

# **INSPECTA**

The realtime frame grabber for the PCI-Bus

Software Reference  
Rev. 1.38

<b>1</b>	<b>GENERAL</b> .....	<b>4</b>
1.1	Scope of this Manual .....	4
1.2	Revisionhistory.....	4
1.3	Trademarks .....	4
<b>2</b>	<b>SOFTWARE</b> .....	<b>5</b>
2.1	<b>Drivers</b> .....	<b>5</b>
2.1.1	Installation of INSPECTA drivers for Windows9x.....	6
2.1.2	Installation of INSPECTA for Windows NT .....	8
2.1.2.1	Definition of image memory, compatibility mode .....	8
2.1.2.2	Definition of image memory, „maxmem“ mode .....	8
2.1.2.3	Registry.....	9
2.1.3	Multiple Inspectas and WinNT .....	10
2.1.3.1	Multiple Inspectas, WinNT in compatibility mode.....	10
2.1.3.2	Multiple Inspectas, WinNT in „maxmem“ mode.....	10
2.1.4	Installation of INSPECTA-2 for DOS Extender .....	12
2.1.4.1	Definition of image memory.....	12
2.2	<b>Level1 Funktionen</b> .....	<b>14</b>
2.2.1	mvfg_open ( pcCameraProfile, DeviceNumber ).....	14
2.2.2	mvfg_setparam ( pcParamName, pcParamValue, DeviceNumber ).....	15
2.2.3	mvfg_getparam ( pcParamName, pValueBuffer, DeviceNumber ).....	17
2.2.4	mvfg_grab ( iCommand, DeviceNumber ).....	18
2.2.5	mvfg_getbufptr ( DeviceNumber ).....	20
2.2.6	mvfg_errmessage ( iCode ) .....	21
2.2.7	mvfg_close ( DeviceNumber ).....	22
2.3	<b>Level0 Functions</b> .....	<b>23</b>
2.3.1	INSPECTA-Initialization .....	23
2.3.1.1	mvfg_test (), mfgtest() .....	24
2.3.1.2	mvfg_isr (irqmode).....	25
2.3.1.3	vmfg_isr (int_mode).....	25
2.3.1.4	mfg_int (int_mode).....	26
2.3.1.5	mvfg_maskint (onoff, mask), mfg_maskint .....	26
2.3.1.6	mvfg_datpnt (), mfg_datpnt () .....	27
2.3.1.7	mvfg_alloc( void ); .....	28
2.3.1.8	mvfg_lock( HGLOBAL sel); .....	28
2.3.1.9	mvfg_unlock( HGLOBAL sel ); .....	29
2.3.1.10	mvfg_size ( HGLOBAL sel );.....	29
2.3.1.11	mvfg_contWriteInit ( DWORD PhysAddr );.....	29
2.3.2	Camera selection .....	30
2.3.2.1	mvfg_modcam ( mode ), mfg_modcam ( mode ); .....	31
2.3.2.2	mvfg_camsel (camnr), mfg_camsel (camnr).....	32
2.3.2.3	mvfg_digsel (camnr).....	32
2.3.2.4	mvfg_whitelevel (whitelevel), mfg_whitelevel (whitelevel).....	33
2.3.2.5	mvfg_blacklevel (blacklevel), mfg_blacklevel (blacklevel) .....	33
2.3.2.6	mvfg_synclevel (synclevel), mfg_synclevel (synclevel).....	33
2.3.2.7	mvfg_blank (blank_time) , mfg_blank (blank_time) .....	34
2.3.2.8	mvfg_pal (0/1), mfg_pal (0/1) .....	34
2.3.3	Memory-management.....	35
2.3.3.1	mvfg_black (black_lines), mfg_black (black_lines).....	38
2.3.3.2	mvfg_blackend (black_linesend), mfg_blackend (black_linesend) .....	38

