
… catapult	Catapult is a Windows launcher for openMSX. Not used by Fusion-C
... codec	Used by openMSX
… doc	openMSX documention
… hard-drive	Contain the virtual hard-drive for open MSX
… MSX_config	Contains the configuration files of different MSX platfoms
... share	Constinas the systems rom, and configurations used by openMSX
Tools	contains different tools and information for MSX development

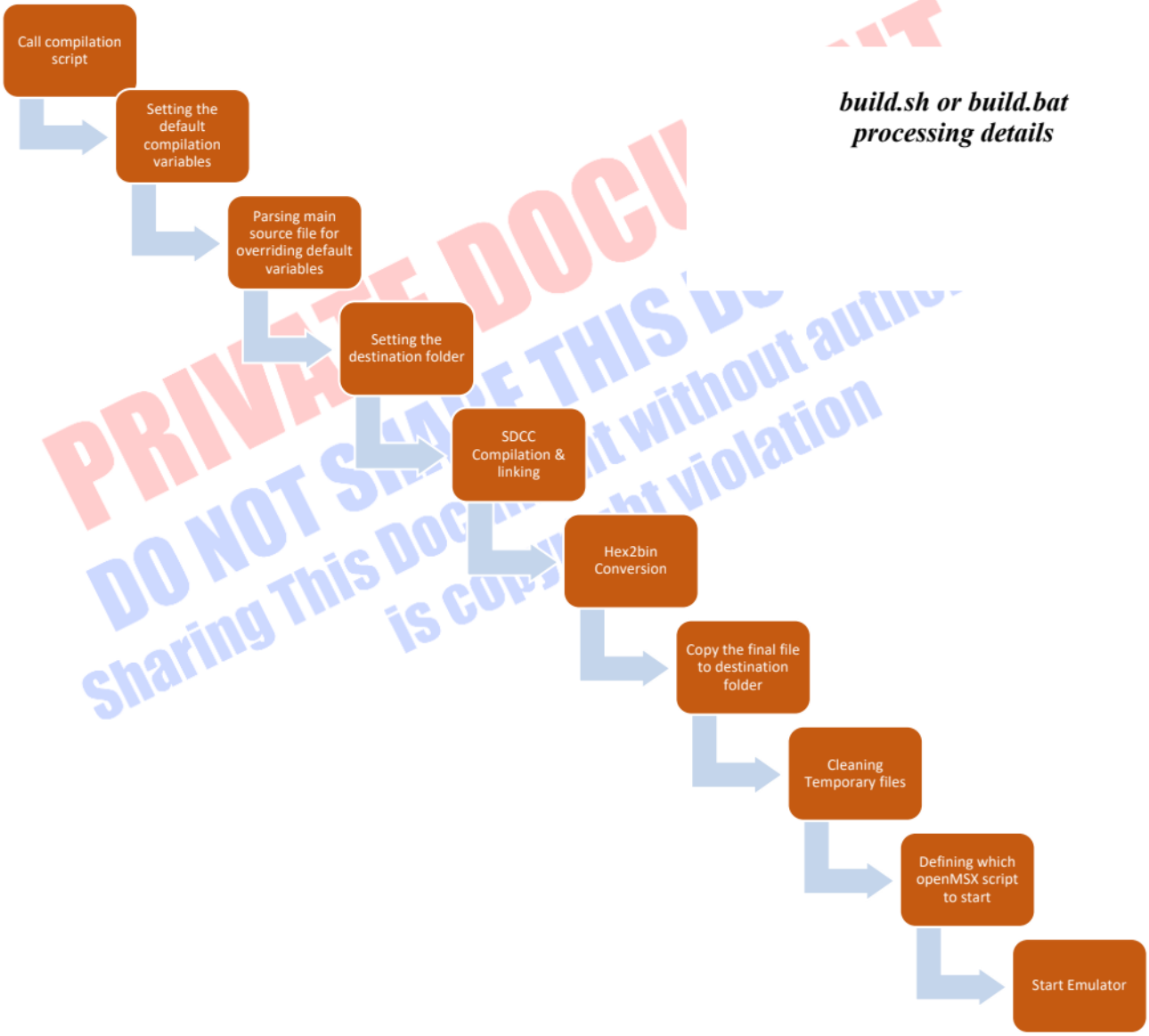
Know the compilation script

The compilation scripts, **build.sh** for Mac OS**,** and **build.bat** for Windows, are designed to automate the process of building and testing your programs. Scripts can be called automatically from your favorite code editor, Sublime text, VS Code… but also manually from a shell / Dos window.
Compilation scripts are in **WorkingFolder/fusion-c/** folder.

Once the script is called it will do the C compilation, the linking, and transform the generated hexadecimal assembler code to a binary executable file compatible with MSX-DOS, the file will be copied inside the destination folder and openMSX will be launched to test your program.

If you choose to use the whole tools chain, and respect all the installation steps, you may not have to change anything inside the compilation script. Anyway, it's a good idea to see what does the script.

build.sh or build.bat processing details



openMSX configuration files

At the end of the compilation process, the script will start **openMSX** emulator to test your program.
⚠ ***Note:*** If an instance of the emulator is already opened, no new instance will be open.

You can choose which version of the MSX computer will be opened, as well as the peripherals that will be associated with it.
There are 7 configurations already defined. Each configuration is described in a text file which can be found here: «**WorkingFolder/openMSX/MSX_config/**».

You can modify these basic configurations yourself by editing these files. Various variables allow you to activate or deactivate the most common MSX options and peripherals. For more details, consult the openMSX documentation.

By default, openMSX will be launched with **script n°2** which corresponds to an **MSX2 Philips NMS 8255** model, with 2 floppy disk drives, 128K of Ram and 128K of Vram. It is extended with an MSX-DOS2 expansion. A joystick is connected to port A (Emulated by the keyboard keys), a mouse is connected to port B, and a Covox module is connected to the printer port.

Note: The configuration files are used to define the behavior of openMSX, as well as the keys to control the emulator.

Config number	Description	Disk Folder
1	**MSX1 Sanyo PHC28L + Floppy Drive.** 64K RAM. 16K VRAM Floppy Disk Drive With MSX-DOS1 Joystick in port A Covox/Simpl Module	emulated disk-drive folder: **WorkingFolder/dsk/dska**
2	**MSX2 Philips NMS 8255 + Floppy Drive** 128K RAM. 128K VRAM Floppy Disk Drive With MSX-DOS2 Joystick in port A Mouse in port B FMPAC + SCC 1024K Ram Expansion Covox/Simpl Module	emulated disk-drive folder: **WorkingFolder/dsk/dska**
3	**MSX2+ Panasonic FS-A1 WSX + Floppy Drive** MSX Music (FM) 64K RAM. 128K VRAM Floppy Disk Drive With MSX-DOS2 Joystick in port A Mouse in port B SCC 1024K Ram Expansion Covox/Simpl Module	emulated disk-drive folder: **WorkingFolder/dsk/dska**
4	**MSX Turbo-R FS-A1 GT + Floppy Drive** MSX Music (FM) 512K RAM. 128K VRAM Floppy Disk Drive With MSX-DOS2 Joystick in port A Mouse in port B SCC Covox/Simpl Module	emulated disk-drive folder: **WorkingFolder/dsk/dska**
5	**MSX2 Philips NMS 8255 + Hard Drive** 128K RAM. 128K VRAM Floppy Disk Drive A Joystick in port A Mouse in port B FMPAC + SCC 1024K Ram Expansion Covox/Simpl Module	emulated Hard-drive folder: **WorkingFolder/dsk/hda-1/**

6	**MSX2+ Panasonic FS-A1 WSX + Hard Drive** MSX Music (FM) 64K RAM. 128K VRAM Floppy Disk Drive With MSX-DOS2 Joystick in port A Mouse in port B SCC 1024K Ram Expansion Covox/Simpl Module	emulated Hard-drive folder: **WorkingFolder/dsk/hda-1/**
7	**MSX Turbo-R FS-A1 GT + Hard Drive** MSX Music (FM) 512K RAM. 128K VRAM Floppy Disk Drive With MSX-DOS2 Joystick in port A Mouse in port B SCC Covox/Simpl Module	emulated Hard-drive folder: **WorkingFolder/dsk/hda-1/**

Example: the configuration used to start a MSX2 emulation **(*emul_start_MSX2_config.txt*)**

SDK Key Maping

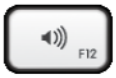
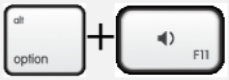
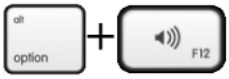
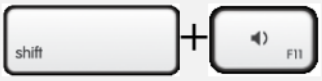
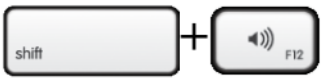
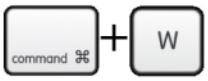
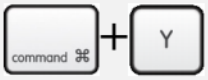
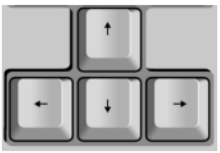
The keys you can use to control the openMSX:

openMSX - MSX KEYBOARD MAPPING

MSX	OSX	Windows
CTRL	(Left) control	(Left) Ctrl
(Dead key. Accent key) / •• \ ^	(Right) control	(Right) Ctrl
GRAPH	(Left) alt option	(Left) Alt
CODE	(Right) alt option	(Right) Alt
SELECT	◂◂ F7	F7
STOP	▸II F8	F8
INS	command ⌘ + |	Insert

openMSX - KEYS CONTROL	OSX	Windows
PAUSE Emulator	command ⌘ + P	Pause Break
QUIT Emulator	command ⌘ + Q	Alt + F4
Save SCREENSHOT	command ⌘ + D	Print Scrn SysRq
Go 1 Sec. back in time *	page up	Page Up
Go 1 Sec. Forward in time *	page down	Page Down
Toggle fastforwad mode	command ⌘ + T	F9
Toggle Console display	command ⌘ + L	F10
Full Screen mode	command ⌘ + F	Alt + Enter
Toggle Audio Mute (Modified from default)	control + command ⌘ + U	Ctrl + ⊞ + U
Quick Load State	command ⌘ + R	Alt + F7
Quick Save State	command ⌘ + S	Alt + F8
COPY from MSX to clipboard	command ⌘ + C	Ctrl + ⊞ + C
PASTE clipboard into MSX	command ⌘ + V	Ctrl + ⊞ + V

* If reverse feature is enabled

openMSX - SDK KEYS CONTROL	OSX	Windows
Cycle from different video sources if exists		
Force or release all inputs grabbed by OpenMSX		
Decrease scale of the emulator window		
Increase scale of the emulator window		
Decrease emulation Speed by 10%		
Increase emulation Speed by 10%		
Import files to MSX virtual Hard-Drive. *		
Export content of MSX Hard-Drive. **		
Reset The Emulated MSX		
Toggle Power ON/OFF		
Type the content of the current autoexec.bat to screen		
Joystick 1		
Joystick 1 - Button A		
Joystick 1 - Button B		

* if MSX Hard-drive is enabled. Copy **dsk/hda-1/*.*** to Virtual HD Drive, HDA partition 1
** if MSX Hard-Drive is enabled. Content of Partitions 1 & 2 is copied to **dsk/export/**

SUBLIME TEXT 3		
	OSX	Windows
Start the compilation		

VS CODE		
	OSX	Windows
Start the compilation		

Know the MSX Floppy Drive and Hard-Drive

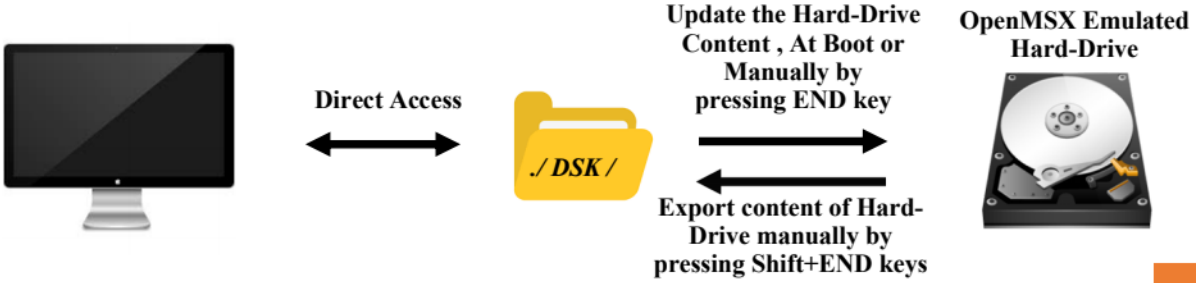
You can put your source codes inside the folder « **WorkingFolder** ». Once you source code is compiled it will be copied to the **./dsk/** folder, which is predefined as the floppy drive **A:** of the emulated MSX.
Inside the **./dsk/** folder you must also have the files that are required to start the MSX-DOS operating system : **COMMAND.COM** and **MSXDOS.SYS** (or COMMAND2.COM and MSXDOS2.SYS).
If your program need other files to work, like images, of any other type files, you can copy them inside the **./dsk/** folder.
Considering this folder acts as a floppy drive, its maximum capacity is 720 KB. Files that exceed the 720KB are ignored.

In case you need much more space for your MSX project, you can use a virtual Hard-Drive for the MSX computer. As you may know, there are several interfaces that allow you to install a hard drive on a real MSX computer. The one we are using by default is the Sunrise IDE interface.
You can activate this Hard-Drive, by modifying the configuration file of the MSX machine you want to use, which is in the folder **./openMSX/MSX_config/**

The Hard-Drive is available for the MSX2, MSX2+, and the MSX Turbo-R.
Edit the configuration file and just change the variable **USE_HARDDRIVE** value from 0 to 1 to activate the Hard-Drive instead of the Floppy Drive. The new configuration will be applied the next time openMSX is launched. You should have something like this:

```
variable  USE_HARDDRIVE 1
variable  USE_DOS2 0
```

The virtual Hard-Drive is physically represented by the file **virtual-HardDrive.dsk** which is in the folder **./openMSX/** . The Hard-Drive capacity is 20 MB, in one and unique partition.
Unfortunately we do not have direct access to the contents of the Hard-Drive, as we have for the floppy drive. This is why we will use in this case the **./dsk/** folder's content to fill the virtual hard disk with the files we need.
Each time the openMSX emulator will be launched with the Hard-Drive enabled, the virtual Hard-Drive will be totally erased, then the content of the **./dsk/** folder will be copied to the Hard-Drive. That way you don't have to worry about anything. The latest versions of your files will always be accessible on the virtual Hard-Drive. The capacity of **./dsk/** folder may not exceed the Hard-Drive capacity (20MB by default).
If you are used to not closing the openMSX window after each test, or each step of your project, you need to update the content of the virtual Hard-Drive manually by pressing the ***END*** key on your keyboard. By pressing ***END***, you launch the procedure for updating the Hard-Drive's content (Erasing Hard-Drive, then copy all files from **./dsk/** to the Hard-Drive

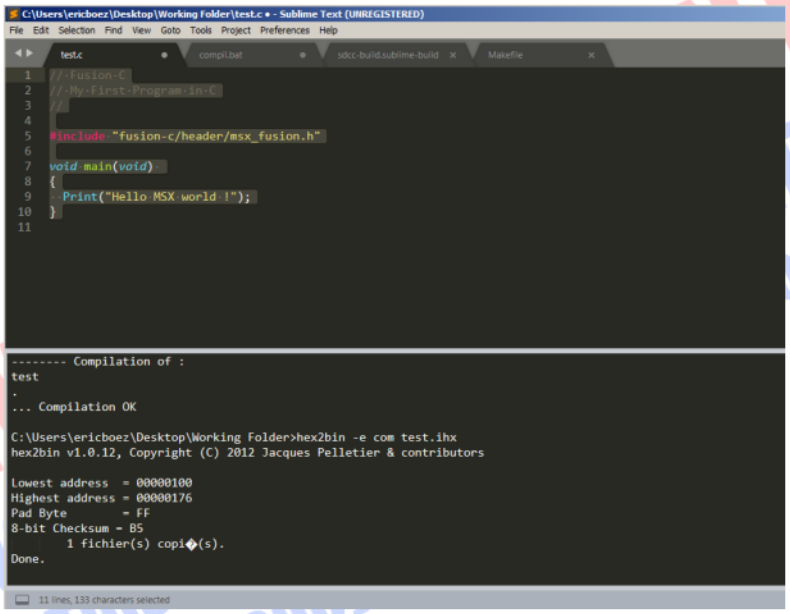


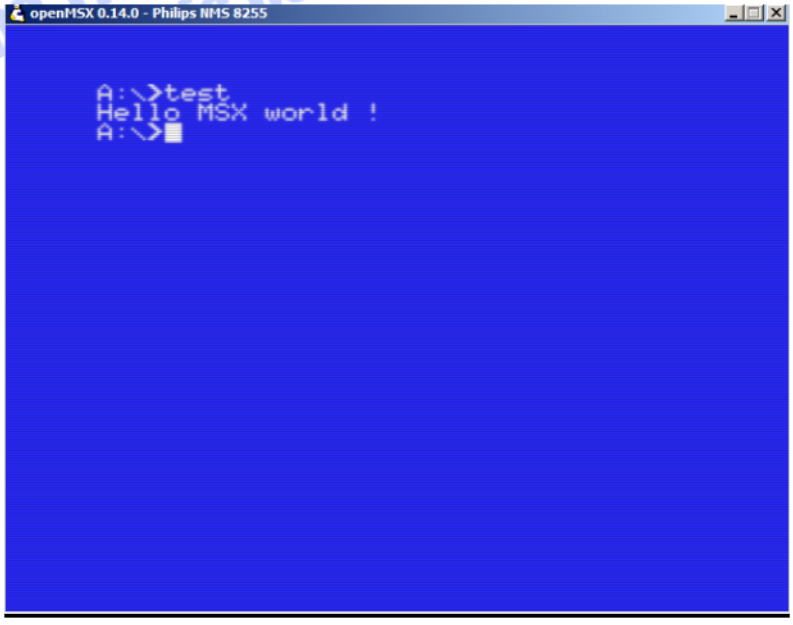
Start your first compilation

Now all your setup must be ready, it's time to start your first compilation for MSX. Open SublimeText, and load **hello.c** from the « **WorkingFolder** ». To launch the compilation…
If you are on MacOS press:

If you are on Windows press:

If all goes well, you must see at the bottom of the Sublime text window that the program is well compiled, and the openMSX must open itself and the **hello.com** program should start automatically





FAQ

Compilation not working with MacOs Catalina, what can I do?

If you are using MacOS Catalina or superior (10.15 +)
Catalina may deny you access to certain tools. In this case, proceed as follows:

With the Finder navigate to the folder **/usr/local/bin/**
(You must have previously activated the display of hidden files mode. See the chapter 'Manual Installation dor MacOS > Note for MacOS Catalina's users)

Sort the files by "**Type**", and select all the "**Unix Executable**":

```
as2gbmap    sdas6808    sdasrab     sdcdb.el    sdldpdk     shc08
makebin     sdas8051    sdasstm8    sdcdbsrc.el sdldstm8    spdk
packihx     sdasgb      sdastlcs90  sdcpp       sdldz80     sstm8
s51         sdaspdk13   sdasz80     sdld        sdnm        stlcs
sdar        sdaspdk14   sdcc        sdld6808    sdobjcopy   sz80
sdas390     sdaspdk15   sdcdb       sdldgb      sdranlib
```

Do a right mouse click on this selection, while holding down the **CTRL** key of the keyboard, then choose **Open** inside the contextual menu. Finder will ask you for confirmation, you must accept.
For some files, an Alert will be displayed "*MacOS cannot verify the developer*", choose the option to open anyway.

Now navigate to « **WorkingFolder/openMSX** », Do a right mouse click on the file « **openMSX.app** », and while holding down the **CTRL** key of the keyboard, choose **Open** inside the contextual menu.

How can I call compilation script manually from anywhere?

With MacOS one way is to add a symbolic link to the compilation script.
Open the Terminal, and type this command:

```
> ln  -s  /Users/<your folder name>/Desktop/WorkingFolder/fusion-c/build.sh
/usr/local/bin/compil
```

This command creates a symbolic link for the compilation script "build.sh" which is located in "WorkingFolder/fusion-c /" to a virtual command which is located in "/usr/local/bin".
This assumes that you have placed the "WorkingFolder" on your desktop. Change the path of the command if necessary.

Now, you can call the compilation manually from the Terminal, by typing:

```
> compil  myprog.c
```

You just have to make the call from the folder where the C source file is.
For example, if the source file is in Desktop/C-source/

```
> CD  Desktop/C-source/
> compil  myprog.c
```

With Windows one way is to add the path to the compilation script in the Windows environment variables.
First open the DOS window prompt as administrator. To do this, click **Start** or press the **Windows key**, type the letters **CMD**, and right-click on the **Command Prompt** entry at the top of the Start menu and choose **Run as administrator** from the context menu, then type this command line inside the DOS's window:

```
> setx PATH "%PATH%;C:\Users\<your folder name>\Desktop\WorkingFolder\fusion-c\"
> shutdown /r
```

This assumes that you have placed the "WorkingFolder" on your desktop. Change the path of the command if necessary. The "Shutdown" will restart your computer, it's necessary to apply the changes.

Now, you can call the compilation's script manually from the DOS prompt, by typing:

```
    > CD desktop\C-source
    > build.bat  myprog.c
```

How can I compile my source to create a ROM?

You must add a directive to the script compiler inside you code.

```
    #define __SDK_ADDRCODE__ ROM
```

This directive will create a ROM with standard 32K ROM parameters. To set other parameters, please check the chapter **"*The compilation directives*"**

Use Microsoft Visual Studio Code

If you are familiar with the **Microsoft VS Code** you will be delighted to be able to use this software to generate code for the MSX. If you are not comfortable with Sublime Text 3, or just want to try an alternative code editor, **VS Code** is a great altenative, it is free and has a version suitable for all 3 major operating systems. I will explain here how to configure **VS Code** to use the Fusion-C.

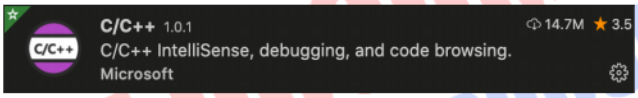
1 – Download VS Code

Go to this page to download the latest version of **VS Code for your operating system**:
https://code.visualstudio.com/download

2 – Install VS Code

MacOS Users, you just have to move the application to your Application's folder, and open the VS Code application. ***Windows's users***, install VS Code as any other windows Application.
Now we will start by installing the C/C++ language support to VS Code.

Click on the "**Extensions**" icons inside the left vertical menu bar, inside the search field, enter the text "C/C++". Choose the **Microsoft C/C++** extension, then install it.



We also need to add Z80 Assembler langage support, so, go back to the search field, and enter the text "Z80". The extension you must install is the **Z80 macro-assembler**.

