

# ***FUSION-C***

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

**Quick Manual for FUSION-C v1.2**

Revision – 2019 August, 16



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

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

**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**



**Attribution-ShareAlike 4.0 International  
(CC BY-SA 4.0)**

**You are free to :**

**Share** : Copy and redistribute the material in any medium or format

**Adapt** : remix, transform, and build upon the material for any purpose, even commercially.

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 permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

## FUSION-C

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

### Sound FX Editor

AYFX Editor by Shiru.

<https://shiru.undergrund.net/software.shtml>

Source code and updates can be found here :

<https://github.com/Threetwosevensixseven/ayfxedit-improved>

### Binary to hexadecimal converter

Bin2froh By Ming Chen

Source code and updates can be found here :

<https://sourceforge.net/projects/bin2froh>

### MSX Disk image manager

MSX DiskImage by ?

Disk-Manager by Lex Lechz

<http://www.lexlechz.at>

DSKTOOL v1.3 by Ricardo Bittencourt, Tony Cruise 2010, Natalia Pujol 2017

Updates and source code can be found here

[https://github.com/nataliapc/MSX\\_devs](https://github.com/nataliapc/MSX_devs)

### Disk to Rom converter

Disk2Rom 0.8 by Vincent van Dam

### MSX-DOS

MSX-DOS 1.03 By Microsoft

MSX-DOS 2.30 By ASCII

### Emulator

OpenMSX

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

C Library for MSX-DOS with SDCC compiler

**Sc2 GraphXConvertor 0.5**

by Eric Boez & Leandro Xorreia

**RLEWbCompressor 1.0b**

BY ERIC BOEZ & AORANTE

**For their help or contribution**

**I want to thank:**

**Sylvain Cregut**

**Mvac7/303bcn**

**Nester Soriano**

**T.Hara**

**Jorge Torres Chacón**

**GDX**

**Javi Lanvandeira**

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

If you need more information about how to install the tools chains, and to automatize the compilation process, or if you need MSX hardware information to make your games, please consider to buy the “complete journey” manual on **Amazon**.

Go to Amazon and just do a search: “**MSX FUSION-C LIBRARY**” or search for **ERIC BOEZ**, you will find a big 290 pages book about **FUSION-C**.

Consider buying this book to encourage our work for the MSX.

**Content of the FUSION-C , Complete journey:**

**What is « FUSION-C »**

**Installing the Tools Chain**

**Step 1 – Download files**

**Step 2 - Setting your working folder**

**Step 3 - Installing Sublime Text**

**Step 4 - Installing Hex2bin**

**Step 5 - Installing Open MSX Emulator**

**Step 6 - Installing SDCC package**

**Step 7 - Customize the SDCC Default Library**

**Step 8 - Customize the Compilation script ( Optional )**

**Start your first compilation**

**Example of a C program**

**Example of our working environment**

**Content of the FUSION-C library**

**MSX BASIC VS Fusion-C**

**The Library’s source code**

**The C standard functions**

**Adding Assembler source code inside your C program**

**Use command line arguments with your program**

**Technical information about MSX & MSX2**

**MSX Models summary**

**MSX 1 video screen modes**

**MSX 2 video screen modes**

**MSX2, screen Map and Vram To Vram Copy System**

**Screen 5 Map**

**Screen 6 Map**

**Screen 7 Map**

**Screen 8 Map**

**Vram to Vram copy**

**The Sprites**

**The MSX Cartridges and rom mapper**

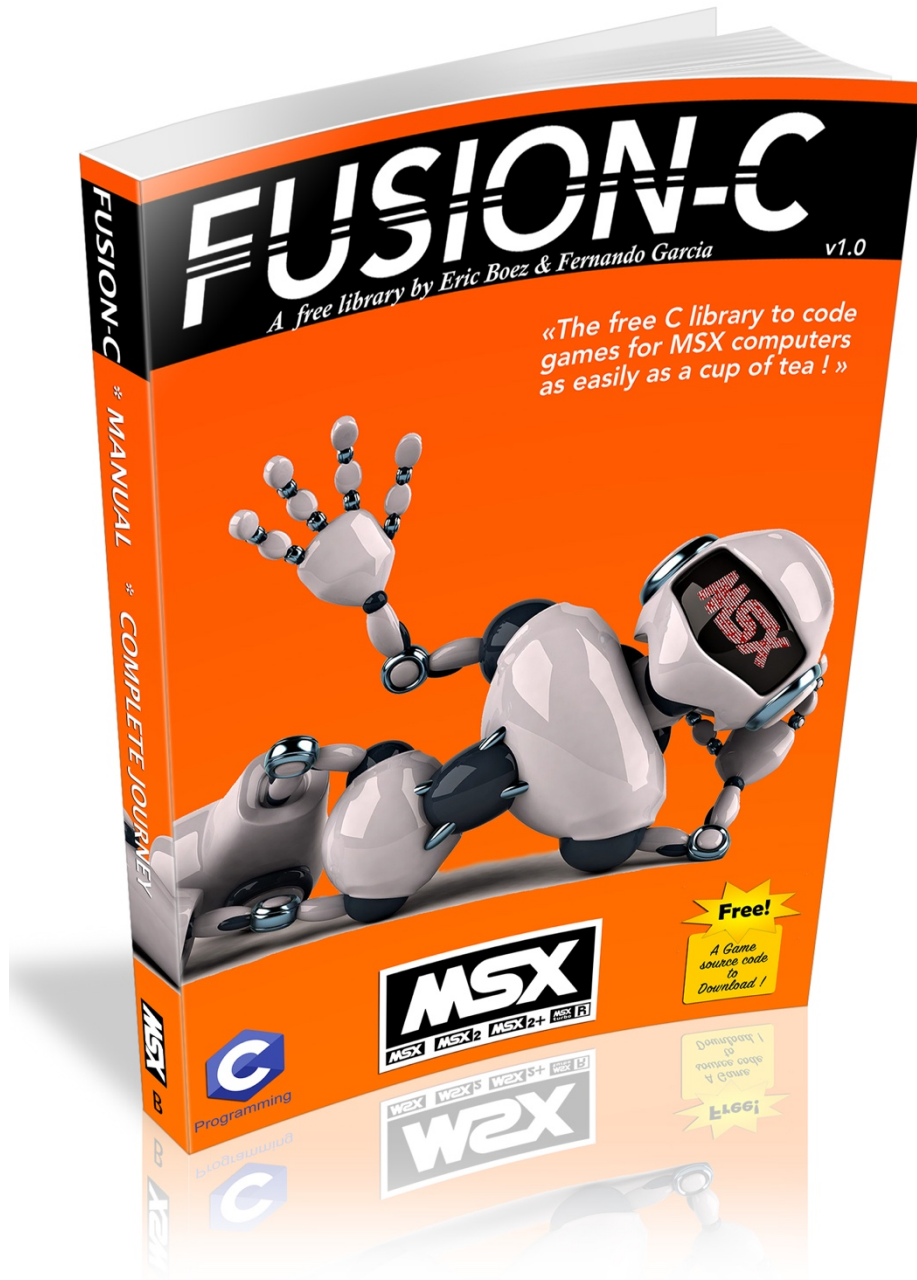
**MSX Ram Memory Mapper**

**MSX-DOS Operating System**

**MSX DOS Memory map**

**Memento about C language**

## FUSION-C



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





# FUSION-C

<b>What is « FUSION-C » ?</b> .....	<b>16</b>
<b>What's new in version 1.2</b> .....	<b>17</b>
<b>What's new in version 1.1</b> .....	<b>19</b>
<b>Installing the Tools Chain</b> .....	<b>21</b>
Download files .....	22
Setting your working folder .....	23
Customize the SDCC Default Library .....	25
Configuring the MSX you will use with OpenMSX .....	27
<b>Content of the FUSION-C library</b> .....	<b>29</b>
MSX FUSION_[ MSX_FUSION.H ].....	31
<b>Console Functions</b> .....	<b>31</b>
<i>Print</i> .....	31
<i>PrintNumber</i> .....	31
<i>PrintFNumber</i> .....	32
<i>PrintChar</i> .....	32
<i>PrintDec</i> .....	32
<i>InputChar</i> .....	32
<i>InputString</i> .....	32
<i>Getche</i> .....	33
<i>Locate</i> .....	33
<i>printf</i> .....	33
<i>PrintHex</i> .....	33
<i>CheckBreak</i> .....	34
<i>PutCharHex</i> .....	34
<b>Miscellaneous Functions</b> .....	<b>35</b>
<i>Cls</i> .....	35
<i>KeySound</i> .....	35
<i>FunctionKeys</i> .....	35
<i>ReadMSXtype</i> .....	36
<i>ChangeCap</i> .....	36
<i>Screen</i> .....	36
<i>Beep</i> .....	36
<i>RealTimer</i> .....	37
<i>SetRealTimer</i> .....	37
<i>CovoxPlay</i> .....	37
<i>RleWBToRam</i> .....	38
<i>RleWBToVram</i> .....	38
<b>Joystick &amp; mouse functions</b> .....	<b>39</b>
<i>JoystickRead</i> .....	39
<i>TriggerRead</i> .....	39
<i>MouseRead</i> .....	40
<i>MouseReadTo</i> .....	41
<b>Keyboard Functions</b> .....	<b>42</b>
<i>GetKeyMatrix</i> .....	42
<i>KillKeyBuffer</i> .....	42
<i>Inkey</i> .....	42
<i>WaitKey</i> .....	43
<i>KeyboardRead</i> .....	43
<b>I/O Port Functions</b> .....	<b>44</b>
<i>OutPort</i> .....	44
<i>InPort</i> .....	44