2.3.3.3	mvfg_hstart, h_start (linelen, numlin, interlace, req_frm) .....	39
2.3.3.4	mvfg_selfframe (framenr), mfg_selfframe (framenr) .....	40
2.3.3.5	mvfg_PhysBuffer (*framestruc), mfg_PhysBuffer (*framestruc).....	41
2.3.3.6	mvfg_input ( DWORD timeout ) .....	42
2.3.3.7	mvfg_inputEx( int flag, DWORD timeout ) .....	42
2.3.3.8	mvfg_get_event (DWORD event_id ) .....	43
2.3.3.9	mvfg_set_vflag ( int flag ); .....	44
2.3.3.10	mvfg_get_vflag ( int iFlag );.....	44
2.3.3.11	mvfg_xchg (), mfg_xchg ().....	45
2.3.3.12	mvfg_ActualDmaPointer (), mfg_ActualDmaPointer().....	45
2.3.3.13	m(v)fg_DefineNextImage ( ImageNr, ShutterTime, DoubleBuffer) .....	46
2.3.4	Cameracontrol .....	48
2.3.4.1	mvfg_startstop ( DWORD flag ); mfg_startstop ( DWORD flag) .....	48
2.3.4.2	mvfg_stat (), mfg_stat () .....	49
2.3.4.3	mfg_fdvhi ().....	50
2.3.4.4	mfg_fdvlo ().....	50
2.3.4.5	mvfg_photo (time), mfg_photo (time) .....	50
2.3.4.6	mvfg_ScanPeriod (ScanPeriod), mfg_ScanPeriod (ScanPeriod).....	52
2.3.4.7	mvfg_contWrite ( DWORD NrOfFrames, DWORD Circular ); .....	53
2.3.4.8	m(v)fg_Snap ( DWORD mode, DWORD stop/exchange ); .....	54
2.3.4.9	mvfg_GrabberSelect ( DWORD number ); .....	55
2.3.4.10	mvfg_Linescan ().....	55
2.3.5	Display .....	57
2.3.5.1	init928 (gmode) .....	58
2.3.5.2	palette928 (palette) .....	58
2.3.5.3	mfg_clip (onoff) .....	59
2.3.5.4	ms3_setclipwindow (x, y, w, h) .....	59
2.3.5.5	mfg_setcolor (foreground, background) .....	60
2.3.5.6	mfg_fill (x, y, w, h, ram).....	60
2.3.5.7	mfg_c ( x, y, width, height, bitmap, ram).....	61
2.3.5.8	ms3_line (npoint, xyv, ram, plmode).....	61
2.3.5.9	ms3_pblt (xs, ys, smask, xd, yd, dmask, width, height, ram).....	61
2.3.5.10	vmfg_put2win (ptr, x, y, width, takepix, height, takeline, pitch, ram) .....	62
2.3.5.11	vmfg_put4win (ptr, x, y, width, multx, height, multy, pitch, ram) .....	62
2.3.5.12	linadrs3 () .....	63
2.3.5.13	mvfg_dmawin (x, y, width, height, pointer, pitch, color).....	63
2.3.5.14	linends3 ().....	64
2.3.6	Miscellaneous functions .....	64
2.3.6.1	mvfg_IntCallback (), mfg_IntCallback .....	64
2.3.6.2	mvfg_IntSource (), mfg_IntSource () .....	65
2.3.6.3	mfg_sync ().....	66
2.3.6.4	mvfg_chkclk (), mfg_chkclk (), .....	67
2.3.6.5	mvfg_ppin (), mfg_ppin (), .....	67
2.3.6.6	mvfg_ppout (dout), mfg_ppout (dout).....	67
2.3.6.7	mfg_bmp (parameters).....	68
2.3.6.8	m(v)fg_SetVideoClock (frequency) .....	68
2.3.6.9	mvfg_get_info( Z_MVFG_INFO * mvfg_info ) .....	69
2.3.7	Testfunctions .....	70
2.3.7.1	palette ().....	70
2.3.7.2	mvga (alt_frame_start, frame_start, vga_start, granularity) .....	70
2.3.7.3	palette13 .....	71
2.3.7.4	m13vga ( ptr, vga_ptr, granularity ).....	71

# **1 General**

## **1.1 Scope of this Manual**

This manual is written for the experienced programmer. It describes the use of the MVFG software library for Windows 3.11/95 and Phar Lap Dos-Extender

For an in-depth description of the INSPECTA hardware refer to the Hardware Reference Manual.

Appendix A shows a complete list of the I/O ports and the functions of their bits. Together with the assembler-source-code the INSPECTA hardware can be programmed on the hardware function level.

Differences between INSPECTA 1 and INSPECTA 2 are mentioned in the appropriate chapters.

## **1.2 Revisionhistory**

This manual describes the software:

PCAM.EXE ; starting Apr. 96

MPFGDRV.ASM ; starting Apr. 96

Information presented in this publication has been carefully checked for reliability; however, no responsibility is assumed for inaccuracies. The information contained in this document is subject to change without notice.

## **1.3 Trademarks**

All brand and product names which appear in this manual may be trademarks or registered trademarks of the corresponding companies.

Intel, the Intel Inside logo, Pentium® are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries, and are used under license.