Now go to the menu **File > Open** then navigate to your user's Desktop and choose to open "WorkingFolder". At this point **VS Code** will create a ***".vscode"*** sub folder inside the main folder.

Click on the "**Explorer**" icon inside the left vertical bar. The content of the "WorkingFolder" must appear as a file list in the explorer's window.

Open the "**hello.c**" source code by clicking on it. From the main menu, choose **Terminal** > **Configure Default Build Task**. A dropdown will appear listing various predefined build tasks for the compilers that VS Code found on your machine. Choose **"open tasks.json file",** replace the existing configuration by this one to teach VS Code how to compile the source code by using our compilation script:

.vscode/task.json

```
{"version": "2.0.0",
"tasks": [
  {
    "type": "shell",
    "label": "Fusion-C SDCC Build",
    "command": "./fusion-c/build.sh",
    "args": [
      "${file}"
    ],
    "options": {
      "cwd": "${workspaceFolder}"
    },
    "group": {
      "kind": "build",
      "isDefault": true
    },
    "presentation": {
      "echo": false,
      "focus": false,
      "reveal": "always",
      "clear": true,
      "showReuseMessage": false,
      "panel": "shared"
    }
   }
  ]
 }
```

.vscode/task.json

```
{"version": "2.0.0",
    "tasks": [
      {
        "type": "shell",
        "label": "Fusion-C SDCC Build",
        "command": "fusion-c\\build.bat",
        "args": [
          "${file}"
        ],
        "options": {
          "cwd": "${workspaceFolder}"
        },
        "group": {
          "kind": "build",
          "isDefault": true
        },
        "presentation": {
          "echo": false,
          "focus": false,
          "reveal": "always",
          "clear": true,
          "showReuseMessage": false,
          "panel": "shared"
        }
      }
   ]
 }
```

Once done, save the **tasks.json** inside the **".vscode/"** sub folder.

VS Code may ask you to configure the "**c_cpp_properties.json**". If not, just create or modify this file inside the **".vscode/"** sub folder.
You must add the path to the SDCC Include directory, and to the Fusion-C Header directory. Here what you must have:

.vscode/c_cpp_properties.json

```
{
    "configurations": [
        {
            "name": "Mac",
            "includePath": [
                "${workspaceFolder}/**",
                "/usr/local/share/sdcc/include/**",
                "${workspaceFolder}/fusion-c/header/**"
            ],
            "defines": [],
            "cStandard": "gnu17",
            "intelliSenseMode": "gcc-x64"
        }
    ],
    "version": 4
}
```

Once done, save the **c_cpp_properties.json** file inside the **".vscode/"** sub folder.

Now comme back to the "**hello.c" tab**. From the main menu choose **Terminal > Run Build Task**
Or press:

.vscode/c_cpp_properties.json

```
{
    "configurations": [
        {
            "name": "Win",
            "includePath": [
                "${workspaceFolder}/**",
                "C:/Program Files/SDCC/include/**",
                "C:/Users/ericboez/Desktop/WorkingFolder/fusion-c/header/**"
            ],
            "defines": [],
            "cStandard": "gnu17",
            "intelliSenseMode": "msvc-x64"
        }
    ],
    "version": 4
}
```

Once done, save the **c_cpp_properties.json** file inside the **".vscode/"** sub folder.

Now comme back to the "**hello.c" tab**. From the main menu choose **Terminal > Run Build Task**
Or press:

The compilation directives

The compilation parameters are defined inside the compilation script (**build.sh** / **build.bat**). Until now, when you wanted to modify these parameters, it was necessary to modify the script. It is now possible to define some parameters directly from the source code of your program.

The **Fusion-C 1.3** compilation script now detect compilatation directives that will override the default parameters. It lets you specify compilation parameters and the testing environnement.
The way to do it is simple, just use the C preprocessor command, ***#define*** with the associated keyword and value.

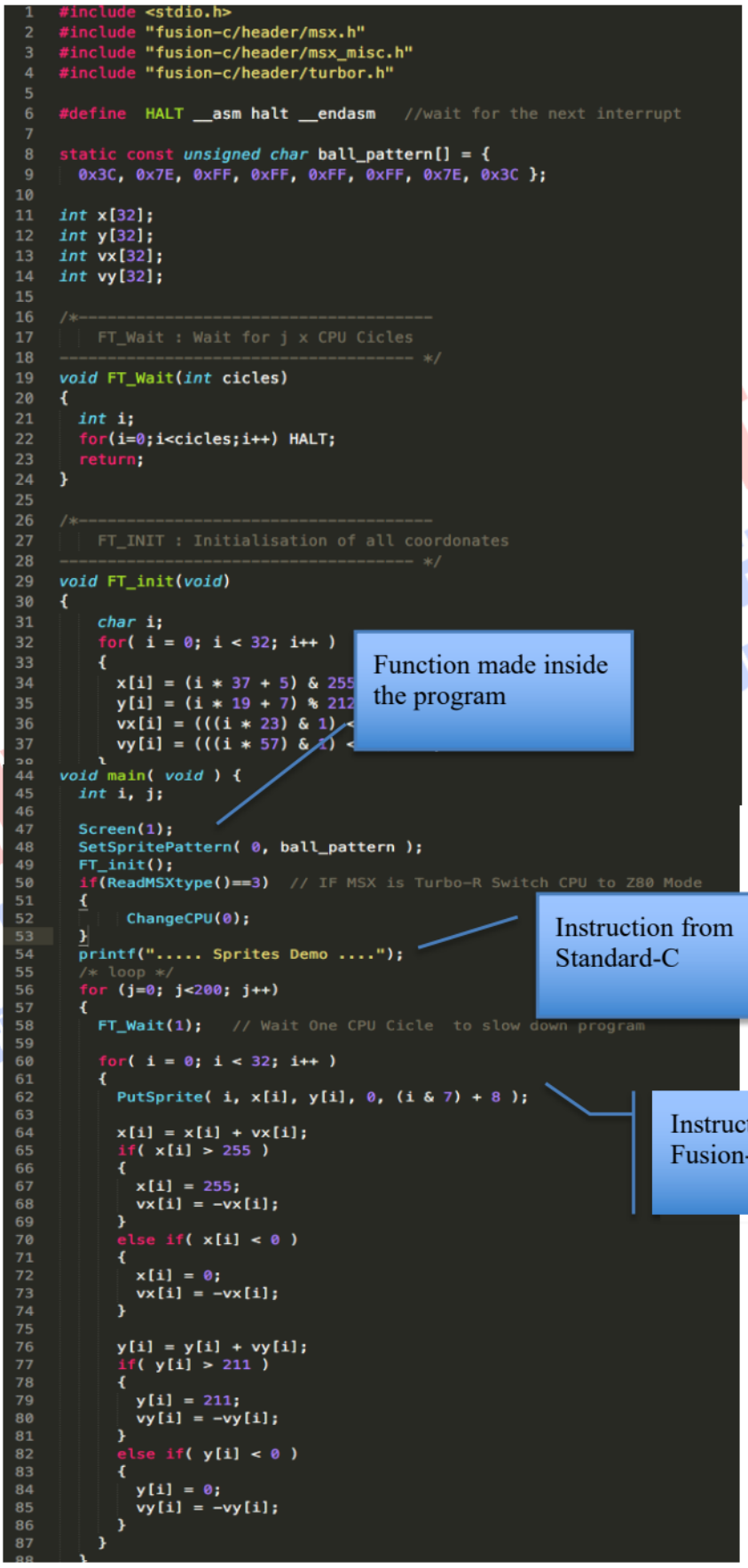
Override parameter	Effect	Default value	Possible values
__SDK_OPTIMIZATION__	Changes the optimization parameter of the compiler.	0	0: Size optimization 1: Speed optimization
__SDK_MSXVERSION__	Which version of MSX to start at the end of compilation porcess.	2	0: Do not start openMSX 1: Start MSX1 with Floppy Drive 2: Start MSX2 with Floppy Drive 3: Start MSX2+ with Floppy Drive 4: Start Turbo-R with Floppy Drive 5: Start MSX2 with Hard-drive 6: Start MSX2+ with Hard-drive 7: Start Turbo-R with Hard-drive
__SDK_ADDRCODE__	Defines the code-loc address of the compiled program.	0x106	Any valid Addresse. Example: must be **0x170** if using **crt0_msxdos_advanced.rel**
__SDK_ADDRDATA__	Defines the data-loc address of the compiled program.	0x00	Any valid Addresse.
__SDK_CRT0__	Defines which **crt0** to use	crt0_msxdos.rel	Any valid **crt0_xxx_xx.rel** file in ***fusion-c/include*** directory.
__SDK_DEST__	Destination folder of the final compiled file	**dsk/dska/** or **dsk/hda-1/**	Any accessible folder.
__SDK_AUTOEXEC__	defines if **the autoexec.bat** file must be written to start the compiled program when starting OpenMSX.	1	0: do not write autoexec.bat 1: write autoexec.bat
__SDK_EXT__	Defines the default file extension of the final compiled program.	com	Any valide extension. For example: rom
__SDK_VERBOSE__	Defines the level of information displayed during compilation	2	0: Minimal information displayed 1: Medium information displayed 2: Full information displayed
__SDK_MSXDOS__	Defines the MSX-DOS version to use when booting openMSX. This parameter forces the compilation script to copy the defined MSX-DOS files to the destination folder.	0	0: Do not change files 1: Use MSXDOS 1 files 2: Use MSXDOS 2 files
__SDK_ROMSIZE__	Defines the final ROM size. The final file could be truncated if the final file is too big.	Not set	8000: For a 32K ROM C000: For a 48K ROM

Example of use

```
// Includes
#include "msx_fusion.h"
#include "vdp_graph2.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdio.h>

// Compilation & SDK  directives
#define  __SDK_OPTIMIZATION__ 1
#define  __SDK_MSXVERSION__ 5
#define  __SDK_AUTOEXEC__ 1
```

Note: To be sure the directive will be correctly analysed, respect the case, and number of spaces between words.
***Put the directive as comment with "//" or " */.. /*" will not disable them**. You must remove the text to disable compilation directives; or disable the detection of the directives inside the compilation script.*
*If you want to disable the detection of the compilation directives commands, you must edit the file 'build.sh' / 'build.bat' and change the value of the **CHECK_OVERRIDE** variable to **0***

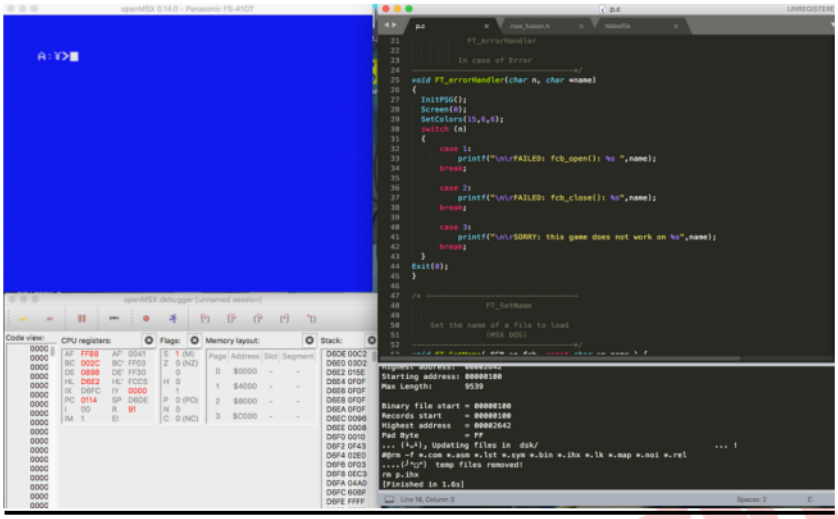
Example of a C program




```c
#include <stdio.h>
#include "fusion-c/header/msx.h"
#include "fusion-c/header/msx_misc.h"
#include "fusion-c/header/turbor.h"

#define  HALT __asm halt __endasm   //wait for the next interrupt

static const unsigned char ball_pattern[] = {
  0x3C, 0x7E, 0xFF, 0xFF, 0xFF, 0xFF, 0x7E, 0x3C };

int x[32];
int y[32];
int vx[32];
int vy[32];

/*-----------------------------------------
    FT_Wait : Wait for j x CPU Cicles
-----------------------------------------*/
void FT_Wait(int cicles)
{
  int i;
  for(i=0;i<cicles;i++) HALT;
  return;
}

/*-----------------------------------------
    FT_INIT : Initialisation of all coordonates
-----------------------------------------*/
void FT_init(void)
{
    char i;
    for( i = 0; i < 32; i++ )
    {
      x[i] = (i * 37 + 5) & 255
      y[i] = (i * 19 + 7) % 212
      vx[i] = (((i * 23) & 1) <
      vy[i] = (((i * 57) & 1) <
    }
void main( void ) {
  int i, j;

  Screen(1);
  SetSpritePattern( 0, ball_pattern );
  FT_init();
  if(ReadMSXtype()==3)  // IF MSX is Turbo-R Switch CPU to Z80 Mode
  {
      ChangeCPU(0);
  }
  printf("..... Sprites Demo ....");
  /* loop */
  for (j=0; j<200; j++)
  {
    FT_Wait(1);   // Wait One CPU Cicle  to slow down program

    for( i = 0; i < 32; i++ )
    {
      PutSprite( i, x[i], y[i], 0, (i & 7) + 8 );

      x[i] = x[i] + vx[i];
      if( x[i] > 255 )
      {
        x[i] = 255;
        vx[i] = -vx[i];
      }
      else if( x[i] < 0 )
      {
        x[i] = 0;
        vx[i] = -vx[i];
      }

      y[i] = y[i] + vy[i];
      if( y[i] > 211 )
      {
        y[i] = 211;
        vy[i] = -vy[i];
      }
      else if( y[i] < 0 )
      {
        y[i] = 0;
        vy[i] = -vy[i];
      }
    }
  }
```

The first four lines of this program are the « includes » that allow the compiler to know in which library to find the functions of your program. Note that the definition files which compose your own library must be included with a full path to them.
At line 6 you can see a Define. The Define is some kind of « Search and Replace » routine made by the pre-compiler. It will replace the first part of the sentence by the second part. In our example, « HALT » will be replaced by « __asm halt __endasm » in the source code, before compilation.
From lines 8 to 14, there are global variable definitions. The global variables are available from any function of the program, without the need to pass them thru function variables.
At line 19. There is our first function called « FT_Wait ». It does not return anything (void), and to be use you must enter a variable named « cycles » which is an « int » (Number).
From line 44 to the end, there is the « Main » function. The « main » is the part of the program which will be executed at first. This an indispensable part of any C program.
Note that the way the functions are named. All standard C functions are in lower case. Check line 54: printf.
All functions coming from the FUSION-C Library always begins with an upper case letter, and has an upper-case letter at the beginning of each word that compose the function name. Check line 52 « ChangeCpu », is a function defined in « turbor.h »
As a good practice, we encourage you to name your own functions with a distinctive character. Check lines 29, and 19. The functions that compose the program itself always start with « FT_ ».
Thus, when you read the code listing, you can know where come from the function, it's easier to debug your program.

Example of the working environment



On the upper left of the screen, the OpenMSX Emulator's window which start the compiled program.
On the lower left, the OpenMSX debugger, just in case ...
On the right side, The Sublime text window. You can notice on the lower side of this window, the compilation information provided by SDCC and Hex2Bin once the program is compiled.
Of course you can manage windows in the way you want.

Content of the FUSION-C SDK

The Library

Composed of ***297*** functions dedicated to the MSX. We will describe each of 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	:	Primary and essential 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_paint.h	:	MSX2 Fast painting function
vdp_sprites.h	:	Sprite related functions

Optional header files

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

g9klib.h	:	V9990 GFX9000 Support functions (Beta version)
macro.inc	:	part of the g9klib.h
gr8net-tcpip.h	:	TCPIP Functions for the Gr8NET cartridge (Beta version)

The tools

to help you and facilitate the development of games for MSX are provided in the package.

Image To Sprite Editor	:	Tool to transform black and white image icons 16x16 sprites.
Sprite Path Editor	:	Tool to draw paths or routes, retrieve the coordinatesof each point.
RLEWB Compressor	:	A command line tool to compress data into the RLEWB format.
SC2 GraphX Conv	:	A command line tool to transfom bitmapped image to the MSX Screen 2 format, with a very good algorithm.

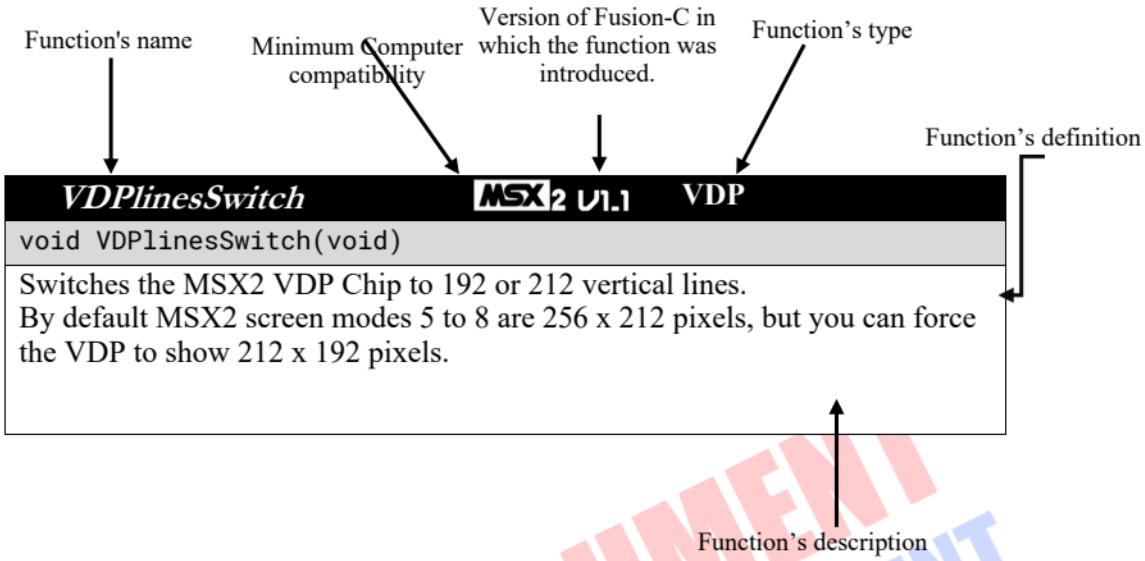
Other tools from other developpers:

AYFX Edit	:	A Windows Only editor to edit AYFX sound effects.
Bin2Hex	:	A command line tool to transform data into Hexadecimal array.
BitBuster	:	Tools for compression (Windows) decompression (MSX) datas.
Disk Image Managers	:	Three tools to edit and manage disk images.
Disk2Rom	:	Tool to transform a 720K Disk Image into a ROM file
DskTool	:	A command line editor to manage disk images
MSX Gaphic Palette	:	MSX Graphic palette information.
MSX Image Viewer	:	A Windows to show image save in MSX format.
nMSXTiles	:	A tool to draw and create tiles.
sjasm42c	:	A Z80 assembler compiler.
SpriteSX	:	A Windows tool to create sprites.
Vortex Tracker	:	A Windows tool to create music in PT3 format


HiT BiT
PRINTER
FDD
F2 F7
F3 F8
F4 F9
F5
R
E
Z
A
S
D
Q
X

Functions List

Note about function's description



	No icon means the function works on all MSX computers from the first **MSX1** to the **MSX Turbo-r**
MSX2	This icon indicates the function can only be used with **MSX2** computers and upper.
MSX2+	This icon indicates the function can only be used with **MSX2+** computers and upper.
MSX turbo R	This icon indicates the function can only be used with **MSX TURBO-R**.
DOS	This icon indicates the function works with **MSX-DOS** & **MSX-DOS2**
DOS2	This icon indicates the function works with **MSX-DOS2** only
ROM	This icon indicates the function is dedicated for ROM programming only
V1.1	This icon indicates the function has been introduced or corrected from version **1.1** of FUSION-C
V1.2	This icon indicates the function has been introduced or corrected from version **1.2** of FUSION-C
V1.3	This icon indicates the function has been introduced or corrected from version **1.3** of FUSION-C
	This icon indicates an important change since FUSION-C's previous version, which may require adaptation on your part by updating or re-installation...

MSX FUSION

[msx_fusion.h]

This is the main part of the library. You should always include msx_fusion.h in your programs, it provides all basic components for your projects.

Console Functions

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.		

Getche	*CONSOLE*
`char` **`Getche`** `(void)`	
Reads and display character from console. Returns the read character.	

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

InputString	*CONSOLE*
`char` **`InputString`** `(char *Dest, unsigned 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	

Locate	*CONSOLE*
`void` **`Locate`** `(unsigned int x, unsigned int y)`	
Sets console cursor to ***X & Y*** coordinates.	

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

PutCharHex v1.2 *CONSOLE*

```
void    PutCharHex (char num)
```

Prints the hexadecimal representation of the char ***num***, on the text screen mode.