<i>OutPorts</i> .....	44
<b>VDP Functions</b> .....	<b>45</b>
<i>VDPstatus</i> .....	45
<i>VDPstatusNi</i> .....	45
<i>VDPwriteNi</i> .....	45
<i>VDPwrite</i> .....	45
<i>IsVsync</i> .....	45
<i>IsHsync</i> .....	46
<i>Vpeek</i> .....	46
<i>Vpoke</i> .....	46
<i>VpokeFirst</i> .....	46
<i>VpokeNext</i> .....	46
<i>VpeekFirst</i> .....	46
<i>VpeekNext</i> .....	47
<i>Width</i> .....	47
<i>SetColors</i> .....	47
<i>SetBorderColor</i> .....	48
<i>SetDisplayPage</i> .....	48
<i>SetActivePage</i> .....	48
<i>SetScrollH</i> .....	48
<i>SetScrollV</i> .....	48
<i>HideDisplay</i> .....	49
<i>ShowDisplay</i> .....	49
<i>FillVram</i> .....	49
<i>PutText</i> .....	49
<i>VDP50Hz</i> .....	49
<i>VDP60Hz</i> .....	49
<i>VDPLinesSwitch</i> .....	50
<i>CopyRamToVram</i> .....	50
<i>CopyVramToRam</i> .....	50
<i>GetVramSize</i> .....	50
<b>Type Functions</b> .....	<b>51</b>
<i>IsAlphaNum</i> .....	51
<i>IsAlpha</i> .....	51
<i>IsAscii</i> .....	51
<i>IsCtrl</i> .....	51
<i>IsDigit</i> .....	51
<i>IsGraph</i> .....	51
<i>IsLower</i> .....	52
<i>IsUpper</i> .....	52
<i>IsPrintable</i> .....	52
<i>IsPunctuation</i> .....	52
<i>IsSpace</i> .....	52
<i>IsHexDigit</i> .....	53
<i>IsPositive</i> .....	53
<i>IntToFloat</i> .....	53
<i>IntSwap</i> .....	53
<b>String Functions</b> .....	<b>54</b>
<i>CharToLower</i> .....	54
<i>CharToUpper</i> .....	54
<i>StrCopy</i> .....	54
<i>NStrCopy</i> .....	54

## FUSION-C

<i>StrConcat</i> .....	54
<i>NStrConcat</i> .....	54
<i>StrLen</i> .....	55
<i>StrCompare</i> .....	55
<i>NStrCompare</i> .....	55
<i>StrChr</i> .....	55
<i>StrPosStr</i> .....	55
<i>StrSearch</i> .....	55
<i>StrPosChr</i> .....	56
<i>StrLeftTrim</i> .....	56
<i>StrRightTrim</i> .....	56
<i>StrReplaceChar</i> .....	56
<i>StrReverse</i> .....	56
<i>Itoa</i> .....	57
<b>Memory Functions</b> .....	<b>58</b>
<i>Poke</i> .....	58
<i>Pokew</i> .....	58
<i>Peek</i> .....	58
<i>Peekw</i> .....	58
<i>MemChr</i> .....	58
<i>MemFill</i> .....	59
<i>MemCopy</i> .....	59
<i>MemCopyReverse</i> .....	59
<i>MemCompare</i> .....	59
<i>MMalloc</i> .....	59
<i>ReadTPA</i> .....	60
<i>ReadSP</i> .....	60
<b>Interrupt Functions</b> .....	<b>61</b>
<i>EnableInterrupt</i> .....	61
<i>DisableInterrupt</i> .....	61
<i>Halt</i> .....	61
<i>Suspend</i> .....	61
<i>InitInterruptHandler</i> .....	61
<i>EndInterruptHandler</i> .....	62
<i>SetInterruptHandler</i> .....	62
<b>PSG Functions</b> .....	<b>63</b>
<i>InitPSG</i> .....	63
<i>PSGread</i> .....	63
<i>PSGwrite</i> .....	63
<b>MSX-DOS File I/O Functions</b> .....	<b>64</b>
<i>Predefined macros &amp; structures</i> .....	64
<i>fcb_open</i> .....	65
<i>fcb_create</i> .....	65
<i>fcb_close</i> .....	65
<i>Fcb_read</i> .....	66
<i>Fcb_write</i> .....	66
<i>fcb_find_first</i> .....	66
<i>Fcb_find_next</i> .....	66
<b>MSX-DOS Functions</b> .....	<b>67</b>
<i>Predefined macros &amp; Structures</i> .....	67
<i>GetDate</i> .....	67
<i>GetTime</i> .....	68

## C Library for MSX-DOS with SDCC compiler

<i>SetDate</i> .....	68
<i>SetTime</i> .....	68
<i>IntDos</i> .....	68
<i>IntBios</i> .....	69
<i>Exit</i> .....	69
<b>Turbo-r Functions</b> .....	<b>70</b>
<i>GetCPU</i> .....	70
<i>ChangeCPU</i> .....	70
<i>PCMPlay</i> .....	70
<b>File I/O [ IO.H ]</b> .....	<b>71</b>
<i>Predefined macros &amp; Structures</i> .....	71
<i>DiskLoad</i> .....	72
<i>GetOSVersion</i> .....	73
<i>Open</i> .....	73
<i>Create</i> .....	73
<i>FCBlist</i> .....	74
<i>Close</i> .....	74
<i>Read</i> .....	74
<i>Write</i> .....	75
<i>OpenAttrib</i> .....	75
<i>CreateAttrib</i> .....	75
<i>GetCWD</i> .....	75
<i>GetDisk</i> .....	76
<i>SetDisk</i> .....	76
<i>Ltell</i> .....	76
<i>Lseek</i> .....	76
<i>Remove</i> .....	76
<i>FindFirst</i> .....	77
<i>FindNext</i> .....	77
<i>MakeDir</i> .....	77
<i>RemoveDir</i> .....	77
<i>tell</i> .....	78
<i>seek</i> .....	78
<i>size</i> .....	78
<i>GetDiskParam</i> .....	78
<i>SetDiskTrAddress</i> .....	79
<i>GetDiskTrAddress</i> .....	79
<i>SectorRead</i> .....	79
<i>SectorWrite</i> .....	79
<b>MSX1 GRAPHICS [VDP_GRAPH1.H]</b> .....	<b>80</b>
<i>Predefined macros &amp; Structures</i> .....	80
<i>SC2WriteScr</i> .....	80
<i>SC2ReadScr</i> .....	80
<i>ReadBlock</i> .....	81
<i>WriteBlock</i> .....	81
<i>Get8px</i> .....	81
<i>Get1px</i> .....	81
<i>Set8px</i> .....	82
<i>Set1px</i> .....	82
<i>Clear8px</i> .....	82
<i>Clear1px</i> .....	82
<i>GetCol8px</i> .....	82

## FUSION-C

<i>SetCol8px</i> .....	83
<i>SC2Point</i> .....	83
<i>SC2Pset</i> .....	83
<i>SC2Line</i> .....	83
<i>SC2Rect</i> .....	84
<i>SC2Paint</i> .....	84
<i>SC2Draw</i> .....	84
<b>MSX2 GRAPHICS [vdp_graph2.h].....</b>	<b>85</b>
<i>Predefined Structures and macros</i> .....	85
<i>Logical operations</i> .....	86
<i>vMSX</i> .....	86
<i>WriteScr</i> .....	86
<i>ReadScr</i> .....	87
<i>SetSC5Palette</i> .....	87
<i>RestoreSC5Palette</i> .....	88
<i>Pset</i> .....	88
<i>Point</i> .....	88
<i>Line</i> .....	88
<i>BoxLine</i> .....	88
<i>BoxFill</i> .....	89
<i>Paint</i> .....	89
<i>Draw</i> .....	89
<i>HMMM</i> .....	90
<i>LMMM</i> .....	91
<i>fLMMM</i> .....	91
<i>HMCM</i> .....	92
<i>HMCM_SC8</i> .....	92
<i>YMMM</i> .....	92
<i>HMMC</i> .....	93
<i>LMMC</i> .....	93
<i>HMMV</i> .....	94
<i>LMMV</i> .....	94
<b>SPRITES [vdp_SPRITES.h] .....</b>	<b>95</b>
<i>SpriteOn</i> .....	95
<i>SpriteOff</i> .....	95
<i>Sprite8</i> .....	95
<i>Sprite16</i> .....	95
<i>SpriteSmall</i> .....	95
<i>SpriteDouble</i> .....	96
<i>SpriteReset</i> .....	96
<i>SpriteCollision</i> .....	96
<i>SpriteCollisionX</i> .....	96
<i>SpriteCollisionY</i> .....	96
<i>SetSpritePattern</i> .....	97
<i>PutSprite</i> .....	97
<i>Sprite32Bytes</i> .....	97
<i>SC5SpriteColors</i> .....	97
<i>SC8SpriteColors</i> .....	97
<b>CIRCLE [VDP_CIRCLE] .....</b>	<b>98</b>
<i>CircleFilled</i> .....	98
<i>Circle</i> .....	98
<i>SC2CircleFilled</i> .....	98

## C Library for MSX-DOS with SDCC compiler

<i>SC2Circle</i> .....	98
<b>MSX-DOS 2 RAM MAPPER [RAMMAPPER.H]</b> .....	<b>99</b>
<i>InitRamMapperInfo</i> .....	99
<i>Get_PN</i> .....	99
<i>Put_PN</i> .....	100
<i>AllocateSegment</i> .....	100
<i>FreeSegment</i> .....	100
<b>P S G [PSG.H]</b> .....	<b>101</b>
<i>Sound</i> .....	101
<i>SoundFX</i> .....	101
<i>SetChannelA</i> .....	101
<i>SilencePSG</i> .....	102
<i>GetSound</i> .....	102
<i>SetTonePeriod</i> .....	102
<i>SetNoisePeriod</i> .....	102
<i>SetEnvelopePeriod</i> .....	102
<i>SetVolume</i> .....	103
<i>SetChannel</i> .....	103
<i>PlayEnvelope</i> .....	103
AYFX PLAYER.....	104
<i>InitFX</i> .....	104
<i>PlayFX</i> .....	104
<i>UpdateFX</i> .....	104
<i>TestFX</i> .....	104
<i>FreeFX</i> .....	105
<b>MUSIC PT3 REPLAYER [PT3REPLAYER.H]</b> .....	<b>106</b>
<i>PT3Init</i> .....	106
<i>PT3Play</i> .....	106
<i>PT3Rout</i> .....	106
<i>PT3Mute</i> .....	106
<b>The C standard functions</b> .....	<b>107</b>
<b>CTYPE.H</b> .....	<b>107</b>
<b>MATH.H</b> .....	<b>108</b>
<b>STDLIB.H</b> .....	<b>112</b>
<b>STRING.H</b> .....	<b>114</b>
<b>TIME.H</b> .....	<b>117</b>
<b>STDARG.H</b> .....	<b>118</b>
<b>Publish and distribute your game</b> .....	<b>120</b>
<b>Contacts</b> .....	<b>122</b>

### **What is « FUSION-C » ?**

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

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

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

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

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

In 2006 T. HARA a Japanese developer who worked on the one chip MSX made some functions and routines for the SDCC compiler. Some of its routines are included in this Library.

Other routines comes from the work of Néstor Soriano, Jorge Torres Chacon, MVAC7, Eric Boez, and Fernando Garcia.

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

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

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



## What's new in version 1.2

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

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

Added functions :

**HMMM** (vdp\_graph2.h)

Vram to Vram copy function for MSX2 graphic modes

**LMMC** (vdp\_graph2.h)

Ram to Vram copy function with logical operation

**YMMM** (vdp\_graph2.h)

Vram to Vram Copy function, only on Y axe

**HMMV** (vdp\_graph2.h)

High speed VDP filled rectangle drawing

**LMMV** (vdp\_graph2.h)

High speed VDP filled rectangle drawing with logical op

**MouseReadTo** (msx\_fusion.h)

A simple way to read mouse direction and mouse buttons

**Itoa** (msx\_fusion.h)

An integer to string conversion function

**StrReverse** (msx\_fusion.h)

Reverse order of letters inside a string

**BoxFill** (msx\_fusion.h)

Simple way to draw a filled rectangle

**BoxLine** (msx\_fusion.h)

Simple way to draw rectangle lines

Some functions have been removed because they were in duplicate :

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

## FUSION-C

Fixed or rewritten source code's functions :

**RleWBToRam** and **RleWBToVram**

**MouseRead**

**LMMM** ( \!/ variables order changed)

**Line** (Now Works on Screen 7)

**Point** (Now Works on Screen 7)

**Pset** (Now Works on Screen 7)

New source code examples where added to the example directory.

## What's new in version 1.1

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

More than that, many functions have been completely rewritten, for more speed and more compatibilities in your programs.

In addition, 22 new added functions complete FUSION-C with new possibilities like, mouse support, interrupt handler, a real timer, the ability to write and read sectors of disks and more ...

Added functions :

**-MouseRead** (msx\_fusion.h)

Read MSX mouse coordinates attached to a joystick port

**-SetRealTimer** (msx\_fusion.h)

**-RealTimer** (msx\_fusion.h)

To use a real timer in your programs

**-CovoxPlay** (msx\_fusion.h)

Play Sample sounds to a attached covox module

**-SC2Circle** (vdp\_circle.h)

**-SC2FilledCircle** (vdp\_circle.h)

Draw circles in Screen mode 2

**-VDPLinesSwitch** (msx\_fusion.h)

Enable MSX2 screen modes to display 212 or 192 horizontals lines

**-RleWBToVram** (msx\_fusion.h)

**-RleWBToRam** (msx\_fusion.h)

Decompress RLEWB compressed data to RAM or VRAM

**-GetDiskParam** (io.h)

**-GetDiskTrAddress** (io.h)

**-SetDiskTrAddress** (io.h)

**-SectorRead** (io.h)

**-SectorWrite** (io.h)

Enable you to write, read directly sectors of disk devices

**-CopyRamToVram** (msx\_fusion.h)

**-CopyVramToRam** (msx\_fusion.h)

Functions to copy memory block to ram or to vram

## FUSION-C

- PutText** (msx\_fusion.h)  
Enhanced version of previous PutText function, with logical operator
- GetVramSize** (msx\_fusion.h)  
Function to return the VRAM size of a MSX Computer
- SetBorderColor** (msx\_fusion.h)  
Set the screen border color
- InitInterruptHandler** (msx\_fusion.h)
- EndInterruptHandler** (msx\_fusion.h)
- SetInterruptHandler** (msx\_fusion.h)  
Functions to activate and control an interrupt handler

Some functions have been removed because they were in duplicate :

- WaitForKey** idem as WaitKey (msx\_fusion.h)
- KeyboardHit** idem as Inkey (msx\_fusion.h)
- Getcon** idem as Getche (msx\_fusion.h)

Other updates :

Added long support to printf

Fixed or rewritten source code's functions :

- getche.s
- vdp\_graph2.s
- setdate.s
- inkey.s
- waitkey.s
- getche.s
- joystickRead.c
- triggeread.c
- killekeybuffer.c
- fillvram.c
- crt0\_msxdos.s

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

Errors in definitions fixed in msx\_fusion.h

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

## Installing the Tools Chain

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

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

Here the tools you will need :

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

\* *mandatory*

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

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

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

## *Download files*

Download the « **SDCC Package** » for your operating system:

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

(Choose Version 3.6.0)

Download « **Sublime Text 3** », for your operating system:

<http://www.sublimetext.com/3>

Download the latest version of « **Open MSX** » emulator:

<https://openmsx.org/>

Download the « **Hex2Bin** » tool:

<https://sourceforge.net/projects/hex2bin/>

Finally, download the « **FUSION-C** » Library from :

<http://www.repro-factory.com>

(it's totally free)



[www.repro-factory.com/](http://www.repro-factory.com/)

On [www.repro-factory.com](http://www.repro-factory.com), you can also download a game made with FUSION-C and the full source code, and also some others useful archives ... ☺

## *Setting your working folder*

### Content of the Fusion-C Package Folder

#### **Working Folder**

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

**dsk** : Destination folder of compiled programs, ready to be tested with OpenMSX

**fusion-c** : library main folder.

**examples**: Source codes examples

**header**: all header files needed to declare functions in your code

**include**: binaries folder used by compiler

**lib**: this folder contains fusion-c dynamic library

**source**: this contains sources codes

**lib**: contains all library source code and library compilation script

**openMSX**: MSX Emulator folder

**tools**: contains useful tools, and information

*Make file*: Compilation script for MacOS / Linux

*Compil.bat* : Compilation script for windows

*test.c* : test program

## FUSION-C

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

Copy the « **working folder** » as is, where you want on your computer, and start to work from here.