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 the previous line, stop at top left of the screen.
\t (0x09) - Horizontal Tab. Tab, overwrites with spaces up to 8th next 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 must be the ascii number of the char to use 32=' ', 48='0', etc..
length must bebetween 1 to 5

PrintChar *CONSOLE*

```
void    PrintChar (char car)
```

Prints the character ***car*** to console (or to a text screen mode)

PrintDec *CONSOLE*

```
void    PrintDec (unsigned int num)
```

Prints signed integer ***num*** from -32768 to 32767 to a text screen mode

printf	***CONSOLE***

```
unsigned 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
%u	unsigned int
%x	hexadecimal int
%c	character
%s	string
%%	a % character
%l	signed long
%ul	unsigned long
%lx	hexadecimal long

Miscellaneous Functions

Cls — *MISCELLANEOUS*

```
void    Cls (void)
```

Clears console or any screen mode

KeySound — *MISCELLANEOUS*

```
void    KeySound (char n)
```

Enables or disables Key Sound. The ***n*** value must be:

0 To Disable Key Sound
1 To Enable Key Sound

FunctionKeys — *MISCELLANEOUS*

```
void    FunctionKeys (char n)
```

Shows or hides Function Keys on a Basic text screen mode. The ***n*** value must be:

0 To Disable Function keys
1 To Enable Function keys

ChangeCap — *MISCELLANEOUS*

```
void    ChangeCap (char n)
```

Changes the state of the Cap Led. The ***n*** value must be:

0 To Disable Cap Led
1 To Enable Cap Led

ReadMSXtype — *MISCELLANEOUS*

```
char    ReadMSXtype (void)
```

Reads and returns the MSX type. The returned values can be:

0 MSX 1
1 MSX 2
2 MSX2+
3 MSX Turbo-r

ReadKeyboardType	V1.3	*MISCELLANEOUS*

```
char    ReadKeyboardType (void)
```

Reads and returns the MSX keyboard type. The returned values can be:

0	Japanese
1	International
2	French
3	UK
4	German

Screen	*MISCELLANEOUS*

```
void    Screen (char mode)
```

Sets display to specified screen mode.
mode can be a valid screen mode number, between **0** and **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 increased by 1 on each VDP complete screen draw. The refresh rate depends of the MSX computer, it can be 50 Hz on european's computers (PAL), or 60 Hz on Japanese computers (NTSC).
On MSX2, 2+ and MSX-Turbo-r the screen's refresh rate can be adjusted manually with ***VDP50Hz()*** and ***VDP60Hz()*** 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**.

CovoxPlayVram	V1.1 MSX2	*MISCELLANEOUS*

```
void    CovoxPlayVram (char Page, unsigned int StartAddress, unsigned int
Length, char Speed)
```

Plays 8bits PCM audio 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.

Page	is the VRAM page where the sample is stored
StartAddress	represents the first VRAM address of the Sample
Length	is the length of byte you want to play
Speed	is the playing speed. More this number is high, more the playing is slow.

CovoxPlayRam	V1.3	*MISCELLANEOUS*

```
void    CovoxPlayRam (void StartAddress, unsigned int Length, char Speed)
```

Plays 8bits PCM audio stored in RAM thru Covox/simpl module. The PCM audio sample must be stored in the MSX RAM.

StartAddress	is the address in RAM where the sample is stored
Length	is the length of byte you want to play
Speed	is 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.

PatternRotation	V1.3	*MISCELLANEOUS*

```
void    PatternRotation (unsigned int *Pattern, unsigned int *buffer, char
rotation)
```

Rotates the 8x8 pixels sprite pattern. ***Pattern*** is the address of the 8x8 pixels source pattern, ***buffer*** is the address of a 8 bytes buffer where the rotated pattern will be stored.

Rotation must be

0	for a 90° right rotation
1	for a 90° left rotation

PatternHFlip	V1.3	*MISCELLANEOUS*

```
void    PatternHFlip (unsigned int *Pattern, unsigned int *buffer)
```

Flips horizontally an 8x8 pixels pattern. ***Pattern*** is the address of the 8x8 pixels source pattern, ***buffer*** is the address of an 8 bytes buffer where the flipped pattern will be stored.

PatternVFlip	**V1.3**	***MISCELLANEOUS***

```
void    PatternVFlip (unsigned int *Pattern, unsigned int *buffer)
```

Flips vertically a 8x8 pixels pattern. ****Pattern*** is the address of the 8x8 pixels source pattern, ****buffer*** is the address of an 8 bytes buffer where the flipped pattern will be stored.

TurboMode	**V1.3**	***MISCELLANEOUS***

```
void    TurboMode (char mode)
```

Enables the Turbo Mode of the MSX2+ Panasonic FS-A1WSX, FS-A1WX, FS-A1FX.
The turbo mode is the possibility to run the Z80 at 5.37 Mhz instead of 3.57Mhz
mode must be:

1 to activate the turbo mode.
0 to deactivate the turbo mode

Exit	***MISCELLANEOUS***

```
void    Exit (char Num)
```

Exits from C program, and go back to MSX-DOS. ***Num*** represents a MSX DOS error number you want to show when exiting. **0** means no error.
In case of a ROM compilation, this function is reseting the MSX, in this case ***Num*** have no effect.

Joystick & mouse functions

JoystickRead — *JOYSTICK*

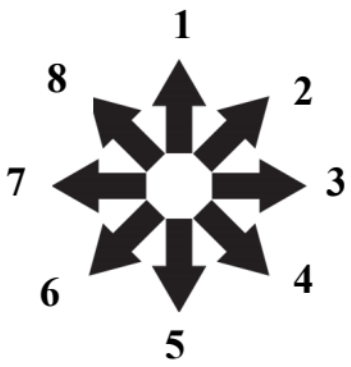
```
char    JoystickRead (char JoyNumber)
```

Reads and returns the state (Direction) of a joystick.

JoyNumber is the joystick port you want to read, it can be:

0 Keyboard's Arrow keys.
1 Joystick port 1
2 Joystick port 2

Returned values:



0=inactive

TriggerRead — *JOYSTICK*

```
char    TriggerRead (char TriggerNumber)
```

Reads and returns state of a joystick button.

TriggerNumber is the button you want to read, it 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 The button is inactive,
255 The button is pressed

JoystickReadTo	V1.3	*JOYSTICK*

`void` **`JoystickReadTo`** `(JOY_DATA *jd)`

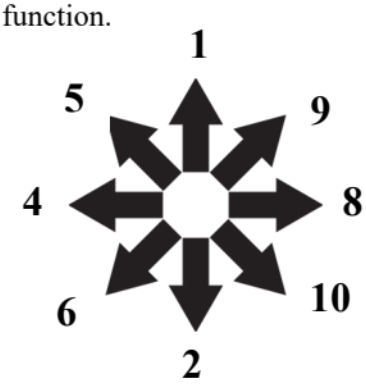
Reads and returns state of a joystick in a faster way than ***JoystickRead.***
This function directly queries the PSG port which controls the joysticks, also it checks Fire Buttons and direction in the same time.
The function is using this pre-defined structure to return the read values.

```
typedef struct {
    char   joyport;
    char   up;
    char   down;
    char   left;
    char   right;
    char   button1;
    char   button2;
    char   global;
} JOY_DATA;
```

You must declare the structure before using this function, like this:
static JOY_DATA jd;
Returned values goes to the structure variables. According to the previous structure declaration, you will receive datas inside

jd.up	1 if joystick goes up
jd.down	1 if joystick goes down
jd.left	1 if joystick goes left
jd.right	1 if joystick goes right
jd.button1	1 if button is pressed
jd.button2	1 if button is pressed
jd.global	the global value see below

Note, ***jd.joyport*** is an input variable. It says to the routine which joystick port to read. The value must be set before calling the function.
jd.joyport must be **1** if you want to read joystick port n°1, and **2** if you want to read joystick port n°2.
There are two ways to use the function inside your own program. You can choose to use the ***left, right, up, down*** variables, or use the global value returned in ***jd.global***. In this last case, it's most like the ***JoystickRead*** function.
jd.global returned values are:



0 = inactive
Note, you can also check impossible moves, like UP + DOWN or LEFT + RIGHT etc. By analyzing the binary value of ***jd.global***

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	RIGHT	LEFT	DOWN	UP

MouseRead v1.1 *MOUSE*

```
unsigned int   MouseRead (unsigned 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 16-bit value.
Decoding this value must be done like this:

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

MousePort is referring to joystick port 1 or port 2, it is defined as a predefined macro, 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**, this means no mouse is connected.

MouseReadTo v1.2 *MOUSE*

```
void    MouseReadTo (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 to return the read values.

```
typedef struct {
   signed char   dx;
   signed char   dy;
   char     lbutton;
   char     rbutton;
} MOUSE_DATA;
```

You must declare this structure before using this function, like this :

```
static MOUSE_DATA md;
```

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 datas inside

- ***md.dx***
- ***md.dy***
- ***md.lbutton***
- ***md.rbutton***

lbutton and rbutton are returning 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 returned is by default:
1 = not pressed.
0 = pressed

International Key Matrix

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

This array shows the international key matrix. Please check the ***MSX Keyboard*** chapter to know more about other key matrix.
Code example: if you want to check the state of the ESC key

```
Esc_state = (GetKeyMatrix(7) >>2) & 1;
```

Inkey — *KEYBOARD*

```
char    Inkey (void)
```

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

KillKeyBuffer — *KEYBOARD*

```
Void    KillKeyBuffer (void)
```

Clears the key buffer.

WaitKey — *KEYBOARD*

```
Char    WaitKey (void)
```

Waits for a key pressed. Returns the ASCII code of the key.

Rkeys V1.3 *KEYBOARD*

```
char    Rkeys (void)
```

Returns the state of line 8 of the Key matrix into a byte. **(Arrows keys support.)** If a bit is active (1) this means the associated key is pressed. (It can report 2 keys pressed at the same time.)

bit	7	6	5	4	3	2	1	0
Key	RIGHT	DOWN	UP	LEFT	DEL	INS	HOME	SPACE

The easy way to check one key pressed is to refer to the returned value.

```
 1        SPACE is pressed
 2        HOME is pressed
 4        INS is pressed
 8        DEL is pressed
16        LEFT is pressed
32        UP is pressed
64        DOWN is pressed
128       RIGHT is pressed
```

Note : this example method cannot be used to check if several keys are pressed at the same time, to do that, analyse the returned byte value

Fkeys V1.3 *KEYBOARD*

```
char    Fkeys (void)
```

Returns the state of line 8 of the Key matrix into a byte. **(Function keys support.)** If a bit is active (1) the associated key is pressed. It can report 2 keys pressed at the same time.

bit	7	6	5	4	3	2	1	0
Key	STOP	GRAPH	ESC	F5	F4	F3	F2	F1

The easy way to check one key pressed is to refer to the returned value.

```
 1        F1 is pressed
 2        F2 is pressed
 4        F3 is pressed
 8        F4 is pressed
16        F5 is pressed
32        ESC is pressed
64        GRAPH is pressed
128       STOP is pressed
```

Note: this method cannot be used to check if several keys are pressed at the same time, to do that, analyse the returned byte value

I/O Port Functions

OutPort	***I/O PORT***
`void  OutPort (char port, char data)`	
Sends an 8-bit ***data*** value to a MSX ***port***	

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

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

VDP Functions

VDPstatus — *VDP*

```
char    VDPstatus (char vdpreg)
```

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

VDPstatusNi — *VDP*

```
char    VDPstatusNi (char vdpreg)
```

Reads VDP Status register ***vdpreg*** and returns the value. The Interrupt is not modified by this function.

VDPwriteNi — *VDP*

```
void  VDPwriteNi (char vdpreg, char data)
```

Writes ***data*** to the VDP register ***vdpreg***. The Interrupt is not modified by this function.

VDPwrite — *VDP*

```
void    DPwrite (char vdpreg, char data)
```

Writes **data** to the VDP register ***vdpreg***. The Interrupt is disabled before writing, and enabled after writing.

IsVsync — *VDP*

```
char    IsVsync (void)
```

Check the VDP and returns the Vblank state.
Return **1**, if true.

IsHsync — *VDP*

```
char    IsHsync (void)
```

Checks the VDP and returns HSynch state.
Return **1** if true.

Vsynch — v1.3 — *VDP*

```
char    Vsynch (void)
```

Same effect as the ***IsVync()*** function, but is much faster because it does not read the state from the VDP register. The function returns **0** while VDP raster is sending the image to the screen.

Vpeek — *VDP*

```
char    Vpeek (int address)
```

Reads and returns a byte from Vram Memory ***address***

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

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

VpeekNext	*VDP*
`char` **`VpeekNext`** `(void)`	
Reads and returns a byte from the Vram Memory address set by ***VpeekFirst***, and increments address for the 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`** `(char` **`ForeCol`**`, char` **`BackgrCol`**`, char` **`BorderCol`**`)`	
Sets, foreground ***ForeCol*** color, background ***BackgrCol*** color and border color ***BorderCol***. Supports MSX2's screen mode like Screen8, where colors can be a number between 0 and 255.	

SetColor	V1.3	*VDP*
`void` **`SetColor`** `(char` **`ForeCol`**`)`		
Sets only the foreground with ***ForeCol*** color. Useful for function that do not have a color parameter, like ***PutText***.		

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

SetColorPalette V1.2 MSX2 *VDP*

```
void   SetColorPalette (char Color, char Red, char Green, char Blue)
```

Sets a color with new RGB parameters.
Color: is the color number you want to modify (Between 1 and 15)
Note: The color 0 cannot be modified
Red, ***Green***, and ***Blue*** parameters are levels of Red, Green and Blue you want to assign to this C***olor***. Each parameter is a number between **0** and **7**
Example:

```
SetColorPalette (15, 7, 0, 0);
```

This sets the color number 15 to a pure red color.

SetPalette MSX2 *VDP*

```
void   SetPalette ((Palette *) mypalette)
```

Sets the screen color palette with the new « ***mypalette*** » structure data.
The predefined « Palette Structure » is composed of 16 lines of 4 values: ***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};
SetPalette((Palette *) mypalette);
```

SetTransparent V1.3 MSX2 *VDP*

```
void   SetTransparent (char n)
```

By default, color number 0 is used as a transparent color (n=0). This function disables the transparent mode, and give you the possibility to redefine this color number 0 in the same way as other colors of the palette, for example with the SetColorPalette function.
The parameter ***n***, can be **0** to enable transparency color mode, or **1** to disable the transparency color mode.
example:

```
SetTransparent (1);             // Disable color #0 transparent mode
SetColorPalette (0, 2,6,7);     // Redefine color #0 with new RGB Values
```

RestorePalette MSX2 *VDP*

```
void    RestorePalette (void)
```

Restore the MSX default Color Palette.
MSX Default color palette.

	Name	R G B
00	TRANSPARENT	0. 0. 0.
01	BLACK	0. 0. 0.
02	MEDIUM_GREEN	1. 6. 1.
03	LIGHT_GREEN	3. 7. 3.
04	DARK_BLUE	1. 1. 7.
05	LIGHT_BLUE	2. 3. 7.
06	DARK_RED	5. 1. 1.
07	CYAN	2. 6. 7.
08	MEDIUM_RED	7. 1. 1.
09	LIGHT_RED	7. 3. 3.
10	DARK_YELLOW	6. 6. 1.
11	LIGHT_YELLOW	6. 6. 4.
12	DARK_GREEN	1. 4. 1.
13	MAGENTA	6. 2. 5.
14	GRAY	5. 5. 5.
15	WHITE	7. 7. 7.

SetDisplayPage MSX2 *VDP*

```
void    SetDisplayPage (char page)
```

Sets the VRAM ***page*** displayed to the screen.
Parameter ***page*** can be 0, 1, 2, 3;
(it depends of the screen mode, see number of pages available for each screen mode)

SetActivePage MSX2 *VDP*

```
void    SetActivePage (char page)
```

Sets the VRAM ***page*** of the that receive modifications and instructions.
Parameter ***page*** can be 0, 1, 2, 3 ; it depends of the screen mode.

SetScrollH MSX2+ *VDP*

```
void    SetScrollH (int n)
```

Uses Hardware **horizontal** screen scrolling of the MSX2+ and MSX Turbo-R. ***n*** is the number of pixels you want to scroll to, it can be positive or negative.
With screen modes 6 and 7, ***n*** must be a multiple of 2.

SetScrollV MSX2 *VDP*

```
void    SetScrollV (char n)
```

Uses Hardware **vertical** screen scrolling of the MSX2, MSX2+ and MSX Turbo-R.
n is the number of pixels you want to scroll to, it can be positive or negative.
With screen modes 6 and 7, ***n*** must be a multiple of 2

SetScrollMask v1.3 MSX2+ *VDP*

```
void    SetScrollMask (char n)
```

When using the MSX2+ / Turbo-R hardware horizontal scrolling capability, you can enable or disable a 8-pixel column's mask at the left of the screen.
Whith this mask enable the scrolling appears much more fluent.
The parameter ***n***, can be **1** to enable the mask or **0** to disable the mask.

SetScrollDouble v1.3 MSX2+ *VDP*

```
void    SetScrollDouble (char n)
```

When using the MSX2+ / Turbo-R hardware horizontal scrolling capability, you have the possibility to use the 2 screen's pages as one and only horizontal screen. This means the second VRAM page is placed at the right of the first page, thus you have a 512 x 256 pixels image to scroll on the horizontal axis.
The parameter ***n***, can be **1** to enable the double screen mode or **0** to disable this mode.

HideDisplay *VDP*

```
void    HideDisplay (void)
```

Hides the screen display (Black screen). This command disables the on-screen display, but graphics commands continue to apply.
Disabling the display allows faster graphics controls, for example to create a game level display.

ShowDisplay *VDP*

```
Void    ShowDisplay (void)
```

Shows the screen display if it was previously hidden.

FillVram *VDP*

```
void    FillVram (unsigned int Startaddress, char value, unsigned int
Length)
```

Fills the Vram from ***Startaddress*** with ***Length*** bytes. The **Startaddress** is limited to a 16 bits address (from 0x0 to 0xFFFF), thus is you need to fill the MSX2's VRAM over 0xFFFF use the **SetActivePage** before.

PutText v1.1 *VDP*

```
void    PutText (unsigned int X, unsigned int Y, char *str, char LogOp)
```

Prints the text string ****str*** to graphic screen (modes 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)
The ***Y*** coordinate is limited to an 18-bit number (from 0x0 to 0xFF), thus is you need to put text over 255 (On another VRAM page) use the **SetActivePage** before.

VDP50Hz MSX2 *VDP*

```
void    VDP50Hz (void)
```

Switches the MSX2 VDP to 50 Hz Pal Mode

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

VDPlineSwitch	V1.1 MSX2	*VDP*
`void VDPlineSwitch (void)`		
Switches the MSX2 VDP Chip to 192 or 212 vertical lines. By default, MSX2 screen modes 5 to 8 have 212 lines heigh, this function force the VDP to 192 lines.		

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. *Note*: The ***DestVramAddress*** is limited from 0 to 65535. To send datas over this limit use the **SetActivePage** function before.		

CopyRamToVram2	V1.3 MSX2	*VDP*
`void CopyRamToVram2 (void *SrcRamAddress, unsigned int DestVramAddress, unsigned int Length)`		
Exactly the same as previous function, but this one is optimsed to be faster on MSX2 and upper computers (Do not use this function on MSX1).		

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*
`char GetVramSize (void)`		
Returns the Vram size of the MSX Computer. Returned values can be: **16**, **64**, **128**.		

SetVDPwrite	V1.2	*VDP*
`void SetVDPwrite (unsigned int Address)`		
Sets the MSX VDP in writing mode. Function is MSX1 and MSX2 VDP compliant. This command is for experts & specific use. You don't need it until you want to write your own graphic routines.		

SetVDPread	v1.2	*VDP*

```
void    SetVDPread (unsigned int StartAddress)
```

Sets the MSX VDP for reading process at ***StartAddress***. MSX1 and MSX2 VDP compliant. You may need this function if you want to write your own graphic routines.

SetExpandVDPcmd	v1.3 MSX2+	*VDP*

```
void    SetExpandVDPcmd (char n)
```

This function enables the capability of the MSX2+ VDP to use the VDP commands with MSX1's screen modes **0** to **4.** *(See all VDP commands in the dedicated chapter.)*
The parameter ***n***, can be **1** to enable or **0** to disable this capability.

SetScreen10	v1.3 MSX2+	*VDP*

```
void    SetScreen10 (char n)
```

When using the MSX2+ / Turbo-R the Screen 10 mode can show up to 12499 colors on the screen. This screen mode is the same as the screen mode 8 but with a different way to encode colors. Thus to use this mode activate first screen mode 8 with the instruction : ***Screen(8),*** then, activate the screen 10 mode.
The parameter ***n***, can be **1** to enable the screen mode or **0** to disable this mode.
example :

```
Screen(8);          // USe Screen mode 8 as basis
SetScreen10 (1);    // Now activating Screen mode 10
```

SetScreen12	v1.3 MSX2+	*VDP*

```
void    SetScreen12 (char n)
```

When using the MSX2+ / Turbo-R the Screen 10 mode can show up to 19268 colors on the screen. This screen mode is the same as the screen mode 8 but with a different way to encode colors. Thus to use this mode activate first screen mode 8 with the instruction : Screen(8), then, activate the screen 12 mode.
The parameter ***n***, can be **1** to enable the screen mode or **0** to disable this mode.
example :

```
Screen(8);          // Use Screen mode 8 as basis
SetScreen12 (1);    // Now activating Screen mode 12
```

SetAdjust	v1.3 MSX2	*VDP*

```
void    SetAdjust (signed char x, signed char y)
```

Adjusts the MSX's screen to center the output image.
x is the screen horizontal offset, it can vary from -8 to +7
y is the screen vertical offset, it can vary from -8 to +7

SaveScreenBoot V1.3 MSX2 *VDP*

```
void    SaveScreenBoot (char Nb)
```

Used to store a series of parameters which will be used each time the MSX is started.

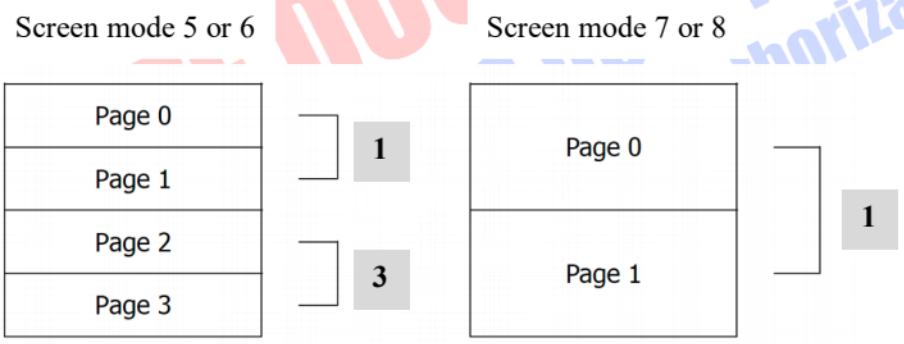
1	X Screen Offset, set with ***SetAdjust***
2	Y Screen Offset, set with ***SetAdjust***
3	Screen Foreground Color, set with ***SetColors*** or ***SetColor***
4	Screen Background Color, set with ***SetColors***
5	Screen Border Color, set with ***Setcolors*** or ***BoderColor***
6	Default Boot Screen mode, set with ***Screen***. Can be Screen mode 0 or 1.
7	Default number of columns of the default boot screen, set with ***Width***

Nb is the number of parameters you want to save. From 1 to 7.
Example: If ***Nb*** = 2, only the two first parameters (X and Y screen offset) will be saved.
Note: The MSX2 RTC's SRAM must be powered by battery to keep theses parameters saved in the MSX's memory when the computer is switch off.

VDPalternate V1.3 MSX2 *VDP*

```
void    VDPalternate (char oddpage, char OnTime, char OffTime)
```

Activates the alternate video mode. It's a special feature of the MSX2 VDP.
Two graphic pages are alternately displayed on the screen at a specific speed.