All necessary tools and scripts will start from the use «working folder».

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



## ***Customize the SDCC Default Library***

The SDCC Library comes with all standard libraries and functions of C.

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

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

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

### **On Mac OS :**

Open your Shell/Terminal and go to the default Z80 library folder:

**/usr/local/share/sdcc/lib/z80/**

Type these commands:

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

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

### **On Windows :**

Open your Dos window and browse to the default Z80 library folder :

**C:\Program Files\SDCC\lib\z80\**

Type these commands:

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

## FUSION-C

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

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

## *Configuring the MSX you will use with OpenMSX*

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

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

The default content of this file is :

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

Thus, this mean the emulator will launch an *MSX2 Model Philips NMS 8255*, with :

- *MSXDOS2 expansion*
- *GFX9000 graphic expansion*
- *Simpl/Covox module* attached to the printer port
- Mouse connected to joystick port A
- First disk drive is emulate in */DSK* computer folder
- *the F12 Key* of your keyboard permut the screen between the *MSX Screen* and the *GFX 9000 Screen*.

By modifying this script you can add yourself other expansions like memory, music module etc ...

Check the openMSX documentation : <https://openmsx.org/manual/user.html>

## FUSION-C

## **Content of the FUSION-C library**

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

Functions are dispatched in some header files:

ayfx_player.h	:	AYFX, Sound FX player
io.h	:	File I/O functions
msx_fusion.h	:	Most of the needed functions are here
psg.h	:	PSG Sound functions
pt3replayer.h	:	PT3 Music replayer functions
rammapper.h	:	MSX-DOS 2 Memory Mapper
vdp_circle.h	:	Drawing circles on graphic screens
vdp_graph1.h	:	Graphic functions for MSX1
vdp_graph2.h	:	Graphic functions dor MSX2
vdp_sprites.h	:	Sprite related functions
Optional header files		
newTypes.h	:	Definition of old school variables
vars_msxBios.h	:	Definition list of BIOS 's routines
vars_msxDos.h	:	Definition list of MSX-DOS 's routines
vars_msxSystem.h	:	Definition list of System variables

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

g9klib.h	:	V9990 GFX9000 Support functions
----------	---	---------------------------------

and Gr8NET TCPIP Cartridge

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

## FUSION-C

## ***MSX FUSION*** ***[ MSX\_FUSION.H ]***

### **Console Functions**

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

<b><i>PrintNumber</i></b>	<b>CONSOLE</b>
<b>void    PrintNumber (unsigned int num)</b>	
<p>Prints the <b><i>num</i></b> number supplied in parameter to a text screen (console).</p>	

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

<i>PrintChar</i>	CONSOLE
<b>void</b> <b>PrintChar(char c)</b>	
Prints the character <i>c</i> to console (or to a text screen mode)	

<i>PrintDec</i>	CONSOLE
<b>void</b> <b>PrintDec(int num)</b>	
Prints signed integer <i>num</i> from -32768 to 32767 to a text screen mode	

<i>InputChar</i>	CONSOLE
<b>char</b> <b>InputChar()</b>	
Reads a character from console, and return it.	

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



<i>Getche</i>	CONSOLE
<b>char</b> <b>Getche()</b>	
Reads and displays character from console. Returns the readed character.	

<i>Locate</i>	CONSOLE
<b>void</b> <b>Locate(int x, int y)</b>	
Sets console's cursor to <i>X &amp; Y</i> coordinates	

<i>printf</i>	CONSOLE
<b>int</b> <b>printf (const char *fmt, ...)</b>	
<p>Prints formatted text data to console (text Screen mode).</p> <p>Example :      <b>int nb=19;</b>                    <b>char src[4]="bag";</b>                    <b>printf("\n\r I have %d lollipops in my %s ",nb,src);</b></p> <p>Supported format specifiers:</p> <p>%d or %i: signed int %ud or %ui: unsigned int  %x: hexadecimal int  %c: character  %s: string  %%: a % character  %l: signed long  %ul: unsigned long  %lx: hexadecimal long</p>	

<i>PrintHex</i>	V1.2	CONSOLE
<b>void</b> <b>PrintHex (unsigned int num)</b>		
Prints the hexadecimal representation of the integer <i>num</i> , on the text screen mode		

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

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

**Miscellaneous Functions**

<i><b>Cls</b></i>	<b>MISCELLANEOUS</b>
<b>void</b>	<b>Cls(void)</b>
Clears console or any screen mode	

<i><b>KeySound</b></i>	<b>MISCELLANEOUS</b>
<b>void</b>	<b>KeySound(char n)</b>
<p>Enables or disables Key Sound.</p> <p><i><b>n</b></i> value must be :</p> <p>0 : Disables Key Sound</p> <p>1 : Enables Key Sound</p>	

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

<i>ReadMSXtype</i>	MISCELLANEOUS
<b>char    ReadMSXtype(void)</b>	
<p>Reads and returns the MSX type.  Returned values :</p> <p>0 : MSX 1  1 : MSX 2  2 : MSX2+  3 : MSX Turbo-r</p>	

<i>ChangeCap</i>	MISCELLANEOUS
<b>void      ChangeCap(char n)</b>	
<p>Changes the state of the Cap Led.  <i>n</i> value must be :</p> <p>0 : Disables Cap Led  1 : Enables Cap Led</p>	

<i>Screen</i>	MISCELLANEOUS
<b>void      Screen( unsigned char mode )</b>	
<p>Sets display to specified screen mode.  <i>mode</i> can be a valid screen mode number, from 0 to 8</p>	

<i>Beep</i>	MISCELLANEOUS
<b>void      Beep(void)</b>	
<p>Plays a beep sound.</p>	

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

<i>SetRealTimer</i>	<i>v1.1</i>	MISCELLANEOUS
<b>void    SetRealTimer (unsigned int value);</b>		
<p>Sets the Real Clock timer of the MSX computer to a specific <i>value</i> between 0 and 65535.</p>		

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

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

<i>RleWBToVram</i>	<i>v1.1</i>	MISCELLANEOUS
<b>void RleWBToVram(unsigned int *RamAddress, unsigned int VramAddress);</b>		
<p>Decompress RLEWB data directly to Vram.</p> <p><b>*RamAddress</b> is the pointer address where RLEWB data are stored in RAM</p> <p><b>VramAddress</b> 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 <b>SetActivePage</b> function to set the VRAM page.</p> <p>Example : RleWBToVram (&amp;rledtata[0],0);</p> <p>Note : The RLEWB Compressor command line tool is provided in the "Tools" folder</p>		

**Joystick & mouse functions**

<i>JoystickRead</i>	JOYSTICK										
<b>char</b>	<b>JoystickRead(char joyNumber)</b>										
<p>Reads and returns state of a joystick.</p> <p><b>JoyNumber</b> must be :</p> <p>0 : Keyboard's Arrow keys  1 : Joystick port 1  2 : Joystick port 2</p> <p>Returned values:</p> <table> <tr> <td>0=inactive</td><td></td></tr> <tr> <td>1=up</td><td>2=up &amp; right</td></tr> <tr> <td>3=right</td><td>4=down &amp; right</td></tr> <tr> <td>5=down</td><td>6=down &amp; left</td></tr> <tr> <td>7=left</td><td>8=up &amp; left</td></tr> </table>		0=inactive		1=up	2=up & right	3=right	4=down & right	5=down	6=down & left	7=left	8=up & left
0=inactive											
1=up	2=up & right										
3=right	4=down & right										
5=down	6=down & left										
7=left	8=up & left										

<i>TriggerRead</i>	JOYSTICK
<b>char</b>	<b>TriggerRead(char TriggerNumber)</b>
<p>Reads and return state of a joystick button.</p> <p><b>TriggerNumber</b> must be :</p> <p>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</p> <p>Note, there is no second button for the keyboard.</p> <p>Returned values:</p> <p>0 if button is inactive  255 if button is active</p>	

<i>MouseRead</i>	v1.1	MOUSE
<b>unsigned int</b>	<b>MouseRead(Int MousePort);</b>	
Reads and returns the mouse offsets of the mouse connected at MousePort.		
The function returns the X offset and the Y offset into a 16bits value. Decoding this value must be done like this: <b>Mouse_offset=MouseRead(MousePort2);</b> <b>Xoffset=Mouse_offset &gt;&gt; 8;</b> <b>Yoffset=Mouse_offset &amp; 0xFF;</b>		
<i>MousePort</i> is refering to joystick port 1 or port 2, it is defined as a predefined macro You must use only : - <b>MousePort1</b> or <b>MousePort2</b> as parameters.		
Once you have decoded X offset and Y offset you can move a sprite object over the screen like this :		
<b>mx=mx-Xoffset;</b> <b>my=my-Yoffset;</b> <b>PuSprite(1,2,mx,my,15);</b>		
Note : If <b>MouseRead</b> returns <b>65535</b> as offset, this means no mouse is connected to the defined MousePort.		



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

**Keyboard Functions*****GetKeyMatrix*****KEYBOARD****char   GetKeyMatrix(char line)**

Returns the value of the specified ***line*** from the keyboard matrix. Each line provides the status of 8 keys. The state of the key is :

1 = not pressed

0 = pressed

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

***KillKeyBuffer*****KEYBOARD****void   KillKeyBuffer(void)**

Clears the key buffer

***Inkey*****KEYBOARD****unsigned char   Inkey(void)**

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

***WaitKey*****KEYBOARD****unsigned char    WaitKey(void)**

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

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

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

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

**I/O Port Functions**

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

<b><i>InPort</i></b>	<b>I/O PORT</b>
<b>unsigned char    InPort(unsigned char port)</b>	
Reads from a MSX <i>port</i>	

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

VDP Functions

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

<i>VDPstatusNi</i>	VDP
<b>unsigned char</b> <b>VDPstatusNi( unsigned char vdpreg )</b>	
Reads VDP Status register <i>vdpreg</i> and returns the value. The Interrupt is not modified by this function.	

<i>VDPwriteNi</i>	VDP
<b>void</b> <b>VDPwriteNi(unsigned char vdpreg, unsigned char data)</b>	
Writes <i>data</i> to the VDP register <i>vdpreg</i> . The Interrupt is not modified by this function.	

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

<i>IsVsync</i>	VDP
<b>unsigned Char</b> <b>IsVsync()</b>	
Detects Vblank state. Returns 1, if true.	

<i>IsHsync</i>	VDP
<b>unsigned Char</b> <b>IsHsync()</b>	
Detects HSynch state. Returns 1 if true.	

<i>Vpeek</i>	VDP
<b>unsigned char</b> <b>Vpeek( unsigned int address )</b>	
Reads and returns a byte from Vram Memory <i>address</i>	

<i>Vpoke</i>	VDP
<b>void</b> <b>Vpoke( unsigned int address, unsigned char data )</b>	
Writes the byte <i>data</i> in Vram at Memory <i>address</i>	

<i>VpokeFirst</i>	VDP
<b>void</b> <b>VpokeFirst( unsigned int address )</b>	
Sets first Vram <i>address</i> for multiple Vpokes. Use VpokeNext after this instruction.	

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

<i>VpeekFirst</i>	VDP
<b>unsigned char</b> <b>VpeekFirst( unsigned int address )</b>	
Sets first <i>address</i> for multiple Vpeek. Use VpeekNext after this instruction.	

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

<i>Width</i>	VDP
<b>void</b> <b>Width(char n)</b>	
Sets the width of a text screen mode, <i>n</i> must be between 1 and 80	

<i>SetColors</i>	VDP
<b>void</b> <b>SetColors( int ForeCol, int BackgrCol, int BorderCol )</b>	
Sets, foreground <i>ForeCol</i> color, background <i>BackgrCol</i> color and border color <i>BorderCol</i> . Supports MSX2's screen mode like Screen8 ; colors can be a number between 0 and 255.	
<b>MSX1 colors palette :</b> - TRANSPARENT            : 0x00 - BLACK                    : 0x01 - MEDIUM_GREEN         : 0x02 - LIGHT_GREEN            : 0x03 - DARK_BLUE              : 0x04 - LIGHT_BLUE              : 0x05 - DARK_RED                : 0x06 - CYAN                     : 0x07 - MEDIUM_RED             : 0x08 - LIGHT_RED                : 0x09 - DARK_YELLOW            : 0x0A - LIGHT_YELLOW            : 0x0B - DARK_GREEN             : 0x0C - MAGENTA                 : 0x0D - GRAY                     : 0x0E - WHITE                    : 0x0F	

<i>SetBorderColor</i>	VDP
<b>void    SetBorderColor( unsigned char BorderCol)</b>	
Sets the screen border color <i>BorderCol</i> of the screen. It uses same colors as <i>SetColors</i> function.	

<i>SetDisplayPage</i>	VDP
<b>void    SetDisplayPage( unsigned char page )</b>	
Sets the <i>page</i> of the screen mode which is displayed to the screen.	

<i>SetActivePage</i>	VDP
<b>void    SetActivePage( unsigned char page )</b>	
Sets the <i>page</i> of the screen mode which receive modifications	

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

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



<i>HideDisplay</i>	VDP
<b>void     HideDisplay(void)</b>	
Hides the screen display (Black screen)	

<i>ShowDisplay</i>	VDP
<b>void     ShowDisplay(void)</b>	
Shows the screen display if it was previously hidden.	

<i>FillVram</i>	VDP
<b>void     FillVram(int Startaddress, char value, int length)</b>	
Fills the Vram from <i>Startaddress</i> with <i>length</i> bytes.	

<i>PutText</i>	<i>v1.1</i>	VDP
<b>void     PutText( int X, int Y, char *str, char LogOp );</b>		
Prints the text string <i>*str</i> to graphic screen (mode 3 to 8) at position <i>X</i> , <i>Y</i> <i>LogOp</i> 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)		

<i>VDP50Hz</i>	VDP
<b>void     VDP50Hz(void)</b>	
Switches the MSX2 VDP to 50 Hz Pal Mode	

<i>VDP60Hz</i>	VDP
<b>void     VDP60Hz(void)</b>	
Switches the MSX2 VDP to 60 Hz NTSC Mode	

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

<i>CopyRamToVram</i>	<i>v1.1</i>	<b>VDP</b>
<b>void CopyRamToVram(void *SrcRamAddress, unsigned int DestVramAddress, unsigned int Length);</b>		
<p>Copy a Ram Memory Block to a Vram Address.</p> <ul style="list-style-type: none"> <li>- <i>*SrcRamAddress</i>, must point to a Memory address or variable.</li> <li>- <i>DestVramAddress</i>, must be a valid VRAM Address</li> <li>- <i>Length</i>, is the length of the Memory Block to Copy</li> </ul>		

<i>CopyVramToRam</i>	<i>v1.1</i>	<b>VDP</b>
<b>void CopyVramToRam(unsigned int SrcVramAddress, void *DestRamAddress, unsigned int Length);</b>		
<p>Copy a Vram Memory Block to a Ram Address.</p> <ul style="list-style-type: none"> <li>- <i>SrcVramAddress</i>, must be a valid VRAM Address</li> <li>- <i>*DestRamAddress</i>, must point to a memory address, or variable.</li> <li>- <i>Length</i>, is the length of the Memory Block to Copy</li> </ul>		

<i>GetVramSize</i>	<i>v1.1</i>	<b>VDP</b>
<b>unsigned char GetVramSize(void);</b>		
<p>Returns the Vram size of the MSX Computer.          Returned values can be, 16, 64, 128.</p>		

**Type Functions**

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

<i>IsAlpha</i>	TYPE
<b>int</b> <b>IsAlpha(char c)</b>	
Checks if the supplied value <i>c</i> is alpha : A..Z or a..z	

<i>IsAscii</i>	TYPE
<b>int</b> <b>isAscii(char c)</b>	
Check if the supplied value <i>c</i> is an ASCII character : - !..~	

<i>IsCtrl</i>	TYPE
<b>int</b> <b>IsCtrl(char c)</b>	
Checks if if the supplied value <i>c</i> is an unprintable control symbol	

<i>IsDigit</i>	TYPE
<b>int</b> <b>IsDigit(char c)</b>	
Checks if the supplied value <i>c</i> is a digit: 0..9	

<i>IsGraph</i>	TYPE
<b>int</b> <b>IsGraph(char c)</b>	
Checks if the supplied value <i>c</i> is a graph representation	

<i>IsLower</i>	TYPE
<b>int</b> <b>IsLower(char c)</b>	
Checks if the supplied value <i>c</i> is a lower char	

<i>IsUpper</i>	TYPE
<b>int</b> <b>IsUpper(char c)</b>	
Checks if the supplied value <i>c</i> is an upper char	

<i>IsPrintable</i>	TYPE
<b>int</b> <b>IsPrintable(char c)</b>	
Checks if the supplied value <i>c</i> is a printable char	

<i>IsPunctuation</i>	TYPE
<b>int</b> <b>IsPunctuation(char c)</b>	
Checks if the supplied value <i>c</i> is a punctuation sign	

<i>IsSpace</i>	TYPE
<b>int</b> <b>IsSpace(char c)</b>	
Checks if the supplied value <i>c</i> is a space	

<i>IsHexDigit</i>	TYPE
<b>int</b> <b>IsHexDigit(char c)</b>	
Checks if the supplied value <i>c</i> is a hexadecimal digit	

<i>IsPositive</i>	TYPE
<b>Int</b> <b>IsPositive(int c)</b>	
Checks if the supplied value <i>c</i> is positive. Return -1 if negative, 1 if positive, 0 if value is null.	