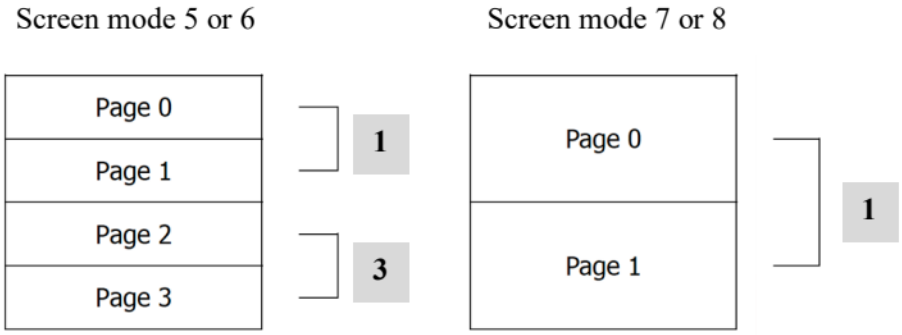
The ***odd page*** parameter indicates wich pages will be alternately displayed. With screen mode 5 or 6, specify **1** to alternately display pages 0 and 1 or specify **3** if you want to alternately display pages 2 and 3.
With screen mode 7 or 8 (up to 12 on MSX2+ and Turbo-r), as there are only 2 pages available, you must specify **1** in the ***odd page*** parameter.
The ***OnTime*** & ***OffTime*** parameters specify the display time for each of the page, it can vary from 0 to 15, according to this array :

speed	VDP 50hz	VDP 60hz	speed	VDP 50hz	VDP 60hz	speed	VDP 50hz	VDP 60hz
0	0 ms	0 ms	**6**	1200ms	1001ms	12	2400ms	2002ms
1	200ms	166ms	**7**	1400ms	1168ms	13	2600ms	2169ms
2	400ms	333ms	**8**	1600ms	1335ms	14	2800ms	2336ms
3	600ms	500ms	**9**	1800ms	1509ms	15	3000ms	2503ms
4	800ms	667ms	**10**	2000ms	1668ms			
5	1000ms	834ms	**11**	2200ms	1835ms			

VDPinterlace V1.3 MSX2 *VDP*

```
void    VDPinterlace (char oddpage)
```

Activates the interlace video mode. It's a special feature of the MSX2 VDP. It allows doubling the vertical resolution of the screen by alternatively displaying 2 graphic pages on the screen. Each pixel line is divided by two. One half of the even page, the other half on the odd page. Depending of the graphic mode used, the graphic games to be used are different.



The ***odd page*** parameter indicates wich pages will be alternately displayed. With screen mode 5 or 6, specify **1** to alternately display pages 0 and 1 or specify **3** if you want to alternately display pages 2 and 3.
With screen mode 7 or 8 (up to 12 on MSX2+ and Turbo-r), as there are only 2 pages available, you do not have choice, you must specify **1** in the ***odd page*** parameter.

Type Functions

IsAlphaNum — *TYPE*

```
char    IsAlphaNum (char c)
```

Checks if the supplied value ***c*** is alpha or numerical: ***A... Z*** or ***a... z*** or ***0...9***
Returned values:
1 if true
0 if false

IsAlpha — *TYPE*

```
char    IsAlpha (char c)
```

Checks if the supplied value ***c*** is alpha: ***A... Z*** or ***a... z***
Returned values:
1 if true
0 if false

IsAscii — *TYPE*

```
char    IsAscii (char c)
```

Check if the supplied value ***c*** is an ASCII character: ***- !..~***
Returned values:
1 if true
0 if false

IsCtrl — *TYPE*

```
char    IsCtrl (char c)
```

Checks if the supplied value ***c*** is a control symbol.
Returned values:
1 if true
0 if false

IsDigit — *TYPE*

```
char    IsDigit (char c)
```

Checks if the supplied value ***c*** is a digit: (***0..9***).
Returned values:
1 if true
0 if false

IsGraph — *TYPE*

```
char    IsGraph (char c)
```

Checks if the supplied value ***c*** is a graph representation.
Returned values:
1 if true
0 if false

IsLower *TYPE*

```
char    IsLower (char c)
```

Checks if the supplied value ***c*** is lower-case.
Returned values:
1 if true
0 if false

IsUpper *TYPE*

```
char    IsUpper (char c)
```

Checks if the supplied value ***c*** is upper-case.
Returned values:
1 if true
0 if false

IsPrintable *TYPE*

```
char    IsPrintable (char c)
```

Checks if the supplied value ***c*** is printable.
Returned values:
1 if true
0 if false

IsPunctuation *TYPE*

```
char    IsPunctuation (char c)
```

Checks if the supplied value ***c*** is a punctuation sign.
Returned values:
1 if true
0 if false

IsSpace *TYPE*

```
char    IsSpace (char c)
```

Checks if the supplied value ***c*** is a space.
Returned values:
1 if true
0 if false

IsHexDigit *TYPE*

```
char    IsHexDigit (char c)
```

Checks if the supplied value ***c*** is a hexadecimal digit.
Returned values:
1 if true
0 if false

IsPositive	*TYPE*
`int` **`IsPositive`** `(int c)`	
Checks if the supplied value ***c*** is positive. Returned values: **-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 ***a*** and ***b***	

String Functions

CharToLower — *STRING*

```
char    CharToLower (char ch)
```

Returns the lower-case version of the char value ***ch***

CharToUpper — *STRING*

```
char    CharToUpper (char ch)
```

Returns the upper-case version of the char value ***ch***

StrCopy — *STRING*

```
void    StrCopy (char *dst, char *src)
```

Copy string from ***src** to ***dst***

NStrCopy — *STRING*

```
void    NStrCopy (char *dst, char *src, unsigned 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**,
Returned values:

-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***.
Returned values:

-1	if (s1<s2)
0	if(s1=s2)
1	if (s1>s2)

StrChr — *STRING*

```
int     StrChr (char *string, char c)
```

Search for the char ***c*** inside ****string***.
Returns **1** if ***c*** is found, otherwise returns **-1**

StrPosStr — *STRING*

```
int     StrPosStr (char *s1, char *s2)
```

Search for the string ****s2*** inside string ****s1*** and return position of the 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 spaces inside the left part of ****string***

StrRightTrim — *STRING*

```
void    StrRightTrim (char *string)
```

Removes spaces in the right part of ****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 reverses the order of the chars inside the string ***str***
The new string is returned as a string, chars * array.

Itoa	V1.2	*STRING*

```
char*  Itoa (int num, char* str, int base)
```

This function converts an integer ***num*** to a string of chars ***str.*** The new string is returned as a string of chars, and must be declared before using the function, with enough ram to recover the converted number.
base indicates which base you want to convert to. It can be : **8**, **10** or **16**
Code example:

```
char temp[6];                    // add one more byte, to include the end of string
int number;
number=65535;
Itoa (number,temp,10);
```

StrToLower	V1.3	*STRING*

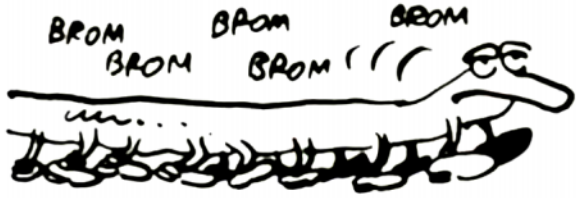
```
void   StrToLower (char *c)
```

Converts the ***c** string to lower-case. Original string is replaced.

StrToUpper	V1.3	*STRING*

```
void   StrToUpper (char *c)
```

Converts the ***c** string to upper case. The original string is replaced.



Memory Functions

Poke	***MEM***
`void  Poke (unsigned int address, char data)`	
Writes the byte (8 bits) value ***data*** to memory ***address***	

Pokew	***MEM***
`void  Pokew (unsigned int address, unsigned int data)`	
Writes the 2 bytes (16 bits) value ***data*** to memory ***address*** (2 bytes).	

Peek	***MEM***
`char  Peek (unsigned int address)`	
Reads and returns a byte (8 bits) value from memory ***address***	

Peekw	***MEM***
`int  Peekw (unsigned int address)`	
Reads and returns a 2 bytes value (16 bits) from memory ***address***. (Will peek 2 bytes).	

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

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

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

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

MemCompare — *MEM*

```
int     MemCompare (char *s1, char *s2, unsigned int n)
```

Compares ***n*** bytes of ****s1*** and ****s2***
Returned values:

-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
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.

BitReturn — V1.3 — *MEM*

```
char    BitReturn (char nbit, char byte)
```

Returns a bit from a byte. The bit number ***nbit*** is returned by the function.

BitReset — V1.3 — *MEM*

```
void    BitReset (char nbit, char *byte)
```

Resets a bit inside a byte. The bit ***nbit*** is reset inside ****byte***.

Interrupt Functions

EnableInterrupt		*INTERRUPT*
`void`	**`EnableInterrupt`** `(void)`	
Enables the Interrupt base system. Useful in some circumstances		

DisableInterrupt		*INTERRUPT*
`void`	**`DisableInterrupt`** `(void)`	
Disables the Interrupt base system. 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 and wait for next interrupt.		

InitInterruptHandler	V1.1	*INTERRUPT*
`void`	**`InitInterruptHandler`** `(char (*p_handler))`	
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.		

EndInterruptHandler	V1.1	*INTERRUPT*
`void`	**`EndInterruptHandler`** `(void)`	
End the Interruption handler initialized by the **SetInterruptHandler**. 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.		

InitVDPInterruptHandler	V1.3	*INTERRUPT*
`void`	**`InitVDPInterruptHandler`** `(char (*p_handler))`	
This function is based on the VDP interruption (50 Hz or 60 Hz). 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.		

EndVDPInterruptHandler	V1.1	*INTERRUPT*

```
void    EndVDPInterruptHandler (void)
```

End the Interruption handler initialized by the **SetVDPinterruptHandler**.
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.

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*
`char` **`PSGread`** `(char` **`psgreg`** `)`	
Read data from ***psgreg*** PSG register.	

PSGwrite	*PSG*
`void` **`PSGwrite`** `(char` **`psgreg`**`, 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 long     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 ;

Returned 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
```

FcbOpen — *FILE I/O*

```
char    FcbOpen (FCB *p_fcb)
```

Opens a file. Set the **FCB** Structure before using this function, all parameters are passed thru this structure. Return **0** if success.

FcbDelete — *FILE I/O*

```
char    FcbDelete (FCB *p_fcb)
```

Delete a file. Set the **FCB** Structure before using this function, all parameters are passed thru this structure. Return **0** if success.

FcbCreate — *FILE I/O*

```
char    FcbCreate (FCB *p_fcb)
```

Creates a file on media. Set the **FCB** Structure before using this function, all parameters are passed thru this structure. Return **0** on success. If the file already exists, it will be erased. This function also opens the file for writing process.

FcbClose — *FILE I/O*

```
char    FcbClose (FCB *p_fcb)
```

Closes a file previously opened. Return **0** on success.

FcbRead — *FILE I/O*

```
unsigned int   FcbRead (FCB *p_fcb, void *p_buffer, unsigned int nsize)
```

Read ***nsize*** bytes from opened file, and sends data to ****p_buffer*** .
****p_fcb*** is the open file handler. The function returns the number of bytes read.

FcbWrite — *FILE I/O*

```
char    FcbWrite (FCB *p_fcb, const void *p_buffer, unsigned int nsize)
```

Write ***nsize*** bytes to an open file, from ****p_buffer***
****p_fcb*** is the open file handler.

FcbFindFirst — *FILE I/O*

```
char    FcbFindFirst (FCB *p_fcb, FCB_FIND *p_result)
```

Finds first entry in the directory.

FcbFindNext — *FILE I/O*

```
char    FcbFindNext (FCB_FIND *p_result)
```

Finds next entry in the directory.

SetDisk — *FILE I/O*

```
char    SetDisk (unsigned int diskno)
```

Sets ***diskno*** as current drive number.
Return the number of available diks.

GetDisk — *FILE I/O*

```
char    GetDisk (void)
```

Gets and returns current drive number.

GetOversion — *FILE I/O*

```
char    GetOSversion(void)
```

Returns MSX-DOS OS version:

1	for MSX DOS 1
2	for MSX DOS 2
0	if not initiated

Other MSX-DOS Functions

Predefined macros & Structures

```
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 ;

typedef struct {
        unsigned int IX,IY,AF,DE,BC,HL;
} REGDATA;
```

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, how to get and show time:

```
TIME __at (0x2100) tm;
GetTime(&tm) ;
PrintDec(tm.hour) ; putch(' :');
PrintDec (tm.min) ; putch(' :');
PrintDec (tm.sec) ;
```

SetDate — *MSX_DOS*

```
char    SetDate (DATE *date)
```

Sets the system date thru the ***date*** structure.
Returns **0** if valid.

SetTime — *MSX_DOS*

```
char    SetTime (TIME *time)
```

Sets the system time thru the ***time*** structure.
Returns **0** if valid.

CallBios V1.3 *MSX_DOS*

```
void    CallBios (void *registers)
```

Performs a direct call to Bios routine. According to the BIOS function you want to call, the registers must be previously sets. The **REGDATA** Structure must be declared and configured before the call.
The parameter **register IX**, must contain the number of the routine to be called.
Example of use:

```
REGDATA Register;
REGDATA Register;           // Declare the structure
Register.IX=0x00A2;         // We want to call function 0xA2 (Console CHPUT)
Register.AF=0x42;           // We Want to print to console the letter B
CallBios(&Register);        // Call the routine
```

CallDos V1.3 *MSX_DOS*

```
void    CallDos (void *registers)
```

Performs a direct call to MSX-DOS routine. According to the DOS function you want to call, the registers must be previously set. The **REGDATA** Structure must be declared and configured before the call.
The parameter register C, must contain the number of the routine to be called.
Example of use:

```
REGDATA Register;           // Declare the structure
Register.BC=0x0002;         // We want to call function 0x02 (Console Output)
Register.DE=0x0041;         // We Want to print to console the letter A
CallDos(&Register);         // Call the routine
```

After the call, the registers structure's parameters ***AF, DE, BC, HL*** are updated if need.

CallSub DOS2 MSX2 V1.3 *MSX_DOS*

```
void    CallSub (void *registers)
```

Performs a direct call to MSX2 SUBROM routine. According to the SUBROM function you want to call, the registers must be previously sets. The **REGDATA** Structure must be declared and configured before the call.
The parameter register IX, must contain the number of the routine to be called.
Example of use:

```
REGDATA Register;           // Declare the structure
Register.IX=0x01B5;         // We want to switch to screen mode 8 (CHGMDP)
Register.BC=0x0008;         // We Want to print to console the letter A
CallSub(&Register);         // Call the routine
```

After the call, the registers structure's parameters ***AF, DE, BC, HL*** are updated if need.

SetRamDisk	DOS 2 V1.3	*MSX_DOS*

```
char    SetRamDisk(char NbSegments)
```

This function creates or destroy a RAMDisk.

If ***NbSegments*** = **255**
The function just returns the number of 16k RAM segments which are allocated to the RAMDisk currently.
A returned value of zero indicates that there is no RAM disk currently defined. I

If ***NbSegments*** = **0**
The current RAM disk will be destroyed, loosing all data which it contained and no error will be returned if there was no RAM disk.

If ***NbSegments*** is between **1 and 254**
The function will attempt to create a new RAM disk using the number of 16k segments specified by ***NbSegments***.
The returned value is the number of segments allocated.
Note that some of the RAM is used for the file allocation tables and the root directory so the size of the RAMDisk as indicated by "DIR" or "CHKDSK" will be somewhat smaller than the total amount of RAM used.
The RAMDisk will always be assigned the drive letter **"H:"** regardless of the number of drives in the system.

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 (unsigned int StartAddress, unsigned int Length)
```

Plays a PCM sound stored in the Vram
StartAddress 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 {
  chardrive;                    // 0 : default drive
  charfilename[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;
  chardevice;
  chardir_location;
  unsigned int top_cluster;
  unsigned int lastacsd_cluster;
  unsigned int clust_from_top;
  charrecord
  unsigned long rand_record
  charnone;                          //+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;
```

FCBlist — *IO*

```
FCBlist *FCB = FCBs ()
```

Mandatory when reading or writing files with MSX DOS 1. (Or MSX DOS 2)
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 118
close(fH) ;                              // Close file
```

DiskLoad — *IO*

```
char   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 address to CALL after loaded

Returns 0 on success.

Open — *IO*

```
unsigned int   Open (char *file_name, unsigned int mode)
```

Opens the file ****file_name.***
mode can be:

- O_RDONLY	0x0	: read only
- O_WRONLY	0x1	: write only

File operation errors are sent to the variable: **_io_errno**
The returned value is the file Handler, or -1 if error

Create — *IO*

```
int    Create (char *file_name)
```

Creates a file, named as ****file_name.*** The file is also opened at the same time.
The returned value is the file Handler, or -1 if error

Close — *IO*

```
char   Close (char fH)
```

Closes a file handler, previously opened. ***fH*** is the file handler that must be closed
File operation errors are sent to the variable: **_io_errno**
Returns 0 if success.

Read — *IO*

```
unsigned int   Read (unsigned int fH, void *buffer, unsigned int nbytes)
```

Read **nbytes** bytes from a opened file handler definied by ***fH*** to ****buffer***.
File operation errors are sent to the variable: **_io_errno**
Returns the number of bytes read.

Write — *IO*

```
unsigned int   Write (unsigned int fH, void *buffer, unsigned int nbytes)
```

Writes ***nbytes*** bytes from ****buffer*** to an opened file handler defined by ***fH*** handler.
File operation errors are sent to the variable: **_io_errno**
Returns the number of bytes actually written.

Ensure — DOS2 — *IO*

```
unsigned int   Ensure (unsigned int fH)
```

Ensure that a file handler previously writed is correct. Verify all flags and flush all data from buffers. ***fH*** is the file handler that must be ensured
File operation errors are sent to the variable: **_io_errno**

OpenAttrib — DOS2 — *IO*

```
char    OpenAttrib (char *file_name, unsigned int mode, unsigned int attr
)
```

Opens athe file ****file_name*** with attributs.
attr can be:

- O_CREAT	0x39	: create file mode
- O_EXCL	0x04	:
- O_TRUNC	0x31	:
- O_APPEND	0x41	: append
- O_TEMP	0x80	

mode can be the same as for the **Open** function.
File operation errors are sent to the variable: **_io_errno**
The returned value is the file Handler, or -1 if error

CreateAttrib — DOS2 — *IO*

```
unsigned int   CreateAttrib (char *file_name, unsigned int  attr)
```

Creates a file named ***file_name** and open it, with attributs.
attr is same as the **OpenAttrib** function. The file is also opened at the same time.
The returned value is the file Handler, or -1 if error

GetCWD DOS2 *IO*

```
unsigned int   GetCWD (char *buf, unsigned int bufsize)
```

Gets current working director. The directory will be store in the buffer ***buf***, number of data ***is*** limited by ***bufsize***
Return 0 on success

Ltell *IO*

```
unsigned int      Ltell (unsigned int fH, unsigned long value)
```

Moves the file handler ***fh*** pointer to ***value*** from the current pointer position.
Returns **0** on success, or error in **_io_error**

Lseek *IO*

```
unsigned int      Lseek (unsigned int fH, unsigned long value, unsigned
int offsetCode)
```

Moves the file handler pointer to ***value*** according to the method of the **offsetCode.**
Returns **0** on success, or error in **_io_error**
offsetCode can be:

- SEEK_SET	0	Relative to the beginning of the file
- SEEK_CUR	1	Relative to the current position
- SEEK_END	2	Relative to the end of the file

Remove *IO*

```
char    Remove (char *filename)
```

Removes file ***filename*** from directory.
Returns **0** on success, or error in **_io_error**

Rename *IO*

```
char    Rename (char *old_name, char *new_name)
```

Rename file or folder ***old_name*** by ***new_name***
Returns **0** on success, or error in **_io_error**

ChangeDir DOS2 *IO*

```
char    ChangeDir (char *Directory)
```

Sets current path to the ***Directory***.
Example: ChangeDir("Games")
Returns **0** or error **in _io_error**

FindFirst DOS2 *IO*

```
unsigned int   FindFirst (char *wildcard, char *result, unsigned int attr)
```

Finds files or folders by wildcard as « *.COM », « ???? », etc.
Returns **0** on success, or error in _io_error.

FindNext DOS2 *IO*

```
unsigned int   FindNext (char *result)
```

Find next occurrence.
See **FindFirst** function for details
example :

```
 n=FindFirst(" *.* ",sbuf,0) ;
for( ;!n ;){
    cputs(sbuf) ; cputs(sn);  n=FindNext(sbuf) ;
}
```

MakeDir DOS2 *IO*

```
char    MakeDir (char *Folder_Name)
```

Creates the directory ***Folder_Name***.
Returns **0**, or error in **_io_error**

RemoveDir DOS2 *IO*

```
char    RemoveDir (char *Folder_Name)
```

Removes the directory ***Folder_Name***.
Returns 0, or error in **_io_error**

GetDiskParam DOS2 V1.1 *IO*

```
char    GetDiskParam (DSKPARAMS *info, char Drive)
```

After Initialization of the DSKPARAMS Structure the call of this function will get all parameters of the disk drive. It may be a floppy Drive or any other MSX-DOS 2 's storage drive.
See the DSKPARAMS Structure at the beginning of this chapter for details.
(Available with **MSX-DOS 2 ONLY**)

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).

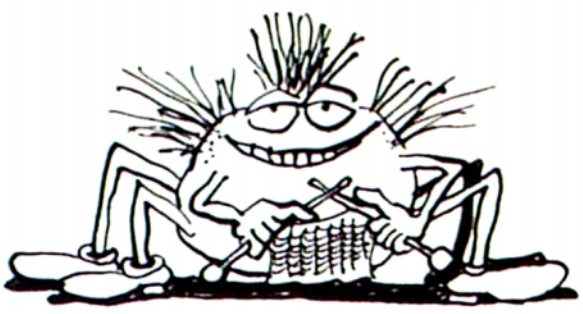
GetDiskTrAddress V1.1 *IO*

```
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*
`char  SectorRead (unsigned int SectorStart, char drive, 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*
`char  SectorWrite (unsigned int SectorStart, char drive, 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 addrPalettes, unsigned int addrColors)
```

Writes screen from RAM memory addresses to VRAM, very dumb one-time way (2x6144 bytes)
Use with Screen mode 2 or 3.

SC2ReadScr — *VDP_GRAPH1*

```
void    SC2ReadScr (unsigned int addrPalettes, unsigned int addrColors)
```

Reads screen from VRAM to memory addresses, very dumb one-time way (2x6144 bytes)
Use with Screen mode 2 or 3.