<i>IntToFloat</i>	TYPE
<b>Float</b> <b>IntToFloat(int c)</b>	
Returns a float value of the supplied value <i>c</i>	

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

String Functions

<i>CharToLower</i>	STRING
<b>char</b> <b>CharToLower(char c)</b>	
Returns the supplied value <i>c</i> to the lowercase	

<i>CharToUpper</i>	STRING
<b>char</b> <b>CharToUpper(char c)</b>	
Returns the supplied value <i>c</i> to the uppercase	

<i>StrCopy</i>	STRING
<b>void</b> <b>StrCopy( char *dst, char *src )</b>	
Copy string from <i>*src</i> to <i>*dst</i>	

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

<i>StrConcat</i>	STRING
<b>void</b> <b>StrConcat( char *dst, char *src )</b>	
Concatenates string <i>*src</i> at the end of <i>*dst</i>	

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

<i>StrLen</i>	STRING
<b>int</b> <b>StrLen( char *string )</b>	
Returns length of the <i>*string</i>	

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

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

<i>StrChr</i>	STRING
<b>int</b> <b>StrChr( char *string, char c )</b>	
It returns 1 if <i>c</i> is found in <i>*string</i> , otherwise returns -1	

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

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

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

<i>StrLeftTrim</i>	STRING
<b>void</b> <b>StrLeftTrim( char *string )</b>	
Removes left spaces inside <i>*string</i>	

<i>StrRightTrim</i>	STRING
<b>void</b> <b>StrRightTrim( char *string )</b>	
Removes right spaces inside <i>*string</i>	

<i>StrReplaceChar</i>	STRING
<b>void</b> <b>StrReplaceChar( char *string, char c, char new_c )</b>	
Replaces all chars <i>c</i> by <i>new_c</i> in string <i>*string</i>	

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



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

# FUSION-C

## Memory Functions

<i>Poke</i>	MEM
<b>void</b> <b>Poke( unsigned int address, char data )</b>	
Writes a byte <i>data</i> to Memory <i>address</i>	

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

<i>Peek</i>	MEM
<b>Unsigned char</b> <b>Peek( unsigned int address )</b>	
Reads and returns a byte value from Memory <i>address</i>	

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

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

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

<i>MemCopy</i>	MEM
<b>void</b> <b>MemCopy( char *dst, char *src, int n )</b>	
Copy <i>n</i> bytes from <i>*src</i> to <i>*dst</i>	

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

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

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

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

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

**Interrupt Functions**

<i>EnableInterrupt</i>	INTERRUPT
<b>void</b> <b>EnableInterrupt(void)</b>	
Enables the Interrupt base system. May be useful in some circumstances	

<i>DisableInterrupt</i>	INTERRUPT
<b>void</b> <b>DisableInterrupt(void)</b>	
Disables the Interrupt base system. May be useful in some circumstances	

<i>Halt</i>	INTERRUPT
<b>void</b> <b>Halt(void)</b>	
Same as the Halt Asm Function. Wait for Interrupt.	

<i>Suspend</i>	INTERRUPT
<b>void</b> <b>Suspend(void)</b>	
Suspend Z80	

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

<i>EndInterruptHandler</i>	<i>v1.1</i>	<b>INTERRUPT</b>
<b>void</b>	<b>EndInterruptHandler(void)</b>	
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.		

<i>SetInterruptHandler</i>	<i>v1.1</i>	<b>INTERRUPT</b>
<b>void</b>	<b>SetInterruptHandler(char (*p_handler)( void ));</b>	
<p>Sets the function of your program the interrupt handler process will call at each interruption.</p> <p>The called function must be a function without any parameters, but it can make use of global variables.</p> <p>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.</p> <p>Code example :</p> <pre>static char my_interrupt( void ) { if( IsVsync() == 0 ) return 0;     count++;     return 1; }  void main (void) { InitInterruptHandler(); SetInterruptHandler( my_interrupt ); ... }</pre>		

**PSG Functions**

<i><b>InitPSG</b></i>	<b>PSG</b>
<b>void    InitPSG(void)</b>	
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.	

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

<i><b>PSGwrite</b></i>	<b>PSG</b>
<b>void    PSGwrite( unsigned char psgreg, unsigned char data)</b>	
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 ;
```

**Return code :**

**FCB\_SUCCESS**                      **0x00**

**Dir attributes :**

**FCB\_DIR ::attribute**

**FCB\_ATTR\_READONLY**              **0x01**

**FCB\_ATTR\_HIDDEN**                **0x02**

**FCB\_ATTR\_SYSTEM**                **0x04**

**FCB\_ATTR\_VOLUME**                **0x08**

**FCB\_ATTR\_DIR**                    **0x10**

**FCB\_ATTR\_ARCHIVE**               **0x20**

***fcb\_open*****FILE I/O**

**unsigned char    fcb\_open( FCB \*p\_fcb )**

Opens a file. Returns 0 if success

***fcb\_create*****FILE I/O**

**unsigned char    fcb\_create( FCB \*p\_fcb )**

Creates a file on media. Returns 0 on success

***fcb\_close*****FILE I/O**

**unsigned        char fcb\_close( FCB \*p\_fcb )**

Closes a file previously opened. Returns 0 on success

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

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

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

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

**MSX-DOS Functions*****Predefined macros & Structures***

```

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

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

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

```

***GetDate*****MSX\_DOS**

**void            GetDate (DATE \*date)**

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

*Code example, get and show date :*

```

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

```

<i>GetTime</i>	MSX_DOS
<b>void</b>	<b>GetTime (TIME *time)</b>
<p>Gets the current time, and send it to <i>*time</i> structure  <i>Code example, get and show date :</i></p> <pre>TIME __at (0x2100) tm ; GetTime(&amp;tm) ; PrintDec(tm.hour) ; putch(' :'); PrintDec (tm.min) ; putch(' :'); PrintDec (tm.sec) ;</pre>	

<i>SetDate</i>	MSX_DOS
<b>int</b>	<b>SetDate (DATE *date)</b>
<p>Sets the system date thru the <i>*date</i> structure. Returns 0 if valid.</p>	

<i>SetTime</i>	MSX_DOS
<b>int</b>	<b>SetTime (TIME *time)</b>
<p>Sets the system time thru the <i>*time</i> structure. Returns 0 if valid.</p>	

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

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

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

## FUSION-C

### Turbo-r Functions

<i>GetCPU</i>	TURBO-R
<b>char    GetCPU (void)</b>	
<p>Gets the CPU Mode of the R800 processor.</p> <p>Returned values :</p> <p>0 : Z80 mode</p> <p>1 : R800 Rom Mode</p> <p>2 : R800 Ram Mode</p>	

<i>ChangeCPU</i>	TURBO-R
<b>void    ChangeCPU (char n)</b>	
<p>Changes the CPU Mode of the R800 processor.</p> <p><b><i>n</i></b> must be :</p> <p>0 : Z80 mode</p> <p>1 : R800 Rom Mode</p> <p>2 : R800 Ram Mode</p>	

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

## ***File I/O***

### **[IO.H]**

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

#### ***Predefined macros & Structures***

```
SEEK_SET    0
SEEK_CUR    1
SEEK_END    2
```

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

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

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

## FUSION-C

```
extern  FCBLIST *FCBs( void ) ;

extern  int  _io_errno ;

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

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

} DSKPARAMS;
```

### ***DiskLoad***

### **IO**

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

Loads binary file from disk to RAM.

- ***\*filename*** : is 11 chars DOS1 for FCB : Example « MYFILE01BIN »
- ***address*** : where to load the first byte
- ***run\_address*** : if not 0, then where to CALL after loaded

Returns 0 on success



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

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

<i>Create</i>	IO
Int	Create (char *file_name)
Creates a file, named as <i>*file_name</i>	

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

<i>Close</i>	<b>IO</b>
<b>Int</b>	<b>Close ( int fH )</b>
<p>Closes a file, previously opened. <i><b>fH</b></i> is the file handler that must be provided as argument.</p>	

<i>Read</i>	<b>IO</b>
<b>Int</b>	<b>Read (int fH, void *buffer, unsigned int nbytes)</b>
<p>Reads <b>nbytes</b> bytes from a file defined by <i><b>fH</b></i> handler to <i><b>*buffer</b></i>. The file must be previously opened.</p>	

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

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

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

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

<i>GetDisk</i>	IO
<b>Int</b> <b>GetDisk(void)</b>	
Gets and returns current drive number.	

<i>SetDisk</i>	IO
<b>void</b> <b>SetDisk(int diskno)</b>	
Sets <i>diskno</i> as current drive number.	

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

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

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

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

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

<b>FindNext</b>	<b>IO</b>
<b>Int</b> <b>FindNext(char *result)</b>	
See findfirst function. Find next occurrence.	

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

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

<i><b>_tell</b></i>	<b>IO</b>
<b>unsigned long</b> <b>_tell(int fH)</b>	
Reads and return file position from FCB Only for MSX DOS 1	

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

<i><b>_size</b></i>	<b>IO</b>
<b>unsigned long</b> <b>_size(int fH)</b>	
Returns the file size from FCB Only for MSX DOS 1	

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

<i>SetDiskTrAddress</i>	<i>v1.1</i>	<b>IO</b>
<b>void SetDiskTrAddress(unsigned int *address);</b>		
<p>Sets the Memory <i>*address</i> where the data will be transferred, when a Reading Sectors instruction is called or when a Writing function is called.</p> <p>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 <i>*address</i> pointer must point to a 512 bytes variable area (minimum).</p>		

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

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

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

## **MSX1 GRAPHICS**

### **[VDP\_GRAPH1.H]**

MSX1 Graphic and draw functions.

#### ***Predefined macros & Structures***

```
// filling mode
NO_FILL 0x00
FILL_ALL 0xFF

/* structure to set/get color of 8 pixels */

typedef struct {
    int col ;      // color number 0..15 for pixels of pattern
    int bg ;      // background color number 0..15
} pxColor ;
```

#### ***SC2WriteScr***

#### **VDP\_GRAPH1**

```
void SC2WriteScr( unsigned int addr_fromPalettes, unsigned int
addr_fromColours )
```

Writes RAM to VRAM (2x6144 bytes)  
Use with Screen mode 2 or 3

#### ***SC2ReadScr***

#### **VDP\_GRAPH1**

```
void SC2Read_Scr( unsigned int addr_toPalettes, unsigned int
addr_toColours )
```

Reads VRAM to RAM(2x6144 bytes)  
Use with Screen mode 2 or 3



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

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

<i>Get8px</i>	VDP_GRAPH1
<b>int Get8px( int X, int Y )</b>	
Gets byte of 8-pixels at (X,Y)	

<i>Get1px</i>	VDP_GRAPH1
<b>int Get1px( int X, int Y )</b>	
Gets pixel of 8-pixels at (X,Y). Returns 0 if not set.	

<i>Set8px</i>	VDP_GRAPH1
<b>void Set8px( int X, int Y )</b>	
Sets whole byte of 8-pixels at (X,Y)	

<i>Set1px</i>	VDP_GRAPH1
<b>void Set1px( int X, int Y )</b>	
Sets pixel of 8-pixels at (X,Y)	

<i>Clear8px</i>	VDP_GRAPH1
<b>void Clear8px( int X, int Y )</b>	
Clears byte (sets=0) of 8-pixels at (X,Y)	

<i>Clear1px</i>	VDP_GRAPH1
<b>void Clear1px( int X, int Y )</b>	
Clears pixel (sets=0) of 8-pixels at (X,Y)	

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

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

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

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

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

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

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

<i><b>SC2Draw</b></i>	<b>VDP_GRAPH1</b>
<b>void SC2Draw( char *drawcommands )</b>	
<p>Remake of BASICs « draw » with original commands (except A,X) syntax</p>	

## **MSX2 GRAPHICS**

### **[vdp\_graph2.h]**

MSX2 Graphic and draw functions.

#### ***Predefined Structures and macros***

##### **Logical fill for rectangle and circle**

FILL\_ALL                0xFF

NO\_FILL                0x00

##### **Palette**

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

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

##### **fLMM variable structure :**

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

**Logical operations**

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

**SC** : is for source color code

**DC** : is for destination color code

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

**vMSX****VDP\_GRAPH2**

**Int**      **vMSX(void)**

Check MSX VDP version.

Returns 1 for MSX1, or 2 for MSX2

**WriteScr****VDP\_GRAPH2**

**void**      **WriteScr(unsigned int addr\_from)**

Writes data from RAM *addr\_from* to VRAM screen

Only work with screen 5 (page 0) mode. This function copies 0x6A00 bytes to the VRAM

<i>ReadScr</i>	VDP_GRAPH2
----------------	------------

<b>void    ReadScr( unsigned int addr_to )</b>
--

Reads VRAM screen (page 0) to RAM memory address <i>addr_to</i> .
---

Only works with screen 5 mode. This function copies 0x6A00 bytes to RAM Use it with Screen5 mode
---

<i>SetSC5Palette</i>	VDP_GRAPH2
----------------------	------------

<b>void    SetSC5Palette ((Palette *) mypalette)</b>
--

Sets the screen 5 color palette with « <i>mypalette</i> » structure data.
---

The predefined « Palette Structure » is composed of 16 lines formatted like this : N, R, G, B
--

N is the number of the color (0 ... 15). R, G, and B are the level of Red, Green or Blue in the final color. R, G and B must be between 0 and 7.
---

Example :
-----------

<b>char mypalette[] =</b>
---------------------------

<pre> {     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 </pre>
---

<b>};</b>
-----------

<b>SetSC5Palette((Palette *) mypalette);</b>
--

<i>RestoreSC5Palette</i>	VDP_GRAPH2
<b>void      RestoreSC5Palette (void)</b>	
Sets screen 5 color palette to the default palette	

<i>Pset</i>	VDP_GRAPH2
<b>void    Pset(int X, int Y, int color, int OP)</b>	
Draws a pixel at <i>X,Y</i> with the defined <i>color</i> and logical operation <i>OP</i>	

<i>Point</i>	VDP_GRAPH2
<b>char    Point(int X, int Y)</b>	
Reads and return color of the pixel at <i>X,Y</i>	

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

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



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

<i>Paint</i>	VDP_GRAPH2
<b>void</b>	<b>Paint( int X, int Y, int color, int border_color )</b>
Slow and dummy paint. Paint the area at <i>X,Y</i> with <i>color</i> , stopping at <i>border_color</i>	

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

High speed VDP Commands

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

GRAPHIC 4 (SCREEN 5)			GRAPHIC 5 (SCREEN 6)	
(0,0)	(255,0)	00000H	(0,0)	(511,0)
Page 0			Page 0	
(0,255)	(255,255)		(0,255)	(511,255)
(0,256)	(255,256)	08000H	(0,256)	(511,256)
Page 1			Page 1	
(0,511)	(255,511)		(0,511)	(511,511)
(0,512)	(255,512)	10000H	(0,512)	(511,512)
Page 2			Page 2	
(0,767)	(255,767)		(0,767)	(511,767)
(0,768)	(255,768)	18000H	(0,768)	(511,768)
Page 3			Page 3	
(0,1023)	(255,1023)		(0,1023)	(511,1023)
		1FFFFH		
GRAPHIC 7 (SCREEN 8)			GRAPHIC 6 (SCREEN 7)	
(0,0)	(255,0)	00000H	(0,0)	(511,0)
Page 0			Page 0	
(0,255)	(255,255)		(0,255)	(511,255)
(0,256)	(255,256)	10000H	(0,256)	(511,256)
Page 1			Page 1	
(0,511)	(255,511)		(0,511)	(511,511)
		1FFFFH		

**HMMM****V1.2****VDP\_GRAPH2**

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

High speed from VRAM to VRAM

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

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

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

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

<i><b>HMCM_SC8</b></i>	<b>VDP_GRAPH2</b>
<b>void HMCM_SC8 ( int X1, int Y1, int X2, int Y2, void *tobuffer, unsigned char OP )</b>	
<p>High speed Copy the rectangle (<i><b>X1,Y1</b></i>)-(<i><b>X2,Y2</b></i>) from VRAM to the RAM buffer <i><b>*tobuffer</b></i>. Only for Screen 5 mode. <i><b>OP</b></i> is a standard logical operator. Use this command only with screen mode 8.</p>	

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

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

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

<i>HMMV</i>	V1.2	VDP_GRAPH2
<b>void HMMV( int XS, int YS, int DX, int DY, char COLOR)</b>		
<p>High speed filling of a rectangle box.</p> <p>Rectangle top left corner is defined <i>at XS,YS</i> its length is <i>DX</i> pixels, its height is <i>DY</i> pixels. The color to use is defined by <i>COLOR</i></p> <p>No logical operation allowed.</p> <p>When working on screen 5 or 7, HMMV will fill 2 horizontal pixels at the same time. The <i>COLOR</i> variable must be divided into two blocks of 4 bits example <i>COLOR I: 0bAAAA BBBB</i> . The left 4 bits will be used for the left pixel color, and the 4 right bits will be used for right pixel.</p> <p>If you do not need this feature, your <i>COLOR</i> variable can be calculated by this formula :</p> <pre>color =12;                // use color 12 color=((color &lt;&lt; 4)   color);    // Use color 12 for both pixels</pre>		

<i>LMMV</i>	V1.2	VDP_GRAPH2
<b>void LMMV( int XS, int YS, int DX, int DY, char COL, unsigned char OP)</b>		
<p>High speed fill of a rectangle box, with logical operation.</p> <p>Rectangle top left corner is defined <i>at XS,YS</i> its length is <i>DX</i> pixels, its height is <i>DY</i> pixels. The color to use is defined by <i>COL</i>, and the logical operation is <i>OP</i>.</p>		

**SPRITES**  
**[vdp SPRITES.h]**

<i>SpriteOn</i>	VDP_SPRITES
<b>void</b> <b>SpriteOn( void )</b>	
Enables Sprites	

<i>SpriteOff</i>	VDP_SPRITES
<b>void</b> <b>SpriteOff( void )</b>	
Disables Sprites	

<i>Sprite8</i>	VDP_SPRITES
<b>void</b> <b>Sprite8( void )</b>	
Sets sprites size to 8x8 pixels pattern mode	

<i>Sprite16</i>	VDP_SPRITES
<b>void</b> <b>Sprite16( void )</b>	
Sets sprites to 16x16 pixels pattern	

<i>SpriteSmall</i>	VDP_SPRITES
<b>void</b> <b>SpriteSmall(void)</b>	
Sets normal pixel sprite size	

<i>SpriteDouble</i>	VDP_ SPRITES
<b>void    SpriteDouble(void)</b>	
Sets double pixel sprite size	

<i>SpriteReset</i>	VDP_ SPRITES
<b>void    SpriteReset(void)</b>	
Resets all sprites attributes and patterns	

<i>SpriteCollision</i>	VDP_ SPRITES
<b>char    SpriteCollision(void)</b>	
Returns 1 in case of a sprite collision	

<i>SpriteCollisionX</i>	VDP_ SPRITES
<b>char    SpriteCollisionX(void)</b>	
Return <b>X</b> position of a sprite collision. (Available only for MSX2 and upper)	

<i>SpriteCollisionY</i>	VDP_ SPRITES
<b>char    SpriteCollisionY(void)</b>	
Returns <b>Y</b> position of a sprite collision (Available only for MSX2 and upper)	



***SetSpritePattern*** **VDP\_SPRITES**