Get8px — *VDP_GRAPH1*

```
unsigned int   Get8px (char X, char Y)
```

Gets byte of 8-pixels at (X, Y)

ReadBlock — *VDP_GRAPH1*

```
void    ReadBlock (char X, char Y, char DX, char DY, unsigned int
addr_toPalettes, unsigned int addr_toColours)
```

VRAM to RAM copy (copy block to memory)
X and ***Y*** left upper corner of screen position to copy
DX and ***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 (char X, char Y, char dx, char dy, unsigned int
addr_Palettes, unsigned int addr_Colours)
```

RAM to VRAM. Copy from memory to screen vram.
(***X,Y***) – where to put on screen

Get1px — *VDP_GRAPH1*

```
char   Get1px (char X,  char Y)
```

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

Set8px — *VDP_GRAPH1*

```
void   Set8px(char X,  char Y)
```

Sets whole byte of 8-pixels at (***X, Y***)

Set1px — *VDP_GRAPH1*

```
void   Set1px(char X,  char Y)
```

Sets pixel of 8-pixels at (***X, Y***)

Clear8px — *VDP_GRAPH1*

```
void   Clear8px (char X, char Y)
```

Clears byte (sets=0) of 8-pixels at (***X, Y***)

Clear1px — *VDP_GRAPH1*

```
void   Clear1px (char X,  char Y)
```

Clears pixel (sets=0) of 8-pixels at (***X, Y***)

GetCol8px — *VDP_GRAPH1*

```
void   GetCol8px (char X, char Y, pxColor *C )
```

Get color of 8-pixel pattern at (X, Y) to ***C***
See the predefined structure.

SetCol8px — *VDP_GRAPH1*

```
void   SetCol8px (char X,  char 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***

```
char    SC2Point (char X,  char Y)
```

Gets color of pixel at (*X,Y*), (same for 8-pixel pattern)
Use only with Screen mode 2.

SC2Pset	***VDP_GRAPH1***

```
void    SC2Pset (char X, char Y, char color)
```

Puts pixel at (***X, Y***) position, with color ***color*** .
Use only with Screen mode 2.

SC2Line	***VDP_GRAPH1***

```
void    SC2Line (char X, char Y, char X2,  char Y2, char color)
```

Draws a line from (***X , Y***) position to (***X2, Y2***) position, with color ***color***,
Does not change background color
Use only with Screen mode 2.

SC2Paint	***VDP_GRAPH1***

```
void    SC2Paint (int X, int Y, int color)
```

Paints for small screen regions with color **color** at ***X, Y*** coordinates. Slow Function!
Use with Screen mode 2.

SC2BoxFill	**V1.2**	***VDP_GRAPH1***

```
void    SC2BoxFill (char X1, char Y1, char X2, char Y2, char color)
```

Draws a filled rectangle ***from X1, Y1*** (left upper corner) to ***X2, Y2*** (right bottom corner) with ***color***. No Logical operation. Use only with Screen mode 2.

SC2BoxLine	**V1.3**	***VDP_GRAPH1***

```
void    SC2BoxLine (char X1, char Y1, char X2, char Y2, char color)
```

Draws anempty rectangle ***from X1, Y1*** (left upper corner) to ***X2, Y2*** (right bottom corner) with ***color***. No Logical operation.
Use only with Screen mode 2.

MSX2 GRAPHICS

[vdp_graph2.h]

MSX2 specific graphic functions.

Predefined Structures and macros

Logical fill for rectangle and circle

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

Palette

```
typedef struct {
      char      colour ;    // color number 0..15
      char      R ;         // 0..7 red brightness
      char      G ;         // 0..7 green brightness
      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)
      chars0 ;                  // set to 0, dummy 1st empty byte sent to chip
       charDI ;                 // set to 0 (b), works well from left to right
      charLOP ;                 // 0 to copy (a), Logical+Operation
} MMMtask ;
```

Logical operations

When graphic commands are called, various logical operations can be perrformed 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 and DC
2	LOGICAL_OR	DC=SC or DC
3	LOGICAL_XOR	DC=SC xor DC
4	LOGICAL_NOT	DC=SC ! DC
8	LOGICAL_TIMP	if SC=0 then DC=DC else DC=SC
9	LOGICAL_TAND	if SC=0 then DC=DC else DC=SC and DC
10	LOGICAL_TOR	if SC=0 then DC=DC else DC=SC or DC
11	LOGICAL_TXOR	if SC=0 then DC=DC else DC=SC xor DC
12	LOGICAL_TNOT	if SC=0 then DC=DC else SC ! DC

One of the most used operator is TIMP. It permit to copy a part of the screen over another without the transparent color (color 0 is transparent).

vMSX — *VDP_GRAPH2*

```
char        vMSX (void)
```

Check MSX VDP version.
Returns 1 for MSX1, or **2** for MSX2. DOES IT RETURNS V9958 MSX2+ ?

Pset — MSX2 — *VDP_GRAPH2*

```
void    Pset (unsigned int X, unsigned int Y, char color, char OP)
```

Draws a pixel at ***X, Y*** with the defined ***color*** and logical operation ***OP***

Point — MSX2 — *VDP_GRAPH2*

```
char    Point (unsigned int X, unsigned int Y)
```

Reads and return color of the pixel at ***X, Y***

Line — MSX2 — *VDP_GRAPH2*

```
void    Line (unsigned int X1, unsigned int Y1, unsigned int X2, unsigned
int Y2, char color, char OP)
```

Draws a line from ***X1, Y1 to X2 ,Y2*** with the defined ***color*** and logical operation ***OP***

BoxLine — MSX2 — *VDP_GRAPH2*

```
void    BoxLine (unsigned int X1, unsigned int Y1, unsigned int X2,
unsigned int Y2, unsigned int color, char 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.

BoxFill — V1.2 MSX2 — *VDP_GRAPH2*

```
void    BoxFill (unsigned int X1, unsigned int Y1, unsigned int X2,
unsigned int Y2, 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**.

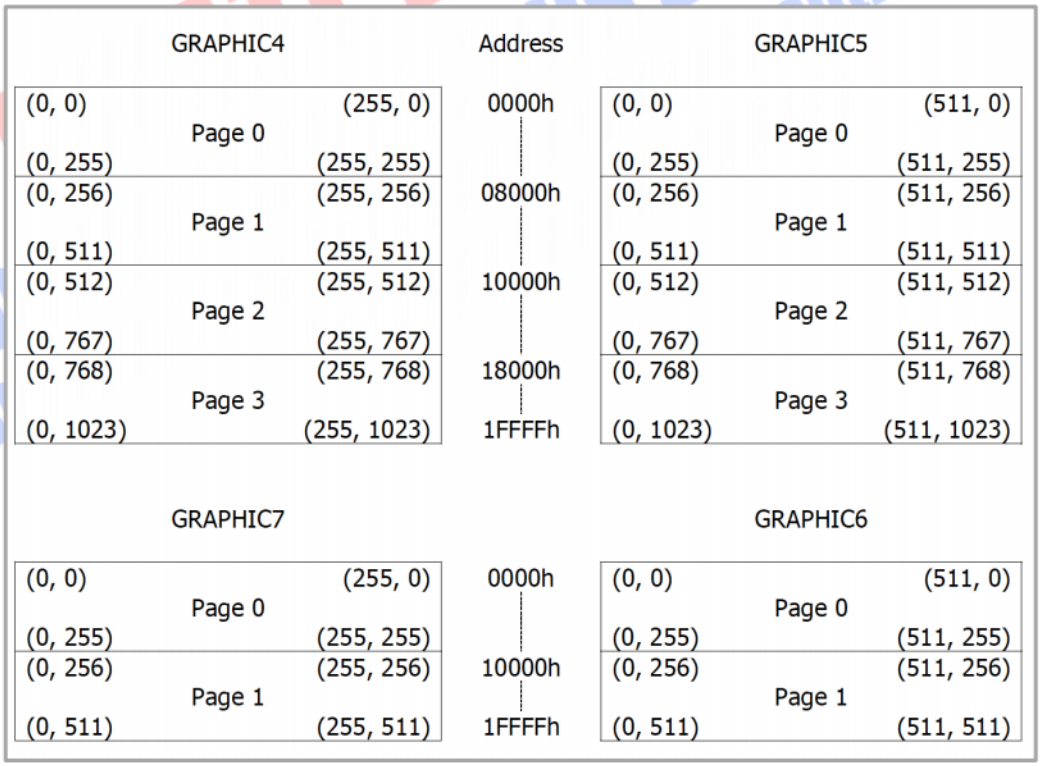
HIGH SPEED VDP COMMANDS

The YAMAHA V9938 VDP Processor of the MSX2, and the YAMAHA V9958, the VDP processor of the MSX2+ and Turbo-R comes with high speed functions you can use to build your own graphic routines.

VDP COMMANDS SUMMARY					
Type	**Source**	**Destination**	**Unit**	**Name**	**Code**
High speed move	CPU	VRAM	Byte	**HMMC**	**0xF0**
	VRAM	VRAM		**YMMM**	**0xE0**
	VRAM	VRAM		**HMMM**	**0xD0**
	VDP	VRAM		**HMMV**	**0xC0**
Logical Move	CPU	VRAM	Dot	**LMMC***	**0xB0**
	VRAM	CPU		**LMCM**	**0xA0**
	VRAM	VRAM		**LMMM**	**0x90**
	VDP	VRAM		**LMMV**	**0x80**
Line	VDP	VRAM		**LINE *****	**0x70**
Search	VDP	VRAM		**SRCH ****	**0x60**
Pset	VDP	VRAM		**PSET *****	**0x50**
Point	VRAM	VDP		**POINT *****	**0x40**
Stop				**STOP**	**0x00**

* Split into 2 individual commands, LMCM5 & LMCM8. Standard LMCM can be used with ***fVDP*** function
** Not implemented as individual command. But you can use it via ***fVDP*** function
*** Functions described in the previous chapter

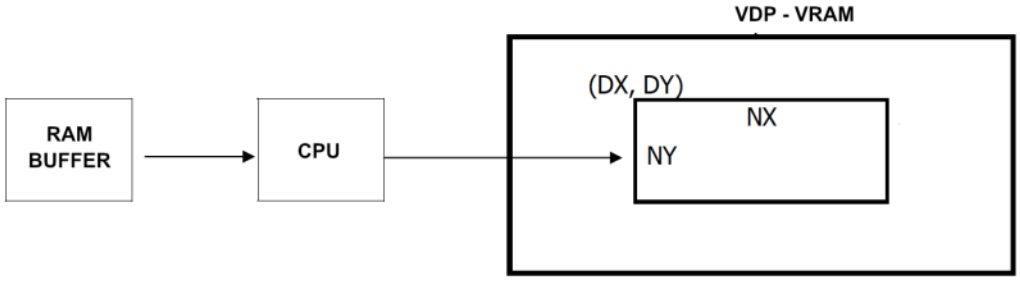
Theses commands use the global coordinates system of the whole MSX's 128 KB VRAM.



HMMC MSX2 *VDP_GRAPH2*

```
void   HMMC (void *pixeldata, unsigned int DX unsigned int DY, unsigned
int NX, unsigned int NY)
```

High speed move CPU to VRAM.
This function sends the RAM ***pixeldata*** buffer to VRAM at ***DX, DY*** coodonates.
The datas sent must be a rectangle.
NX is length of the zone to copy in pixels
NY is height of the zone to copy in pixels
Use ***DY*** Coordinate >= 256 to copy the buffer to other vram page.

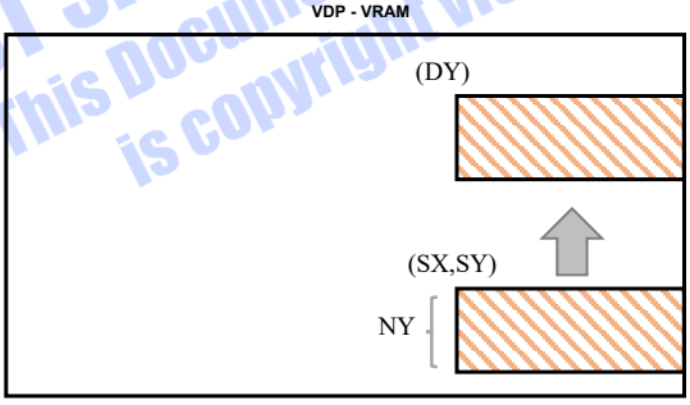


YMMM V1.2 MSX2 *VDP_GRAPH2*

```
void   YMMM (unsigned int SX, unsigned int SY, unsigned int DY, unsigned
int NY)
```

High speed move VRAM to VRAM on Y coordinate only.
This function only copy the rectangle portion of the image from position
(***SX, SY***) - (***255, SY+NX***) to the position ***DY***

This function is usefull to move full image or portion of an image to another VRAM page.
Note: Hardware limitation. In Screen mode 5 & 7, SX must be an even number.
In screen mode 6, SX and NY must be a multiple of 4.



LMMC — V1.2 MSX2 — *VDP_GRAPH2*

```
void   LMMC (void *pixeldata, unsigned int DX, unsigned int DY, unsigned int NX, unsigned int NY, char OP)
```

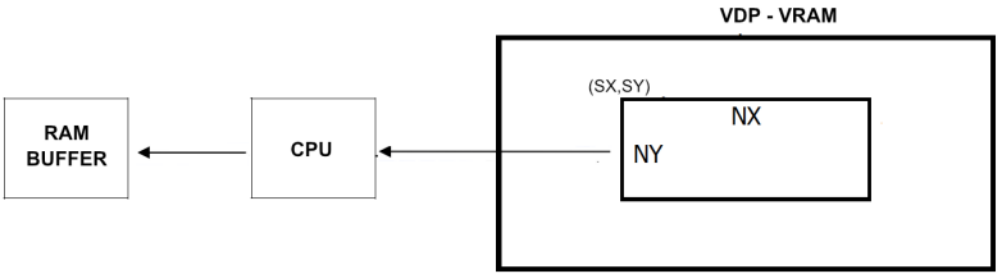
Logical Move CPU to VRAM. (Same as *HMMC* but with Logical operator)
This function sends the RAM ***pixeldata*** buffer to VRAM at ***DX, DY*** coodonates.
The datas sent must be a rectangle.
NX is length of the zone to copy in pixels
NY is height of the zone to copy in pixels
OP is a standard logical operator parameter. See the list in previous chapters.
Note: In Screen mode 5 or 7, if you want ot use LMMC command, you must previously tranfer data to RAM buffer with ***LMCM8*** *instead of* ***LMCM5***

LMCM5 — V1.3 MSX2 — *VDP_GRAPH2*

```
void   LMCM5 (void *buffer, unsigned int SX, unsigned int SY, unsigned int NX, unsigned int NY, char OP);
```

Logical Move VRAM to CPU.
This function sends a copy of the VRAM to the RAM ***pixeldata*** buffer.
The VRAM block sent is a rectangle. The top left of the rectangle is at ***SX***, ***SY*** coordinates.
The length of the rectangle is ***NX*** pixels, and the height of the rectangle is ***NY*** pixels.
This function is coding 2 pixels into 1 byte, so, it works well with Screen modes 5 or 7.
OP is a the logical operator to apply. Two options are possible:

0 IMP
8 TIMP (In other words this operator will copy to RAM all bytes except those with a value of 0)



LMCM8 — V1.3 MSX2 — *VDP_GRAPH2*

```
void   LMCM8 (void *buffer, unsigned int SX, unsigned int SY, unsigned int NX, unsigned int NY, char OP);
```

Logical Move VRAM to CPU. (Same as previous function, but for Screen mode >= 8)
This function sends a copy of the VRAM to the RAM ***pixeldata*** buffer.
The VRAM block sent is a rectangle. The top left of the rectangle is at ***SX***, ***SY*** coordinates.
The length of the rectangle is ***NX*** pixels, and the height of the rectangle is ***NY*** pixels.
This function is coding 1 color pixel per byte, so it works well with Screen modes 8 to 12.
But it may be used also in screen mode 5 or 7 if you need to send back the buffer to the VDP with a logical operator
OP is the logical operator to apply. Only two are options are possible.

0 IMP
8 TIMP (In other words this operator will copy to RAM all bytes except those with a value of 0)

LMMM — V1.2 MSX2 — *VDP_GRAPH2*

```
void   LMMM (unsigned int SX, unsigned int SY, unsigned int DX, unsigned int DY, unsigned int NX, unsigned int NY, char OP)
```

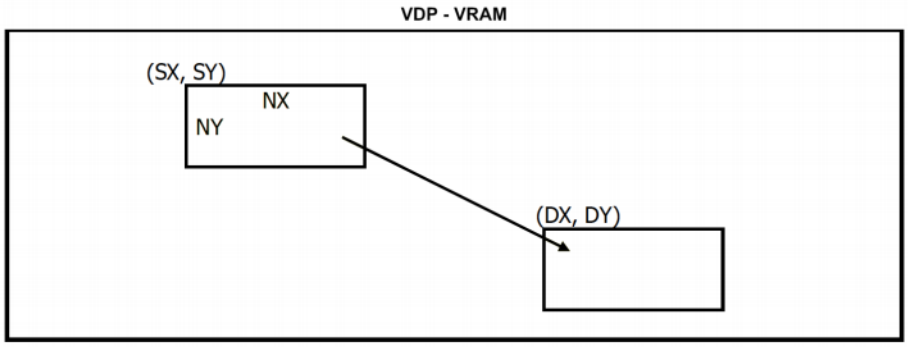
Logical move VRAM to VRAM. (Same as HMMM but with a logical operator)
This function copy a rectangle image starting at the top left coordinates ***SX, SY*** to the destination ***DX, DY*** coordinates. The length of the rectangle is ***NX*** pixels, and the height of the rectangle is ***NY*** pixels.
OP is a standard logical operator parameter. See the list an codes in previous chapters.

HMMM — V1.2 MSX2 — *VDP_GRAPH2*

```
void   HMMM (unsigned int SX, unsigned int SY, unsigned int DX, unsigned int DY, unsigned int NX, unsigned int NY)
```

High speed from VRAM to VRAM
This function copy a rectangle image starting at the top left coordinates ***SX,SY*** to the destination ***DX,DY*** coordinates. The length of the rectangle is ***NX*** pixels, and the height of the rectangle is ***NY*** pixels.

Note: Hardware limitation. In Screen mode 5 & 7, SX & DX must be an even number. In screen mode 6, SX and DX must be a multiple of 4.



HMMV — V1.2 MSX2 — *VDP_GRAPH2*

```
void   HMMV (unsigned int DX, unsigned int DY, unsigned int NX, unsigned int NY, char COL)
```

High speed move VDP to VRAM
Fill a rectangle starting at top left corner ***DX***, ***DY*** with color ***COL***
The length of the rectangle is ***NX*** pixels, and the height of the rectangle is ***NY*** pixels.
Note: When working with screen 5 or 7, HMMV will fill 2 horizontal pixels at the same time. The ***COL*** parameter must be divided into two blocks of 4 bits
example color: ***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 color.
You can also use this formula to calculate the good value of your ***COL parameter***:

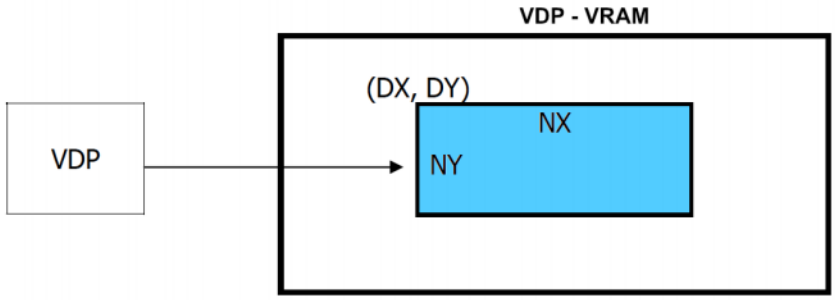
```
color =12;                          // use color 12
color=((color << 4) | color);       // Use color 12 for both pixels
```

Note: Hardware limitation. In Screen mode 5 & 7, DX & NX must be an even number. In screen mode 6, DX and NX must be a multiple of 4.

LMMV v1.2 MSX2 *VDP_GRAPH2*

```
void   LMMV (unsigned int SX, unsigned int SY, unsigned int DX, unsigned int DY, char Color, char OP)
```

Logical Move VDP to VRAM (Same effect as HMMV, but with a logical operator)
Fill a rectangle starting at top left corner ***DX***, ***DY*** with color ***COL***
The length of the rectangle is ***NX*** pixels, and the height of the rectangle is ***NY*** pixels.



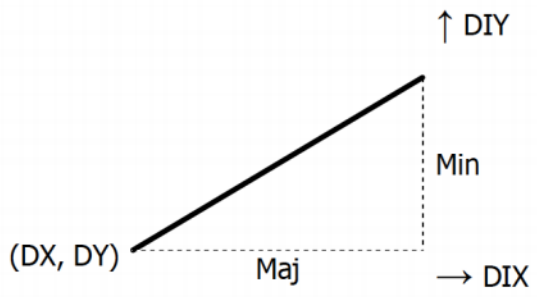
VDPLINE v1.2 MSX2 *VDP_GRAPH2*

```
void   VDPLINE (unsigned int DX, unsigned int DY, unsigned int NX, unsigned int NY, unsigned int COL, char PARAM, char OP)
```

Draw a line with the dedicated VDP command. (The Line Function is using this VDP command).
The line drawn is the hypotenuse of the triangle defined by the "Long" side and the "Short" side. The distances are defined from the starting point coordinates ***DX, DY***
NX parameter must contain the size in pixels of the longest segment of the triangle.
NY parameter must contain the size in pixels of the shortest segment of the triangle.
COL must contain the color number to use to draw the line.
OP is a standard logical operator parameter. See the list in previous chapters.
PARAM, contains parameters coded in a byte

Bit	7	6	5	4	3	2	1	0
	0	0	MXD	0	DIY	DIX	0	MAJ

MAJ bit : must be **0** if the long side is on X axis, **1** if long side is on Y axis
DIX bit : must be **0** if the line goes to the right, **1** if it goes to the left
DIY bit : must be **0** if the line goes to down, **1** if it goes to up.
MXD bit : must be **0** as long as your MSX do not have expansion of VRAM memory



fLMMM MSX2 *VDP_GRAPH2*

```
void    fLMMM (MMMtask *VDPtask)
```

High speed copy by structure ***VDPtask***. Same effect as LMMM command but it is using a structure where you can set your parameters. By using this method the command is faster completed.
First declare the structure, for example : ***static MMMtask t_vdp ;***

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)
        char 0;                  // set to 0. 1st empty byte sent to chip
        char DI;                 // set to 0 (b), works well from left to right
        char LOP;                // 0 to copy (a), Logical Operation
} MMMtask;
```

Note: The fLMMM is useless since the introduction of fVDP command. It is here for compatibility reasons. It's better to use fVDP function.

fVDP V1.3 MSX2 *VDP_GRAPH2*

`void fVDP (void *parameters)`

This Fast VDP function give you access to all VDP Commands in the fastest way possible.
To use this function you must first initialize the dedicated predefined structure ***FastVDP***

```
typedef struct {
        unsigned int sx;        // source X (0 to 511)
        unsigned int sy;        // source Y (0 to 1023)
        unsigned int dx;        // destination X (0 to 511)
        unsigned int dy;        // destination Y (0 to 1023)
        unsigned int nx;        // width (0 to 511)
        unsigned int ny;        // height (0 to 511)
        unsigned char col;      // color used by some commands. or 0 if not used
        unsigned char param;    // Parameters set the direction ex : opDOWN | opRIGHT
        unsigned char cmd;      // VDP Comd + Logical Operator ex: opLMMM | LOGICAL_TIMP
} FastVDP;
```

Command & code		Logical operator & code		Direction & code	
opHMMC	0xF0	LOGICAL_IMP	0x00	opRIGHT	0x0
opYMMM	0xE0	LOGICAL_AND	0x01	opDOWN	0x0
opHMMM	0xD0	LOGICAL_OR	0x02	opLEFT	0x04
opHMMV	0xC0	LOGICAL_XOR	0x03	opUP	0x08
opLMMC*	0xB0	LOGICAL_NOT	0x04		
opLMCM	0xA0	LOGICAL_TIMP	0x08		
opLMMM	0x90	LOGICAL_TAND	0x09		
opLMMV	0x80	LOGICAL_TOR	0x0A		
opLINE	0x70	LOGICAL_TXOR	0x0B		
opSRCH	0x60	LOGICAL_TNOR	0x0C		
opPSET	0x50				
opPOINT	0x40				
opSTOP	0x00				

Some precisions about the variables.
DI is the direction the VDP command will use to make the action. The most common case is to leave it at 0. In some cases, for example if you want to make a right to left scrolling, it can be usefull to change it.
DI is a byte, but only the 4 first bits are used. You can combine two direction by using a logical "or" operator " |"

DI byte							
				D3	D2	D1	D0

CMD is the VDP command code to use. this 8 bits value is separate into 2 quartets.

CMD byte							
C3	C2	C1	C0	L3	L2	L1	L0

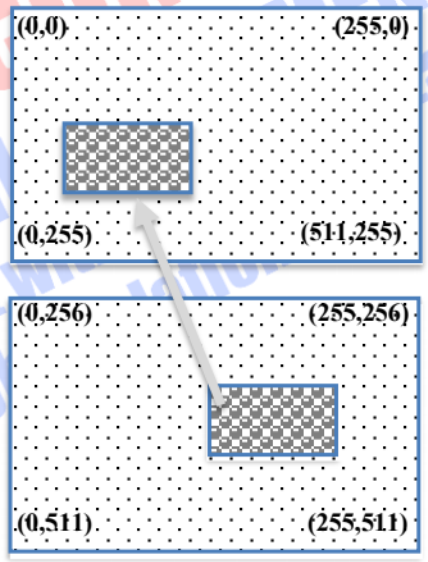
The most significant quartet is the command code (C0 to C3), the less significant quatrtet is the logical operator code (L0 to L3).
Only some commandes accept a logical operator (see the summary array at the beginning of this chapter). To apply a logical operator, use an logical "or" on the CMD code, or leave it to 0.

Example:

Imagine we want to copy a portion of the image of the screen mode 8, from page 1 to page 0 with an transparent logical operator.
The block we want to copy is at (150, 100) on page 1. Its width is 100 pixels, and height 50 pixels.
We want to copy this block to (20,25) on page 0.