```
void SetSpritePattern( char pattern_n, char* p_pattern, char s_size )
```

Sets the *pattern\_n* with *\*p\_pattern* data. *s\_size* is number of line of the pattern.

***PutSprite*** **VDP\_SPRITES**

```
void PutSprite( char sprite_n, char pattern_n, char x, char y, char color)
```

Puts the *sprite\_n* on screen with the defined pattern (*pattern\_n*) at position X and Y and color *color*.

On MSX2 and upper you must define the sprite color with *SC5SpriteColor* or *SC8SpriteColor* functions.

***Sprite32Bytes*** **VDP\_GRAPH2**

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

Uses to define 16x16 patterns. It convert a standard sprite (1-4quadr.) definition 32 char bytes

16 x 16-bit integers as bits (0b1111111100000000)

***SC5SpriteColors*** **VDP\_SPRITES**

```
void SC5SpriteColors( int spriteNumber, unsigned char *data)
```

On MSX2 Screen mode 5. Defines each line of the sprite *spriteNumber* with a specific color, set in *\*data*.

***SC8SpriteColors*** **VDP\_SPRITES**

```
void SC8SpriteColors( int spriteNumber, unsigned char *data );
```

On MSX2 Screen mode 8. Defines each line of the sprite *spriteNumber* with a specific color, set in *\*data*.

## **CIRCLE**

### **[VDP\_CIRCLE]**

Graphic functions to draw circle on MSX2 graphic screen.  
Can be use in complement of VDP\_GRAPH2.H or VDP\_GRAPH1.H

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

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

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

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

## **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.

<i><b>InitRamMapperInfo</b></i>	<b>RAMMAPPER</b>
<b>void InitRamMapperInfo(unsigned char deviceId)</b>	
<p>Initialization of the MSX2 Mapper device.  <i>DeviceId</i> must be set to 0x04 to initialize MSX-DOS2 Mapper.  <b>InitRamMapperInfo(4) ;</b>            After Initialization, the structure is set with all data and information about all mappers found.            typedef struct {                unsigned char slot ;                unsigned char number16KBSegments ;                unsigned char numberFree16KBSegments ;                unsigned char numberAllocatedSystem16KBSegments ;                unsigned char numberUser16KBSegments ;                unsigned char notInUse0 ;                unsigned char notInUse1 ;                unsigned char notInUse2 ;            } MAPPERINFOBLOCK ;</p>	

<i><b>Get_PN</b></i>	<b>RAMMAPPER</b>
<b>unsigned char Get_PN (unsigned char page)</b>	
<p>Gets and returns Segment Address of a Memory <i>page</i>.            page must be 0,1,2 or 3</p>	

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

<i>AllocateSegment</i>	RAMMAPPER
<b>SEGMENTSTATUS *AllocateSegment(unsigned char segmentType, unsigned char slotAddress)</b>	
<p>Allocates next available 16Kbytes ram segment and returns information about this segment in the Status structure.</p> <pre>typedef struct {     unsigned char allocatedSegmentNumber ;     unsigned char slotAddressOfMapper ;     unsigned char carryFlag ; } SEGMENTSTATUS ;</pre> <ul style="list-style-type: none"> <li>- segmentType must be 0 (Allocation of a user Segment)</li> <li>1, means System segment</li> <li>- slotAddress must be set to 0 for automatic allocation</li> </ul> <p><b>AllocateSegment(0,0) ;</b></p>	

<i>FreeSegment</i>	RAMMAPPER
<b>*FreeSegment(unsigned char segmentType, unsigned char slotAddress)</b>	
Free an allocated segment	

**PSG**

**[PSG.H]**

Extended function to control PSG

<i>Sound</i>	PSG
<b>void</b>	<b>Sound(char reg, char value)</b>
Writes a value into a register of PSG (0 to 13)	

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

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

<i>SilencePSG</i>	PSG
<b>void    SilencePSG()</b>	
Plays off all three PSG Channels	

<i>GetSound</i>	PSG
<b>char    GetSound(char reg)</b>	
Reads a PSG Register	

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

<i>SetNoisePeriod</i>	PSG
<b>void    SetNoisePeriod(char period)</b>	
Sets Noise Period. Period must be between 0 and 31	

<i>SetEnvelopePeriod</i>	PSG
<b>void    SetEnvelopePeriod(unsigned int period)</b>	
Sets Envelope Period. Period must be between 0 and 65535	

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

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

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

***AYFX PLAYER*****[ayfx\_player.h]**

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

<b><i>InitFX</i></b>	<b>ayfx_player</b>
<b>void</b>	<b>InitFX (void)</b>
Initialization of the ayFX player	

<b><i>PlayFX</i></b>	<b>ayfx_player</b>
<b>char</b>	<b>PlayFX ( unsigned char nfx)</b>
Plays the sound-FX <i>nfx</i> from the Sound bank.	

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

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



<i>FreeFX</i>		ayfx_player
<b>void</b>	<b>FreeFX (void)</b>	
Free the Memory used by the ayFX player.		

## **MUSIC PT3 REPLAYER**

### **[PT3REPLAYER.H]**

Use those function to play PT3 music files.

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

<i><b>PT3Play</b></i>	<b>ayfx_player</b>
<b>void    PT3Play(void)</b>	
Actualizes the music playing, inside a main loop.	

<i><b>PT3Rout</b></i>	<b>ayfx_player</b>
<b>void    PT3Rout(void)</b>	
Prepares data to be played. (Part of the Playing routine), inside a main loop.	

<i><b>PT3Mute</b></i>	<b>ayfx_player</b>
<b>void    PT3Mute(void)</b>	
Mutes the music.	

## The C standard functions (included in SDCC package)

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

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

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

**STDIO.H**

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

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

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

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

**STDLIB.H**

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

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



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

**STRING.H**

	Library Variable & Description
1	<b>size_t</b> This is the unsigned integral type and is the result of the <b>sizeof</b> keyword.

	Library Macro & Description
1	<b>NULL</b> This macro is the value of a null pointer constant.

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

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

## FUSION-C

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

**TIME.H**

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

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

**STDARG.H**

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

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



## **Publish and distribute your game**

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

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



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

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



Example of games sold on [www.Repro-Factory.com](http://www.Repro-Factory.com) :



Muffie's Tutankham &  
Conversion Classic...

30,00 €

In Stock



Joust - Arcade

38,00 €

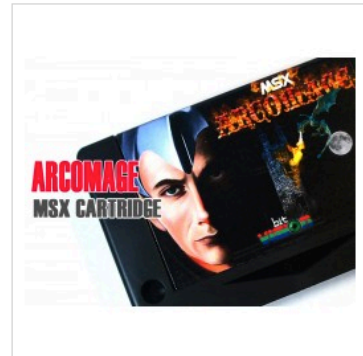
In Stock



GyruSS - Arcade

38,00 €

In Stock



Do not hesitate to contact REPROFACTORY on [www.repro-factory.com](http://www.repro-factory.com)

FUSION-C

## **Contacts**

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

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

**`ericb59@ebsoft.fr`**