```
static FastVDP MyCommand;                // Set the structure as global

MyCommand.SX=150;                        // Set the X source
MyCommand.SY=100+256;                    // Set the Y source on page 1
MyCommand.DX=20;                         // Set the X destination
MyCommand.DY=25;                         // Set the Y destination
MyCommand.NX=150;                        // Set the width of the block
MyCommand.NY=50;                         // Set the height of the block
MyCommand.COL=0;                         // no need color. Leave 0
MyCommand.DI=0;                          // No need to change process direction
MyCommand.CMD=opHMMM | LOGICAL_TIMP;     // Command and Logical operator

fVDP(&MyCommand);                        // Call the command
```

Note : For the next uses, you just have to modify the needed variables.



	HMMC	LMMC	YMMM	HMMM	LMMM	HMMV	LMMV	LMCM
R32	-	-	-	SX	SX	-	-	SX
R33								
R34	-	-	SY	SY	SY	-	-	SY
R35								
R36	DX	DX	DX	DX	DX	DX	DX	-
R37								
R38	DY	DY	DY	DY	DY	DY	DY	-
R39								
R40	NX	NX	-	NX	NX	NX	NX	NX
R41								
R42	NY	NY	NY	NY	NY	NY	NY	NY
R43								
R44	Col*	Col*	-	-	-	Col*	Col*	-
R45	Param*	Param*	Param*	Param**	Param**	Param*	Param*	Param**
R46	0xF0	0xB0|Log*	0xE0	0xD0	0x90|Log*	0xC0	0x80|Log*	0xA0

	LINE	PSET	POINT
R32	-	-	SX
R33			
R34	-	-	SY
R35			
R36	DX	DX	-
R37			
R38	DY	DY	-
R39			
R40	NX	-	-
R41			
R42	NY	-	-
R43			
R44	Color	Color	-
R45	Param***	Param	Param
R46	0x70|Log*	0x50|Log*	0x40

* parameters :

7	6	5	4	3	2	1	0
0	-	MXD	-	DIY	DIX	-	-

MXD : Destination : VRAM=0. Expansion VRAM=1.
DIY : Y transfert direction : Down=0. Up=1.
DIX : X transfert direction : Right=0. Left=1.

* Col : First byte to send
* Log : Logical Operation (4 bits)

** parameters :

7	6	5	4	3	2	1	0
0	-	MXD	MSX	DIY	DIX	-	-

MXD : Destination : VRAM=0. Expansion VRAM=1.
MSX : Source : Vram=0. Expansion VRAM=1.
DIY : Y transfert direction : Down=0. Up=1.
DIX : X transfert direction : Right=0. Left=1.

*** parameters for LINE:

7	6	5	4	3	2	1	0
0	-	MXD	-	DIY	DIX	-	MAJ

MXD : Destination : VRAM=0. Expansion VRAM=1.
MAJ : Longest sebgment is vertical=1. Longest segment is horizontal=0.
DIY : Trace from down to UP=1. Trace from up to down=0.
DIX : Trace from right to left=1. Trace from left to right=0.

PAINT

[vdp_paint.h]

The fast FloodFill routine for MSX2 and upper. Must be used in conjunction with ***vdp_graph2.h***

SetPaintBuffer	V1.3 MSX2	***VDP_PAINT***

```
void    SetPaintBuffer (char* BufAddr)
```

This function defines and initialize the mandatory PaintBuffer.

Paint	V1.3 MSX2	***VDP_PAINT***

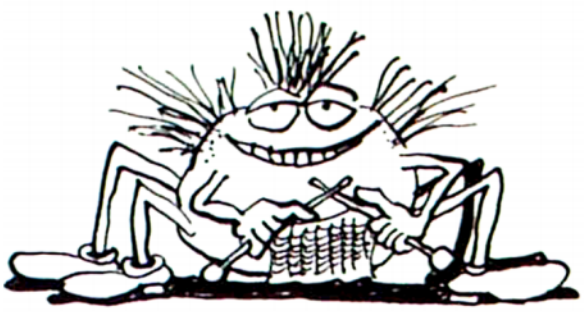
```
void    Paint (unsigned int X, unsigned int Y, unsigned int COLOR)
```

This fast Flood Fill routine for MSX2 paints any area of the VDP VRAM.
The paint process will start at ***X, Y*** coordinates. The color found at ***X, Y*** is considered as the background color, and is replaced by ***COLOR.***
This function needs a buffer to work. This buffer must be initialized before the first use.
Use exactly this way inside your **main**:

```
unsigned char *PaintBuffer;
Paint_vars.MaxRam=MAXPAINT_BUFFER;
PaintBuffer=MMalloc(Paint_vars.MaxRam);
SetPaintBuffer(PaintBuffer);
```

The default buffer size is 250 Bytes. The size varies according to the number of different zones to be treated by the PAINT routine. If you notice that certain zones are not treated, increase this buffer by modifying the value of **MAXPAINT_BUFFER** in the file **fusion-c / heaber/vdp_paint.h**
When you no longer need to use the paint function. Do not forget to free the memory allocated to the buffer. In any case, free this memory at the end of your program.

```
Free (PaintBuffer);
```

SPRITES
[vdp_sprites.h]

All necessary function to control MSX Hardware sprites.

SpriteOn — MSX2 — *VDP_SPRITES*

```
void    SpriteOn (void)
```

Enables Sprites.

SpriteOff — MSX2 — *VDP_ SPRITES*

```
void    SpriteOff (void)
```

Disables Sprites.
Note: Disabling sprite, make VDP run a little faster.

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
Note that 16x16 pixels sprites are in fact composed with four 8x8 pixels patterns. See ***SetSpritePattern*** function for details.

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 MSX2 *VDP_SPRITES*

```
char    SpriteCollisionX (void)
```

Returns the **X** position of the sprite's collision.

SpriteCollisionY MSX2 *VDP_SPRITES*

```
Char    SpriteCollisionY (void)
```

Returns the **Y** position of the sprite's collision.
Always read the X position of the collision before reading the Y position, or the result will be false.

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*** with color ***color***.
On MSX2 and upper you must define the sprite color with ***SpriteColor*** functions.
In case you are using the 16x16 pixels sprite mode, ***pattern_n*** must be equal to the first pattern of the serie (aka Pattern 0, if you are referring to the previous schematic)

fPutSprite V1.3 *VDP_SPRITES*

```
void    fPutSprite ( void *parameters)
```

This function is faster than the standard PutSprite function. It uses a dedicated predefined structure to pass the parameters: ***FastSPRITE***.

```
 typedef struct {
    char spr;          // Sprite ID
    char y;            // X destination of the Sprite
    char x;            // Y destination of the sprite
    char pat;          // Pattern number to use
    char col;          // Color to use (Not usable with MS2's sprites)
} FastSPRITE;
```

To use this function first declare the structure, for example like this:

```
static FastSSPRITE MySprite;
```

Then assign the parameters to the structure, and call the function, like this:

```
MySprite.spr=1;              // Attribut for Sprite 1
MySprite.y=player.y;         // Y coordoniunate of the sprite
MySprite.x=player.x;         // X coordinate of the sprite
MySprite.pat=8;              // Pattern to use
MySprite.col=1;              // Color to use
fPutSprite(&MySprite);       // Call the function
```

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.
In case you are using 16x16 pixels sprite mode, ****p_pattern*** must be composed of 32 bytes in a specific order. A 16x16 pixels sprite's pattern is composed of four 8x8 pixels sprite's patterns like this:

16x16 pixels sprite

Pattern n0	**Pattern n2**
Pattern n1	**Pattern n3**

Thus the 8 first bytes compose the pattern **n0**, the 8 following bytes compose the pattern **n1**, the next 8 bytes compose the pattern **n2**, and finally the 8 last bytes compose the pattern **n3**.

Sprite32Bytes — *VDP_SPRITES*

```
Char    *Sprite32Bytes (unsigned int *bindata)
```

Convert a 32 bytes array to a 16x16 pattern usable by the ***SetSpritePattern***.

```
static const  unsigned int pattern[]={
0b1111111111111111,
0b1111111111111111,
0b1111111111111111,
0b1111111111111111,
0b1111111111111111,
0b1111111001110011,
0b1111110000100001,
0b1111110000100001,
0b1111110000100001,
0b1111110000100001,
0b1111111111111111,
0b1111111111111111,
0b1111111111111111,
0b1111111111111111,
0b1111111111111111,
0b0000000000000000};
```

```
SetSpritePattern(0,Sprite32Bytes(pattern),32);
```

SpriteOverlap — V1.3 — *VDP_SPRITES*

```
char    SpriteOverlap (void)
```

Checks if more than 4 sprites are on the same horizontal line on MSX1's screen modes; or more than 8 sprites on the same horizontal line on MSX2's screen modes. Returns **1** if overlapping is detected.

SpriteOverlapId — *VDP_SPRITES*

```
char    SpriteOverlapId (void)
```

In case of sprites overlapping, this function returns the **ID** of the first extra sprite on the same horizontal line. (The **ID** of the 5th sprite, or the 9th sprite that cause overlapping).

SetSpriteColors MSX2 *VDP_SPRITES*

```
void    SetSpriteColors (char spriteNumber, char *data)
```

With MSX2 Screen's modes, you can define each line of the sprite ***spriteNumber*** with a specific color, set in ****data***.
****data*** must be an array of 16 bytes, one byte for each color line of a 16x16 pixels sprite. If you are using 8x8 pixels sprites mode, only the first 8 bytes of the array will be used.
In Screen Mode 8 the sprite palette is fixed and cannot be redefined.
Here the Sprite color palette used in Screen mode 8:

	R	G	B	Name
00	0	0	0	Black
01	0	0	2	Dark Blue
02	3	0	0	Dark Red
03	3	0	2	Dark Purple
04	0	3	0	Dark Green
05	0	3	2	Turquoise Blue
06	3	3	0	Green Olive
07	3	3	2	Grey
08	4	4	2	Ligh Orange
09	0	0	7	Blue
10	7	0	0	Red
11	7	0	7	Purple
12	0	7	0	Green
13	0	7	7	Light Blue
14	7	7	0	Yellow
15	7	7	7	White

Pattern16RotationVram V1.3 *VDP_SPRITES*

```
void    Pattern16RotationVram (char pattern, signed char rotation, char
DestPattern)
```

Rotates the 16x16 pixels sprite pattern n° ***pattern*** stored inside the VRAM. The new rotated pattern can be copied over the same pattern number by setting ***DestPattern*** to **0**, or copied to another pattern location by setting ***DestPattern*** from **1** to **255**. (Do not forget that one 16x16 pattern is in fact four 8x8 patterns inside the pattern table).
rotation indicates the angle, it can be:
90 90° rotation to the right
-90 90° rotation to the left

Pattern8RotationVram V1.3 *VDP_SPRITES*

```
void    Pattern8RotationVram (char pattern, signed char rotation, char
DestPattern)
```

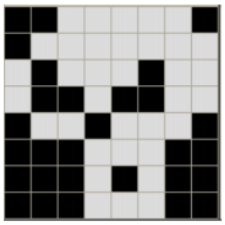
Rotates the 8x8 pixels sprite pattern n° ***pattern*** stored inside the VRAM. The new rotated pattern can be copied over the same pattern number by setting ***DestPattern*** to **0**, or copied to another pattern location by setting ***DestPattern*** from **1** to **255**.
rotation indicates the angle, it can be:
90 90° rotation to the right
-90 90° rotation to the left

Note about Pattern Rotation and Pattern flip: They can be applied to Sprites Pattern in RAM, but also to any 8x8 array, like fonts characters, or any bitmap data stored in RAM.

Pattern8RotationRam V1.3 *VDP_SPRITES*

```
void   Pattern8RotationRam (char pattern, char *SrcPattern, signed int rotation)
```

Rotates the 8x8 pixels sprite pattern n° ***pattern*** stored inside the RAM at address ***SrcPattern**. The new pattern is sent to the VRAM and replace the old one. Original pattern stored in RAM is not modified.
rotation indicates the angle, it can be
90 90° rotation to the right
-90 90° rotation to the left
180 180° rotation)

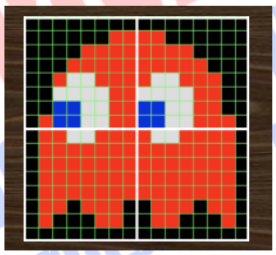


90° (right) rotation …

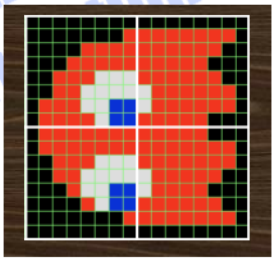
Pattern16RotationRam V1.3 *VDP_SPRITES*

```
void   Pattern16RotationRam (char pattern, char *SrcPattern, signed int rotation)
```

Rotates the 16x16 pixels sprite pattern n° ***pattern*** stored inside the RAM at address ***SrcPattern**. The new pattern is sent to the VRAM and replace the old one. The original pattern stored in RAM is not modified. ***rotation*** indicates the angle, it can be:
90 90° rotation to the right)
-90 90° rotation to the left)
180 180° rotation



-90° (left) rotation …



Pattern8FlipRam V1.3 *VDP_SPRITES*

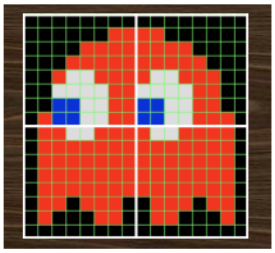
```
void   Pattern8FlipRam (char pattern, char *SrcPattern, char direction)
```

Flips the 8x8 pixels sprite pattern n° ***pattern*** stored inside the RAM at address ***SrcPattern**. The new pattern is sent to the VRAM and replace the old one. The original pattern stored in RAM is not modified.
direction indicates the direction of the flip:
0 horizontal flip
1 vertical flip

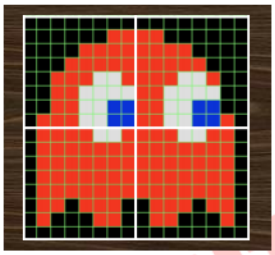
Pattern16FlipRam V1.3 *VDP_SPRITES*

```
void    Pattern16FlipRam (char pattern, char *SrcPattern, char direction)
```

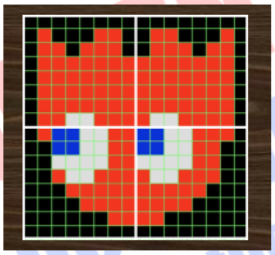
Flips the 16x16 pixels sprite pattern n° ***pattern*** stored inside the RAM at address ****SrcPattern***. The new pattern is sent to the VRAM and replace the old one. The original pattern stored in RAM is not modified.
direction indicates the direction of the flip:
0 horizontal flip
1 vertical flip



Horizontal flip …



Vertical flip …



Pattern8FlipVram V1.3 *VDP_SPRITES*

```
void    Pattern8FlipVram (char pattern, char direction, char DestPattern)
```

Flips the 8x8 pixels sprite pattern n° ***pattern*** stored inside the VRAM. The new flipped pattern can be copied over the same pattern number by setting ***DestPattern*** to **0**, or copied to another pattern location by setting ***DestPattern*** from **1** to **255**.
direction indicates the direction of the flip.
0 horizontal flip
1 vertical flip

Pattern16FlipVram V1.3 *VDP_SPRITES*

```
void    Pattern16FlipVram (char pattern, char direction, char DestPattern)
```

Flips the 16x16 pixels sprite pattern n° ***pattern*** stored inside the VRAM. The new flipped pattern can be copied over the same pattern number by setting ***DestPattern*** to **0**, or copied to another pattern location by setting ***DestPattern*** from **1** to **255**. (Do not forget that one 16x16 pattern is in fact four 8x8 patterns inside the pattern table).
direction indicates the direction of the flip.
0 horizontal flip
1 vertical flip

SpriteFollow	V1.3	*VDP_ SPRITES*

`void` **`SpriteFollow`** `(void *`**`SpriteStruct`**`)`

This function reads and decodes the compressed relative coordinates exported by the **Fusion-C** ***Sprite Path tool.***
First you must define a structure like this one :

```
typedef struct {
          char y;          // important Keep this order
          char x;
          char Data;
           // ...
} DEF_SPRITE;

static DEF_SPRITE sprite1;
sprite1.Data=path1[n];
SpriteFollow(&sprite1);
```

Second step you must send the coordinate data to the ***sprite1.Data***, and call the function with the address of the sprite structure as parameter.
The function will decode coordinates and set the new X and Y position of the sprite inside the structure‘s sprite variable
Please see the ***Sprite Path Editor***’s chapter, and the ***follow.c*** example.

CIRCLE

[vdp_circle.h]

Graphic functions to draw a circle on MSX graphic screen. Must be used with **VDP_GRAPH2.H** or **VDP_GRAPH1.H**

CircleFilled	MSX2	*VDP_CIRCLE*
`void CircleFilled (unsigned int x0, unsigned int y0, unsigned int radius, i unsigned nt color, char OP)`		
Draws a filled circle. Center of the circle ***at x0, y0***, with a radius ***radius.*** The color ***color*** and a logical operator mode ***OP***.		

Circle	MSX2	*VDP_CIRCLE*
`void Circle (unsigned int x0, unsigned int y0, unsigned int radius, unsigned int color, char OP)`		
Draws a circle. Center of the circle at ***x0, y0***, with a radius of ***radius.*** The color ***color*** and a logical operator ***OP***.		

SC2CircleFilled	V1.1	*VDP_CIRCLE*
`void SC2CircleFilled (char x0, char y0, char radius, char color)`		
Only for Screen 2 mode. Draws a filled circle. Center of the circle at ***x0***, ***y0***, with a radius ***radius***, and a color ***color***		

SC2Circle	V1.1	*VDP_CIRCLE*
`void SC2Circle (char x0, char y0, char radius, char color)`		
Only for Screen 2 mode. Draws a filled circle. Center of the circle at ***x0, y0***, with a radius ***radius***, and a color ***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 (char deviceId)
```

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

```
typedef struct {
    charslot;
     charnumber16KBSegments;
     charnumberFree16KBSegments;
     charnumberAllocatedSystem16KBSegments;
     charnumberUser16KBSegments;
     charnotInUse0;
     charnotInUse1;
      charnotInUse2;
} MAPPERINFOBLOCK;
```

Get_PN — *RAMMAPPER*

```
char    Get_PN (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 (char page, 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 segments.

AllocateSegment — *RAMMAPPER*

```
SEGMENTSTATUS   *AllocateSegment (char segmentType, char slotAddress)
```

Allocates next available 16Kbytes ram segment and returns information about this segment in the Status structure.

```
typedef struct {
     charallocatedSegmentNumber;
     charslotAddressOfMapper;
     charcarryFlag;
} 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 (char segmentType, char slotAddress)
```

Free an allocated segment

PSG

[PSG.H]

Extended function to control the sound processor.

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

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`** `(void)`	
Plays off all three PSG Channels.	

GetSound	***PSG***
`char` **`GetSound`** `(char` **`reg`**`)`	
Reads a PSG Register ***reg*** (0 to 13).	

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 has a volume of 16.
Envelope ***shape*** must be between **0** and **15**.

SoundFX — *PSG*

```
void    SoundFX (char channel, FX *sounddata)
```

Plays a FX by a PSG Channel (0,1 or 3). ***sounddata*** comes from this structure

```
typedef struct {
          boolean isTone;
          boolean isNoise;
          unsigned int Tone;
          char Noise;
          unsigned int Period;
          char Shape;
} FX;
```

AYFX PLAYER

[ayfx_player.h]

Use the AYFX sound editor to create and edit AYFX Sound. Save the sounds as a sound bank and placed the data in RAM.
Put the data in RAM. This can be done dynamically, by loading the bank from a disk, or directly as constant data in the body of the C program.
This new driver updated with Fusion-C can play up to 3 different sounds at the same time, on the 3 channels of the PSG. **This driver is not compatible with the simultaneous use of the PT3 Replayer** which has its own sound effects system.

InitFX	*AYFX_PLAYER*
`void InitFX (void *SndBankAddr)`	
Initialization of the ayFX player. ***SndBankAddress*** must point to the AYFX Sound Bank stored in Ram.	

PlayFX	*AYFX_PLAYER*
`char PlayFX (char SoundFX)`	
Plays the sound n° ***SoundFX*** from the sound bank.	

UpdateFX	*AYFX_PLAYER*
`void UpdateFX (void)`	
Currently playing the sounds. This function must be part of the main loop of your program. It updates the PSG registers, with sound FX that must be played.	

StopFX	*AYFX_PLAYER*
`char StopFX (void)`	
Stops playing all sounds.	

MUSIC PT3 + AYFX REPLAYER

[pt3replayer.h]

Use those function to play PT3 music files, and SoundFX from an AYFX Sound Bank.
Since Fusion-C v1.3 , the PT3 Replayer can now also reproduce FX sounds from an AYFX Sound Bank. If you wish to play music, and also sound FX in the same time, you must use these functions.

PT3Init	***PT3REPLAYER***
`void PT3Init (char *SongAddr, char Loop)`	
Initialization of the PT3 replayer. ****SongAddr*** must be set with the PT3 data area in RAM. ***Loop*** must be set to : **0** if you do not want the music loop **1** if you want the music loop continuously	

PT3Play	***PT3REPLAYER***
`void PT3Play (void)`	
Actualizes the music playing, inside a main loop. This function must be executed on each interruption of VBLANK.	

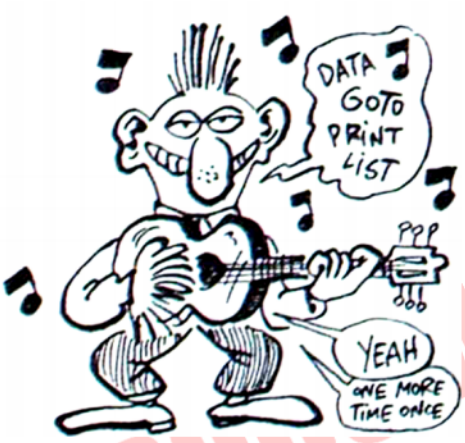
PT3Rout	***PT3REPLAYER***
`void PT3Rout (void)`	
Prepares data to be played by the sound processor. This function must be executed on each interruption of VBLANK.	

PT3Mute	***PT3REPLAYER***
`void PT3Mute (void)`	
Mutes the music. To mute totally the Music, you must, first Call this function, and stop invoking PT3Play function.	

PT3FXInit	***PT3REPLAYER***
`void PT3FXInit (void *BankAddr, char Channel)`	
Initialisation of the AYFX sound process. ****BankAddr*** must be set with the address of the AYFX Bank in Ram. ***Channel*** is the default PSG's channel used to play sound FX, it can be **0**, **1** or **2**	

PT3FXPLay	***PT3REPLAYER***
`void PT3FXPlay (char Sound, char Priority)`	
Plays a souynd FX from the AYFX sound Bank. ***Sound*** represent the sound number from the bank, from **0** to **255**. ***Priority***, represents the sound priority of the sound, from **0** to **15**. Highest priority is **0**, lowest priority is **15**. A sound with high priority will be played over a low priority sound even If it hasn't finished playing.	

PT3FXRout	*PT3REPLAYER*
`void` **`PT3FXRout`** `(void)`	
Sends the sound to the sound processor. This function must be executed on each interruption of VBLANK.	



FUSION-C ENVIRONMENT VARIABLES

When you are Including "msx_fusion.h" in your code, it gives you access to some environment variables. You can read them anywhere in your code, without any previous declaration.

_SpriteOn	V1.3	
_SpriteOn, returns **0** when sprites are activated, **1** when sprites are deactivated. (8-bit value)		

_SpriteSize	V1.3	
_SpriteSize, returns **0** when sprite size is 8x8, **1** when size is 16x16. (8-bit value)		

_SpriteMag	V1.3	
_SpriteMag, returns **0** when sprite size is normal, **1** when sprites are double. (8-bit value)		

_DisplayPage	V1.3 MSX2	
_DisplayPage, returns the displayed VRAM Page, as a ***char*** value from **0** to **3**.		

_ActivePage	V1.3 MSX2	
_ActivePage, returns the active VRAM Page, as a ***char*** value from **0** to **3**.		

_VDPfreq	V1.3 MSX2	
_VDPfreq, returns **1** if VDP frequency is 50 Hz, **0** if VDP frequency is 60 Hz		

_VDPlines	V1.3 MSX2	
_VDPlines, returns **1** if VDP is set to 212 lines, **0** if VDP is set to 192 lines. (8-bit value)		

_ForegroundColor	V1.3	
_ForegroundColor, returns the foreground color, as a ***char*** value from **0** to **255**.		

_BackgroundColor	V1.3	
_BackgroundColor, returns the background color, as a ***char*** value from **0** to **255**.		

_BorderColor	V1.3	
_BorderColor, returns border color, as a ***char*** value from **0** to **255**.		

_ScreenMode V1.3

_ScreenMode, returns the current screen mode, as a ***char*** value from **0** to **12**.

_SpritePatternAddr V1.3

_SpritePatternAddr, returns the default Sprite Pattern table address in Vram for the current screen mode, as an ***unsigned int*** value.

_SpriteAttribAddr V1.3

_SpriteAttribAddr, returns the default Sprite Attributs table address in Vram for the current screen mode, as an ***unsigned int*** value.

_SpriteColorAddr V1.3 MSX2

_SpriteColorAddr, returns the default Sprite Color table address in Vram for the current screen mode, as an ***unsigned int*** value.

_WidthScreen0 V1.3 MSX2

_WidthScreen0, returns the actual width of screen mode 0, as an ***unsigned char*** value.

_WidthScreen1 V1.3 MSX2

_WidthScreen1, returns the actual width of screen mode 1, as an ***unsigned char*** value.

_Time V1.3 MSX2

_Time, returns the actual MSX internal timer as an ***unsigned Int*** value.

_FusionVer V1.3

_FusionVer, returns the current Fusion-C library version, as ***char*** value.
(Divide the returned value by 10 to obtain the real version number)

_FusionRev V1.3

_FusionRev, returns the current Fusion-C library revision date, as an ***unsigned int*** value.
The 5 digits represent the date coded like this:

Number of Year after 2019	Day of revision	Month of the revision
Y	DD	MM

MSX BASIC VS Fusion-C
Instructions comparison

This part is just a quick comparison between MSX BASIC commands and possible C commands you can find in libraries.
Comparison between the two languages is a not really fair because programming in C has nothing to do with programming in MSX Basic. Anyway, it can help beginners to find some usual information.
Just think C libraries are offering much, much more possibilities and easy-to-use functions than Basic.

Jump and Loop	

FOR … NEXT	For (int ; condition ; increment){ *statement…;* }
	while (condition) {*statement …;*}
	do {*statement*…; } while (condition) ;

GOSUB	See GOTO
GOTO	Goto LABEL; … LABEL:
RETURN	Return (n); Return a value at the end of a function

Clock and Time	
INTERVAL	See Interrupt fonctions
GET DATE	GetDate
GET TIME	GetTime
ON INTERVAL GOSUB	SetInterrupt, SetVDPInterrupt
SET DATE	SetDate
SET TIME	SetTime
TIME	RealTimer, SetRealTimer

Conditions	
IF .. THEN … ELSE	If, else
ON .. GOSUB	Switch
ON .. GOTO	Switch

Conversions	
ASC()	Itoa
BIN$()	Use 0b11111111 format
CDBL()	IntToFloat()
CHR$()	Replace print chr$(x) by PrintChar
CINT()	(int)
CSNG()	IsPositive
HEX$()	Use 0xFFFF
VAL()	Atoi

Loading and Saving	
BLOAD	N/A
BSAVE	N/A
CLOAD	N/A
CSAVE	N/A
LOAD	Read
SAVE	Write
CLOSE	Close
OPEN	Open
MERGE	See Read with append mode
RUN	N/A

Graphics and Screen

BASE	
CIRCLE	Circle
CLS	Cls
COLOR	SetColors
COPY	HMMC, LMMC, HMCM, LMMM, HMMM, YMMM
DRAW	Draw
LINE	Line
LOCATE	Locate
WIDTH	Width
PAINT	Paint
POINT	Point
PSET	Pset
POS()	
PRINT	Print, Printf or PrintString …
SCREEN	Screen
SET PAGE	SetActivePage, SetDisplayPage
SET SCROLL	SetScrollH, SetScrollV
SET VIDEO	N/A
SET ADJUST	SetAdjust
VDP()	VDPwrite
VPEEK	Vpeek
VPOKE	Vpoke
INP()	Inport
OUT	Outport

KeyBoard and Controls	
INKEY$	Inkey
INPUT	InputChar, inputString
INPUT$()	WaitKey
KEY	Inkey
KEY()	
LINE INPUT	N/A
ONB KEY GOSUB	Switch
ON STOP GOSUB	
ON STRIG GOSUB	
PAD()	MouseRead, MouseReadTo
PDL()	
STICK()	JoystickRead, JoystickReadTo
STRIG()	TriggerRead

Math	
ABS	fabsf
CDBL	IntToFloat
CINT()	
CSNG	
EXP()	powf
FIX()	
INT()	ceilf, floor
LOG()	logf
RND()	rand
SGN()	IsPositive
SQR()	sqrtf
ATN()	atanf
COS()	cosf
SIN()	sinf
TAN()	tanf

Ram Access	
PEEK	Peek, PeekW
POKE	Poke, PokeW

Sounds	
PLAY	PSGwrite
SOUND	PSGwrite
BEEP	Beep
CALL PCMPLAY	PCMPlay

Sprites	
COLOR SPRITE()	SpriteColors
COLOR SPRITE$()	SpriteColors
ON SPRITE GOSUB	SpriteCollision
PUT SPRITE	PutSprite, fPutSprite
SPRITE	N/A
SPRITE$()	SetSpritePattern

Strings	
INSTR()	StrPosStr
LEFT$	StrLeftTrim
MID$	
RIGHT$	StrRightTrim
SPACE$	
STRING$	

Variables settings	
CLEAR	N/A
DATA	N/A
DIM	Malloc
ERASE	Free
LET	N/A
READ	N/A
RESTORE	N/A
SWAP	IntSwap, StrReverse

The Library's source code

The FUSION-C Library is provided with all source codes you can find all files in the folder «**Fusion-c/Lib/Sources/** ».
There are two types of files. The « **.s** » files are source files in assembler ; the **« .c** » are source files in C.
You are free to modify any file you want, and add your own code, and even complete the library with your own routines and functions.
To add functions to FUSION-C, just add your source file in the « **source/lib** » folder, and launch the compilation script: » **_build_lib.sh** » or « **_build_lib.bat** » if you are using Windows. The compilation script will build a new library, and copy it at the right place. In case of error, you will be warmed, and details of the error can be read in the log file.
Do not forget to add your function to the appropriate header file ***(.h)****.*

Please, share your work. Your needs can be the needs of other coders, so if you do modifications or add functions, send us your files and documentation, we will include your work in an upcoming version of the FUSION-C Library.

For your information, here the instruction used to compile assembler source file:

```
> sdasz80 -o <filename.s>
```

This instruction is used to compile C source file:

```
> sdcc –use-stdout -mz80 -c  <filename.c>
```

The compilator will generate a **« .rel** » with the same name as the original source code filename.
This instruction is used to include a compiled source code into the library:

```
> sdar -rc  fusion.lib <filename>.rel
```

The Source code catalog

If you need to modify or just look how a function is coded, this catalog will help you identify in which file you can find the source code of a function.

FUSION-C 1.3	R10106		
Function 's name	**Language**	**Source file**	**Computer**
AllocateSegment	ASM	**rammapper.s**	
Beep	ASM	**CallBios_Functions.s**	
BitReset	C	**bit.c**	
BitReturn	C	**bit.c**	
BoxFill	C	**vdp_graph2plus.c**	MSX2
BoxLine	C	**vdp_graph2plus.c**	MSX2
CallBios	ASM	**call.c**	
CallDos	ASM	**call.c**	
CallSub	ASM	**call.c**	MSX2
ChangeCap	ASM	**CallBios_Functions.s**	
ChangeCPU	ASM	**CallBios_Functions.s**	Turbo-R
ChangeDir	ASM	**io.s**	
CharToLower	ASM	**ctype.s**	
CharToUpper	ASM	**ctype.s**	
CheckBreak	ASM	**printhex.s**	
Circle	C	**circle.c**	MSX2
CircleFilled	C	**circle.c**	MSX2
Clear1px	ASM	**vdp_graph1.s**	
Clear8px	ASM	**vdp_graph1.s**	
Close	ASM	**io.s**	
Cls	ASM	**CallBios_Functions.s**	
CopyRamToVram	ASM	**Vram.s**	
CopyVramToRam	ASM	**Vram.s**	
CovoxPlayRam	ASM	**covoxplay.c**	
CovoxPlayVram	C & ASM	**covoxplay.c**	MSX2
Create	ASM	**io.s**	
CreateAttrib	ASM	**io.s**	
DisableInterrupt	#define	**msx_fusion.h**	
DiskLoad	ASM	**io.s**	
Draw	ASM	**vdp_graph2.s**	MSX2
EnableInterrupt	#define	**msx_fusion.h**	
EndInterruptHandler	ASM	**interrupt.s**	
EndVDPInterruptHandler	ASM	**interrupt_vdp.s**	
Exit	ASM	**CallDos_Functions.s**	
FcbClose	ASM	**Fcb_access.s**	
FcbCreate	ASM	**Fcb_access.s**	
FcbDelete	ASM	**fcb_access.s**	
FcbFindFirst	ASM	**Fcb_access.s**	
FcbFindNext	ASM	**Fcb_access.s**	
FCBlist	ASM	**Fcb_access.s**	
FcbOpen	ASM	**Fcb_access.s**	
FcbRead	ASM	**Fcb_access.s**	
FcbWrite	ASM	**Fcb_access.s**	
FillVram	ASM	**CallBios_Functions.s**	
FindFirst	ASM	**io.s**	
FindNext	ASM	**io.s**	
Fkeys	ASM	**keyboardread.s**	
fLMMM	ASM	**vdp_graph2.s**	MSX2
fPutSprite	ASM	**vram.s**	
FreeSegment	ASM	**rammapper.s**	
FunctionKeys	ASM	**callBios_Functions.s**	

fVDP	ASM	**vdp_graph2.s**	MSX2
Get_PN	ASM	**rammapper.s**	
Get1px	ASM	**vdp_graph1.s**	
Get8px	ASM	**vdp_graph1.s**	
Getche	ASM	**getche.s**	
GetCol8px	ASM	**vdp_graph1.s**	
GetCPU	ASM	**CallBios_Functions.s**	Turbo-R
GetCWD	ASM	**io.s**	
GetDate	ASM	**Calldos_Functions.s**	
GetDisk	ASM	**CallDos_Functions.s**	
GetDiskParam	ASM	**CallDos_Functions.s**	
GetDiskTrAddress	ASM	**CallDos_Functions.s**	
GetKeyMatrix	C	**msx_fusion.h**	
GetOSVersion	ASM	**io.s**	
GetSound	ASM	**psg.c**	
GetTime	ASM	**CallDos_Functions.s**	
GetVramSize	ASM	**Vram.s**	
Halt	#define	**msx_fusion.h**	
HideDisplay	ASM	**CallBios_Functions.s**	
HMCM	ASM	**vdp_graph2.s**	MSX2
HMCM_SC8	ASM	**vdp_graph2.s**	MSX2
HMMC	ASM	**vdp_graph2.s**	MSX2
HMMM	ASM	**vdp_graph2.s**	MSX2
HMMV	ASM	**vdp_graph2.s**	MSX2
InitFX	C	**ayfxDriver.s**	
InitInterruptHandler	ASM	**interrupt.s**	
InitPSG	ASM	**CallBios_Functions.s**	
InitRamMapperInfo	ASM	**rammapper.s**	
Inkey	ASM	**callBios_Functions.s**	
InPort	ASM	**port_in-out.s**	
InputChar	ASM	**callBios_Functions.s**	
InputString	ASM	**inputstring.s**	
IntSwap	C	**intswap.c**	
IntToFloat	C	**inttofloat.c**	
IsAlpha	ASM	**ctype.s**	
IsAlphaNum	ASM	**ctype.s**	
IsAscii	ASM	**ctype.s**	
IsCtrl	ASM	**ctype.s**	
IsDigit	ASM	**ctype.s**	
IsGraph	ASM	**ctype.s**	
IsHexDigit	ASM	**ctype.s**	
IsHsync	#define	**msx_fusion.h**	
IsLower	ASM	**ctype.s**	
IsPositive	C	**ispositive.c**	
IsPrintable	ASM	**ctype.s**	
IsPunctuation	ASM	**ctype.s**	
IsSpace	ASM	**ctype.s**	
IsUpper	ASM	**ctype.s**	
IsVsync	#define	**msx_fusion.h**	
Itoa	C	**itoa.c**	
JoystickRead	ASM	**CallBios_Functions.s**	
JoystickReadTo	ASM	**Joystick_readTo.s**	
KeySound	#define	**msx_fusion.h**	
KillKeyBuffer	ASM	**CallBios_Functions.s**	
Line	C	**vdp_graph2plus.c**	MSX2
LMMC	ASM	**vdp_graph2.s**	MSX2
LMMM	ASM	**vdp_graph2.s**	MSX2
LMMV	ASM	**vdp_graph2.s**	MSX2

Locate	ASM	**CallBios_Functions.s**	
Lseek	ASM	**io.s**	
Ltell	ASM	**io.s**	
MakeDir	ASM	**io.s**	
MemChr	ASM	**memchr.s**	
MemCompare	ASM	**memcompare.s**	
MemCopy	ASM	**memcopy.s**	
MemCopyReverse	ASM	**memcopyreverse.s**	
MemFill	ASM	**memfill.s**	
MMalloc	C	**mmalloc.c**	
MouseRead	ASM	**mouseread.c**	
MouseReadTo	ASM	**mousereadto.c**	
NStrCompare	ASM	**nstrcompare.s**	
NStrConcat	ASM	**nstrconcat.s**	
NStrCopy	ASM	**nstrcopy.s**	
Open	ASM	**io.s**	
OpenAttrib	ASM	**io.s**	
OutPort	ASM	**port_in-out.s**	
OutPorts	ASM	**port_in-out.s**	
Paint	ASM & C	**vdp_paint.h**	MSX2
Pattern16FlipRam	C	**PatternTransform.c**	
Pattern16FlipVram	C	**PatternTransform.c**	
Pattern16RotationRam	C	**PatternTransform.c**	
Pattern16RotationVram	C	**PatternTransform.c**	
Pattern8FlipRam	C	**PatternTransform.c**	
Pattern8FlipVram	C	**PatternTransform.c**	
Pattern8RotationRam	C	**PatternTransform.c**	
Pattern8RotationVram	C	**PatternTransform.c**	
PatternHFlip	ASM	**PatternTransform.c**	
PatternRotation	ASM	**PatternTransform.c**	
PatternVFlip	ASM	**PatternTransform.c**	
PCMPlay	ASM	**CallBios_Functions.s**	Turbo-R
Peek	#define	**msx_fusion.h**	
Peekw	#define	**msx_fusion.h**	
PlayEnvelope	C	**psg.c**	
PlayFX	ASM	**ayfxDriver.s**	
Point	ASM	**vdp_graph2.s**	MSX2
Poke	#define	**msx_fusion.h**	
Pokew	#define	**msx_fusion.h**	
Polygon	C	**vdp_graph1plus.c**	MSX2
Print	C	**print.c**	
PrintChar	ASM	**callBios_Functions.s**	
PrintDec	ASM	**printdec.s**	
printf	C	**printf-msx.c**	
PrintFNumber	C	**printfnumber.c**	
PrintHex	ASM	**printhex.s**	
PrintNumber	C	**printnumber.c**	
Pset	ASM	**vdp_graph2.s**	MSX2
PSGread	C	**psg.c**	
PSGwrite	C	**psg.c**	
PT3FXInit	ASM	**pt3replayer.s**	
PT3FXPlay	ASM	**pt3replayer.s**	
PT3FXRout	ASM	**pt3replayer.s**	
PT3Init	ASM	**pt3replayer.s**	
PT3Mute	ASM	**pt3replayer.s**	
PT3Play	ASM	**pt3replayer.s**	
PT3Rout	ASM	**pt3replayer.s**	
Put_PN	ASM	**rammapper.s**	

PutCharHex	ASM	printhex.s	
PutSprite	ASM	Vram.s	
PutText	ASM	CallBios_Functions.s	
Read	ASM	io.s	
ReadAdjust	C	vpoke-vpeek.c	MSX2
ReadBlock	ASM	sc2block.s	
ReadKeyboardType	ASM	callBios_Functions.s	
ReadMSXtype	C	readmsxtype.c	
ReadSP	ASM	readsp.s	
ReadTPA	#define	msx_fusion.h	
RealTimer	#define	msx_fusion.h	
Remove	ASM	io.s	
RemoveDir	ASM	io.s	
Rename	ASM	io.s	
RestorePalette	ASM	Set_Palette.s	MSX2
Rkeys	ASM	keyboardread.s	
RleWBToRam	ASM	RLEwb_toram.c	
RleWBToVram	C & ASM	RLEwb_tovram.c	MSX2
SaveScreenBoot	C	call.c	MSX2
SC2BoxFill	C	vdp_graph1plus.c	
Sc2BoxLine	C	vdp_graph1plus.c	
SC2Circle	C	circle.c	
SC2CircleFilled	C	circle.c	
SC2Draw	ASM	vdp_graph1.s	
SC2Line	C	vdp_graph1plus.c	
SC2Paint	ASM	vdp_graph1.s	
SC2Point	ASM	vdp_graph1plus.c	
SC2Pset	ASM	vdp_graph1plus.c	
SC2ReadScr	ASM	readwritescr.s	
SC2WriteScr	ASM	readwritescr.s	
Screen	ASM	CallBios_Functions.s	
SectorRead	ASM	CallDos_Functions.s	
SectorWrite	ASM	CallDos_Functions.s	
Set1px	ASM	vdp_graph1.s	
Set8px	ASM	vdp_graph1.s	
SetActivePage	#define	msx_fusion.h	
SetAdjust	C	vpoke-vpeek.c	MSX2
SetBorderColor	ASM	setbordercolor.s	
SetChannel	ASM	psg.c	
SetCol8px	ASM	vdp_graph1.s	
SetColor	C	msx_fusion.h	
SetColorPalette	ASM	Set_Palette.s	MSX2
SetColors	ASM	CallBios_Functions.s	
SetDate	ASM	CallDos_Functions.s	
SetDisk	ASM	CallDos_Functions.s	
SetDiskTrAddress	ASM	CallDos_Functions.s	
SetDisplayPage	ASM	VDPWrite_functions.s	
SetEnvelopePeriod	C	psg.c	
SetExpandVDPcmd	ASM	VDPWrite_functions.s	MSX2+
SetInterruptHandler	ASM	interrupt.s	
SetNoisePeriod	C	psg.c	
SetPaintBuffer	C	vdp_paint.h	MSX2
SetPalette	ASM	Set_Palette.s	MSX2
SetRamDisk	ASM	callBios_Functions.s	
SetRealTimer	#define	msx_fusion.h	MSX2
SetScreen10	ASM	VDPWrite_functions.s	MSX2+
SetScreen12	ASM	VDPWrite_functions.s	MSX2+
SetScrollDouble	ASM	VDPWrite_functions.s	MSX2+

SetScrollH	ASM	VDPWrite_functions.s	MSX2+
SetScrollMask	ASM	VDPWrite_functions.s	MSX2+
SetScrollV	ASM	VDPWrite_functions.s	MSX2
SetSpriteColors	C	setspritepattern.c	
SetSpritePattern	C	setspritepattern.c	
SetTime	ASM	CallDos_Functions.s	
SetTonePeriod	C	psg.c	
SetTransparent	ASM	VDPWrite_functions.s	MSX2
SetVDPInterruptHandler	ASM	interrupt_vdp.s	
SetVDPread	ASM	Vram.s	
SetVDPwrite	ASM	Vram.s	
SetVolume	C	psg.c	
ShowDisplay	ASM	CallBios_Functions.s	
SilencePSG	C	psg.c	
Sound	ASM	psg.c	
SoundFX	C	psg.c	
Sprite16	ASM	VDPWrite_functions.s	
Sprite32Bytes	ASM	sprite32bytes.s	
Sprite8	ASM	VDPWrite_functions.s	
SpriteCollision	#define	vdp_sprites.h	
SpriteCollisionX	C	spritecollision.c	MSX2
SpriteCollisionY	C	spritecollision.c	MSX2
SpriteDouble	ASM	VDPWrite_functions.s	
SpriteFollow	ASM	spritefollow.c	
SpriteOff	ASM	VDPWrite_functions.s	
SpriteOn	ASM	VDPWrite_functions.s	
SpriteOverLap	#define	VDP_sprites.h	
SpriteOverLapId	#define	VDP_sprites.h	
SpriteReset	ASM	CallBios_Functions.s	
SpriteSmall	ASM	VDPWrite_functions.s	
StopFX	ASM	ayfxDriver.s	
StrChr	ASM	strchr.s	
StrCompare	ASM	strcompare.s	
StrConcat	ASM	strconcat.s	
StrCopy	ASM	strcopy.s	
StrLeftTrim	ASM	strlefttrim.s	
StrLen	ASM	strlefttrim.s	
StrPosChr	ASM	strposchr.s	
StrPosStr	ASM	strposstr.s	
StrReplaceChar	ASM	strreplacechar.s	
StrReverse	C	strreverse.c	
StrRightTrim	ASM	strrighttrim.s	
StrSearch	ASM	strsearch.s	
StrToLower	ASM	ctype.s	
StrToUpper	ASM	ctype.s	
Suspend	ASM	msx_fusion.h	
TriggerRead	ASM	CallBios_Functions.s	
TurboMode	C	vpoke-vpeek.c	
UpdateFX	ASM	ayfxDriver.s	
VDP50Hz	ASM	VDPWrite_functions.s	MSX2
VDP60Hz	ASM	VDPWrite_functions.s	MSX2
VDPalternate	C	vpoke-vpeek.c	MSX2
VDPinterlace	C	vpoke-vpeek.c	MSX2
VDPLINE	ASM	vdp_graph2.s	MSX2
VDPlineSwitch	ASM	VDPWrite_functions.s	MSX2
VDPstatus	ASM	vdpstatus.s	
VDPstatusNi	ASM	vdpstatus.s	
VDPwrite	ASM	VDPWrite_functions.s	

VDPwriteNi	ASM	**VDPWrite_functions.s**	
vMSX	ASM	**vdp_graph2.s**	
Vpeek	**C**	**vpoke-vpeek.c**	
VpeekFirst	#define	**msx_fusion.h**	
VpeekNext	#define	**msx_fusion.h**	
Vpoke	C	**vpoke-vpeek.c**	
VpokeFirst	#	**msx_fusion.h**	
VpokeNext	#define	**msx_fusion.h**	
Vsynch	#define	**msx_fusion.h**	
WaitKey	ASM	**CallBios_Functions.s**	
Width	C	**vpoke-vpeek.c**	
Write	ASM	**io.s**	
WriteBlock	ASM	**sc2block.s**	
YMMM	ASM	**vdp_graph2.s**	MSX2

Adding Assembler source code inside your C program

When writing code to be compiled with SDCC targeting Z80, assembler code fragments can be inserted in the C functions by enclosing them between the « **__asm** » and « **__endasm** » tags:

```
void DoNotDisturb()
{
  __asm
  di
  __endasm;

  DoSomething();

  __asm
  ei
  halt
    endasm;
}
```

Adding « **__naked** » to the function definition will cause the compiler to not generate the ret statement at the end of the function, you will usually use it when the entire body of the function is written in assembler:

```
void TerminateProgram() __naked
{
  __asm
  ld c,#0
  jp #5
  __endasm;
}
```

Assembler code must preserve the value of register IX, all other registers can be used freely.

The Z80 assembler supplied with SDCC uses a pretty much standard syntax for the assembler source code except for the following:
-Decimal numeric constants must be preceded with #
-Hexadecimal numeric constants must be preceded with **#0x**
-The syntax for offsets when using index registers is **n(ix),** where in other assemblers it's usually **(ix+n)**

The return value of a C function is passed to the caller as follows:

-Functions that return **char,** the value goes in the **L** register.
-Functions that return **int** or a pointer the value goes in the **HL** registers.
-Functions that return **long** the value goes in the **DEHL** registers.

```
char GetMagicNumber() __naked
{
  __asm
  ld l,#34
  ret
  __endasm;
}

int GetMagicYear() __naked
{
  __asm
  ld hl,#1987
  ret
  __endasm;
}

long GetReallyStrongPassword() __naked
{
  __asm
  ld de,#0x1234
  ld hl,#0x5678
  ret
  __endasm;
}
```

Functions parameters are pushed to the stack before the function is called. They are pushed from right to left, so the leftmost parameter is the first one found when going up in the stack:

```
char SumTwoChars(char x, char y) __naked
{
    __asm
    ld iy,#2
    add iy,sp ;Bypass the return address of the function.

    ld l,(iy)   ;x
    ld a,1(iy)  ;y

    add l
    ld l,a      ;return value

    ret
    __endasm;
}
```

```
int SumCharAndInt(char x, int y) __naked
{
    __asm
    ld iy,#2
    add iy,sp

    ld e,(iy)   ;x
    ld d,#0

    ld l,1(iy)  ;y (low)
    ld h,2(iy)  ;y (high)

    add hl,de   ;return value

    ret
    __endasm;
}

long SumCharIntAndLong(char x, int y, long z) __naked
{
    __asm
    ld iy,#2
    add iy,sp

    ld c,(iy)   ;x
    ld b,#0
    ld l,1(iy)  ;y (low)
    ld h,2(iy)  ;y (high)
    add hl,bc   ;x+y

    ld a,l
    add 3(iy)   ;z (lower)
    ld l,a
    ld a,h
    adc 4(iy)
    ld h,a
    ld a,#0
    adc 5(iy)
    ld e,a
    ld a,#0
    adc 6(iy)   ;z (higher)
    ld d,a

    ret     ;return value = DEHL
    __endasm;
}
```

It is possible to call other C functions from assembler code. Just push the parameters that the function expects, call the function assembler name (it's the C name prepended with « _ », pop the parameters to restore the stack state, and act on the return value as appropriate:

```
int SumTwo(int x, int y)
{
    return x+y;
}

int SumThree(int x, int y, int z) __naked
{
    __asm

    ld iy,#2
    add iy,sp

    ld l,2(iy)
    ld h,3(iy)
    push hl     ;y for SumTwo
    ld l,(iy)
    ld h,1(iy)
    push hl     ;x for SumTwo

    call _SumTwo  ;Return value in HL

    pop af      ;x
    pop af      ;y

    ld iy,#2
    add iy,sp

    ld e,4(iy)
    ld d,5(iy)  ;z
    add hl,de

    ret

    __endasm;
}
```

It is possible to define pure assembler functions intended to be called exclusively from assembler code. In this case the standard parameter passing rules can be overridden:

```
//DO NOT call this function from C code!
int SumHLDE() __naked
{
    __asm
    add hl,de
    ret
    __endasm;
}

int SumThree(int x, int y, int z) __naked
{
    __asm

    ld iy,#2
    add iy,sp

    ld l,2(iy)
    ld h,3(iy)
    ld e,(iy)
    ld d,1(iy)

    call _SumHLDE

    ld e,4(iy)
    ld d,5(iy)  ;z
    add hl,de

    ret

    __endasm;
}
```


XYZOLOG
DATA RECORDER
DR-202A
PHASE
SAVE
LOAD
REW
F FWD
STOP
EJECT

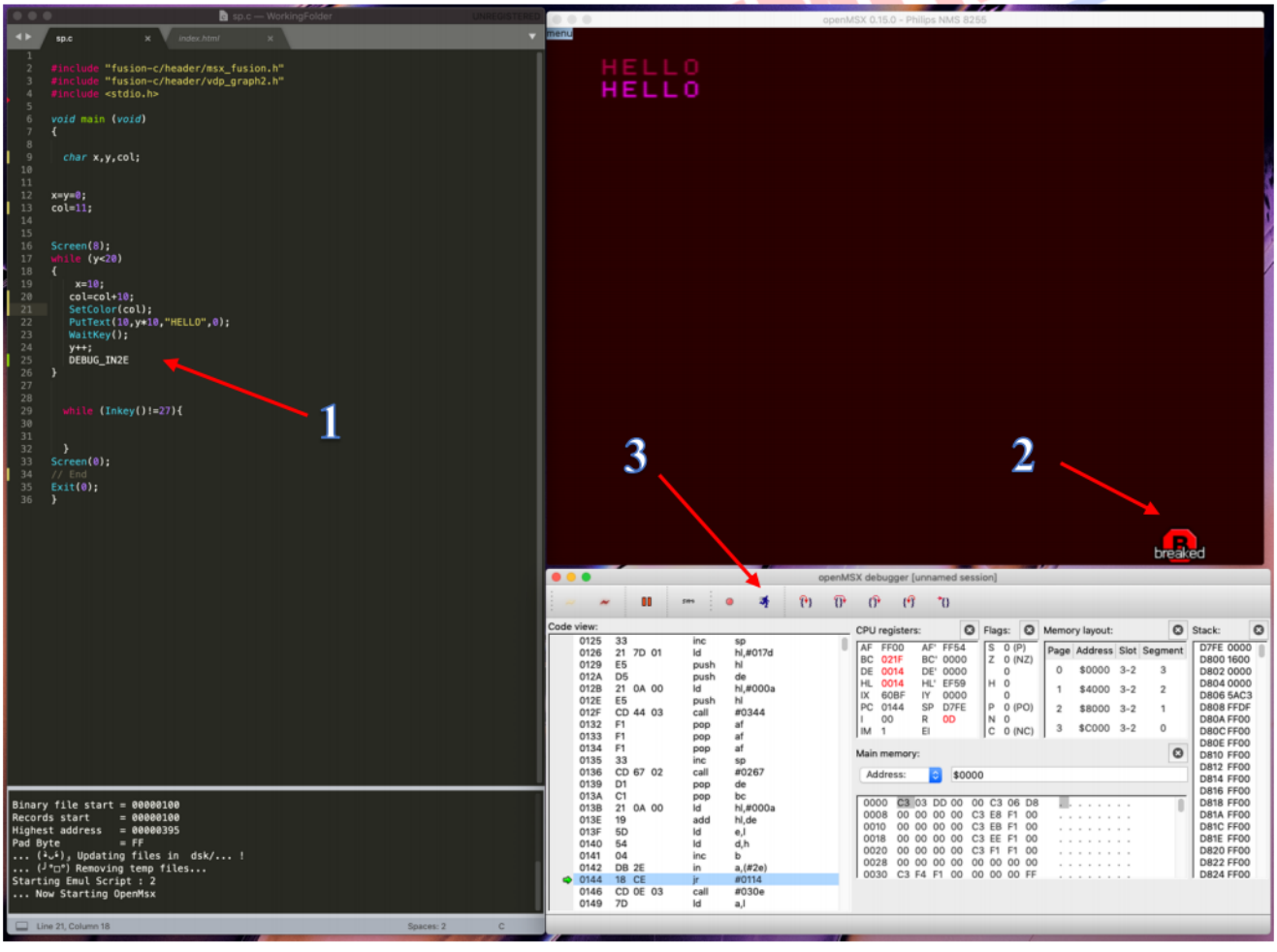
Debugging for advanced users

If you have enough knowledge about MSX and assembler programming, you can use the openMSX Debugger to find problems, or to improve your C program. If you think you are not strong enough, make a little try, take a look to the debugger, it will certainly be useful for you too.

Note for MacOS users: You will need to install QT library before using the debugger. Open your Shell/Terminal windows and install the QT library with this command:

```
> Brew install qt5
```

Launch the debugger which is in **./WorkingFolder/openMSX/ then** connect it to the active openMSX session, and you will see in live everything that happens in the MSX memory, in the registers as well as in the VRAM.
Now it may be useful to break the program you are coding at a specific place. For that, FUSION-C has a special directive. By adding this command inside your source code: **DEBUG_IN2E** you are indicating to openMSX that it must stop the program, and wait for a manual intervention to continue.



We are not going to detail here how the debugger works, because that goes beyond our subject. Let just check theses 3 points:
1 - The stop command in the source code
2- The indication in openMSX that the program is paused
3- openMSX Debugger. To resume program execution, click on RUN

Use command line arguments with your program

It is possible to pass some values from the command line to your C programs when they are executed. These values are called **command line arguments** and many times they are important for your program especially when you want to control your program from outside instead of hard-coding those values inside the code.
The command line arguments are handled using « main () » function arguments where **arc** refers to the number of arguments given, and « **argv []** » is a pointer array which points to each argument given to the program. Following is a simple example which checks if there is any argument supplied from the command line and take action accordingly.

```
#include <stdio.h>
int main( char *argv[], int argc )
{
  if( argc == 1 )
  {
    printf("The argument supplied is %s\n", argv[0]);
  } else if( argc > 1 ) {
     printf("Too many arguments supplied.\n");
   } else {
    printf("One argument expected.\n");
  }
}
```

When the above code is compiled and executed with a single argument, it produces the following result.

```
> a.com testing
The argument supplied is testing
```

When the above code is compiled and executed with two arguments, it produces the following result.

```
> a.com testing1 testing2
Too many arguments supplied.
```

When the above code is compiled and executed without passing any argument, it produces the following result.

```
> a.com
One argument expected
```

It should be noted that « **argv [0]** » is a pointer to the first command line argument supplied, and *argv[n] is the last argument. If no arguments are supplied, argc will be 0.

Important notice about compilation of programs that need arguments.
To be able to use arguments with your MSX-DOS program you must compile your program with an extended version of the crt0 startup code. You have two possibilities to do that…

1 ***- Use the Overriding compilation directives***

Add these 2 lines at the top of your C listing.

```
#define __SDK_CRT0__  rt0_msxdos_advanced.rel
#define __SDK_ADDRCODE__ 0x180
```

or

2 ***- Modify the compilation script, "build.bat" or "build.sh".***

look for the lines:

```
  # -- Default CRT0 ton use
DEFAULT_CRT0="${INCLUDE_DIR}crt0_msxdos.rel"
```

Change the text value "**crt0_msxdos.rel"** by "**crt0_msxdos_advanced.rel"**

```
  # -- Default Code Address (sdcc parameter)
DEFAULT_ADDR_CODE="0x106"
```

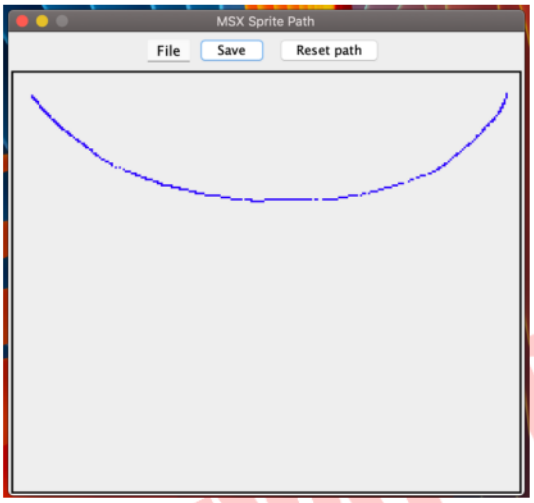
Change the text value "**0x106"** by "**0x180"**

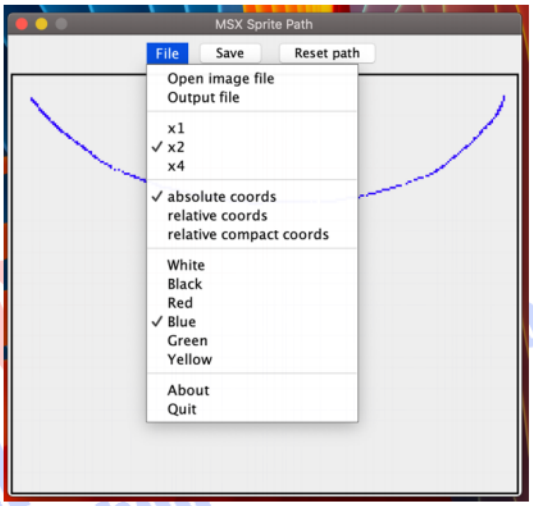
Fusion-C - Tools

Fusion-c comes with several tools that can help you create games. Two of them were specially developed by Sylvain Cregut for Fusion-C. These are **Sprite Path Editor**, and **Image to Sprite Editor**. You will find these 2 tools in .JAR (Java) format in the Tools folder. To be able to use these tools, you must first install Java Runtime on your computer.

Sprite Path

Sprite Path is a tool that lets you draw paths, or routes and retrieve the coordinates of each point drawn, in the order in which they were drawn.





Draw your Path with the left mouse button. The right mouse button let you go back .***Open Image file***: Let you load a 256x212 image in the background of the windows.
OutPut File: Let you choose the folder where you want to save the output file.
X1, X2, X4: let you zoom in the Window
Absolute cords: Is the default choice. It will save the coordinates of each point like they are on the MSX's screen, for example (10,250) or (185,200)

Sample output file: (The last number is the total number of coordinates)

```
{99,26,99,27,100,27,101,27,102,27,103,27,104,27,105,27,106,28,107,28}
10
```

Relative Coords : Will save the coordinates of each point, according to the coordinates of the previous point. For example, if the first point is (100,100), the second point can be (-1,-2), next point (1,4 etc…
Sample output file:

(The first 2 numbers are the absolute coordinates of the first point. The last number is the total number of coordinates)

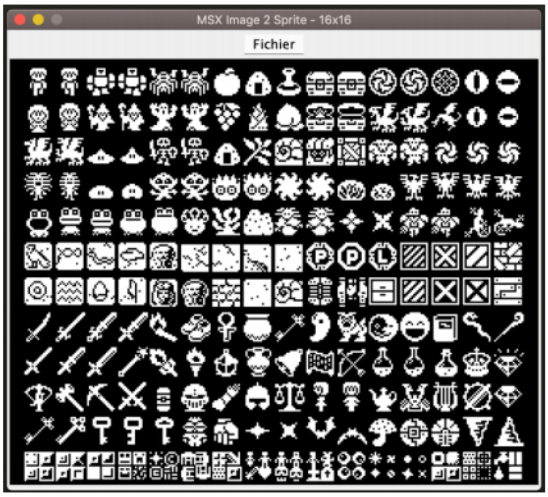
```
{99,26}
{0,1,1,0,1,0,1,0,1,0,1,0,1,0,1,1,1,0,55,30}
11
```

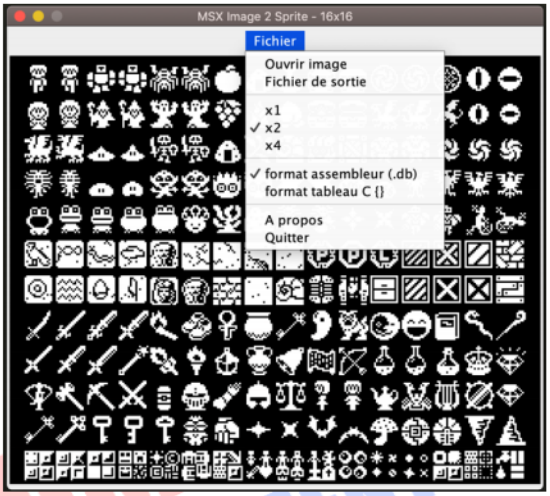
Relative compact Coord: It is the same as above, except that the coordinates X and Y are coded in a single byte instead of two. The first 4 bits of the byte correspond to the relative position on the X-axis, the other 4 bits correspond to the relative coordinate on the Y-axis.
Please note that this means that the points are never spaced more than 7 pixels apart.
To quickly decode these coordinates, you can use the ***SpriteFollow*** function of Fusion-C.
Sample output file:

```
{99,26}
{0x20,0x02,0x02,0x02,0x02,0x02,0x02,0x22,0x02,0x00}
11
```

Image To Sprite Editor

Image To sprite Editor allows you to simplify the creation of 16x16 sprites from already existing single-image images.





Inside the window the mouse pointer is transformed into a red square, it's the top left angle represents the mouse click pointer.
The left first click say to the program what is the color 0.
The second left click, will take a snapshot of the drawing and convert it into a 16x16 sprite data.

Open Image file: Let you load a 256x212 .png or .gif image in the background of the windows.
OutPut File: Let you choose the folder where you want to save the output file.
X1, X2, X4: let you zoom in the Window
C format or ASM format : let you choose the output data format. For use with C code, or assembler code.

Sample output file:

```
   {0xFF, 0xD5, 0xAA, 0x6B, 0x7F, 0x1C, 0x43, 0xBD,
    0xBD, 0xBD, 0x32, 0x36, 0x36, 0x30, 0x00, 0x00,
    0x80, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x80,
    0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}

Or

.db 0x7F, 0xFF, 0xD5, 0xAA, 0x6B, 0x7F, 0x1C, 0x43,
    0xBD, 0xBD, 0xBD, 0x32, 0x36, 0x36, 0x30, 0x00,
    0x00, 0x80, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00,
    0x80, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00
```

Technical information about MSX & MSX2

Here we will describe essential technical information about MSX, you may not need them to code your game in C with FUSION-C. But, in some cases it may be useful. Also it's a way to understand how MSX computers are working ... Especially about the graphics capabilities