## 2 Software

### 2.1 Drivers

INSPECTA-2 is delivered with three floppy-discs for DOSX, Win95 and WinNT. The latest version is also available at: <http://www.mikrotron.de>

The DOSX disc contains the following modules which are archived in the file: mpfgxxx!.exe (xxx = versionnumber)

MPFG EQU	ASM	cameraparameters and internal definitions
INSP2LIB	LIB	Phar-Lap INSPECTA-2 library
MVFGDRV	LIB	Win3.1 link-library for applications
MVFGD32	LIB	Win95, WinNT link-library for applications
ICAM	EXE	DOSX testprogramme for INSPECTA-2
INSP2	386	Win3.1/95 device driver for INSPECTA-2
MVFGDRV	DLL	Win3.1 DLL
MVFGD95	DLL	Win95 DLL
MVFGD32	DLL	Win95 DLL
MVFGDRV	H	Win 3.1/95/NT definitions
MVFG	H	DOSX definitions
MVFGFNT	H	DOSX font-definitions
MVFGTEXT	H	DOSX text-definitions

In addition there are the latest revisions of the manuals:

inspmad!	exe	Inspecta-2 manuals, german, Winword 6/7
inspmae!	exe	Inspecta-2 manuals, english, Winword 6/7

A setup disc for Windows95 and Windows NT contains all necessary files for the operating system and also the example programs.

The drivers source code is the same for all operating systems and written in 80486/Pentium 32 Bit assembler. Assembler switches distinguish between the operating systems.

### 2.1.1 Installation of INSPECTA drivers for Windows9x

There is a setup for installation of Windows9x drivers, which will copy the files to the following directories:

```

\Windows\insp2.386
\Windows\mvfgdrv.dll
\Windows\mvfgd95.dll
\Windows\mvfgd32.dll
\Programfiles\Inspecta-2: VCAM95.EXE, Kamera Profile, RESMEM.EXE
\Programfiles \Inspecta-2\doc\Rev95_dos.txt: readme
\Programfiles \Inspecta-2\include\mvfgdrv.h: Windows prototypes
\Programfiles \Inspecta-2\Lib\mvfgdrv.lib: Windows31 Link Library
\Programfiles \Inspecta-2\Lib\mvfgd32.lib: Windows95 Link Library
\Programfiles \Inspecta-2\Win95\Input95: Input95 Project
\Programfiles \Inspecta-2\Win95\Vcam95: Vcam95 Project

```

As the INSPECTA has no private image memory, some system memory at the end of physical memory is excluded from system usage and reserved for exclusive usage as frame buffer. This will be done automatically when loading the VxD. The amount of excluded memory is configurable through an entry in SYSTEM.INI file.

Subsequent calls from the application software makes this memory accessible as ordinary main memory.

The INSP2.386 VxD is responsible for this task. The following entries are inserted by the setup procedure.:

```

?? String „device=INSP2.386“ within the section [386Enh] in the SYSTEM.INI file.
?? new section [INSP2]
?? Within this section keyword: BlockSize=400h ;size of frame-buffer in pages (hex).

```

```

[INSP2]
BlockSize=400h ; 400h*4096=4MB

```

```

[386enh]
.....
device=insp2.386

```

If an image memory greater than 4Mbyte is requested, value BlockSize has to be increased manually. Take into account that the image memory reduces the amount of memory that is managed by the operating system. The image memory is owned by the application that uses the INSPECTA driver.

The Windows memory managers are in general not capable of reserving a huge amount of physical continuous memory. Depending on the size of requested image memory, different strategies are necessary.

The device driver Insp2.386 has an automatic memory sizer to exclude frame buffer memory from the **end** of physical memory. In some rare configurations the memory sizer cannot find the end of physical memory. In those cases or if more than 16MB is to be reserved, start address of frame buffer can be determined by entries in SYSTEM.INI:

Example with 48MB main memory:

**[INSP2]**

BlockSize=1000h ; 1000h\*4096=16MB

XmsStart=2000h ; frame buffer starts at 32MB

**[386enh]**

.....

device=insp2.386

MaxPhysPage=1fffh ; Windows memory is restricted to 32MB

These changes have to be done manually.

## 2.1.2 Installation of INSPECTA for Windows NT

Use Windows NT Workstation 4.0 with Service Pack >=3.

Use the supplied Inspecta setup diskette which copies to the following directories:

```
\Winnt\system32\driver\mpfgnt.sys: Windows NT device driver  
\Winnt\system32\mvfgd32.dll: Windows NT dynamic link library
```

The mvfgd32.dll carries the same name and contains the same functions as the one for Windows9x. However it is different from the Windows9x DLL and must not be intermixed.

The same name was chosen to allow applications run with both operating systems without recompiling. (Provided that no NT specific functions are used.)

### 2.1.2.1 Definition of image memory, compatibility mode

Beginning with driver version 2.27 image memory is allocated in so called „compatibility mode“. For each Inspecta 8MB memory is reserved out of the „NonPagedMemoryPool“.

When using this mode, no manual intervention is necessary during setup procedure. Only the number of Inspectas used in one system is to be entered.

If existing installations are updated, existing Inspecta Registry Entries are not changed. Therefore the „Maxmem“ method of memory allocation will be still in effect.

### 2.1.2.2 Definition of image memory, „maxmem“ mode

From driver version greater 1.65 the MAXMEM switch in BOOT.INI reduces the amount of memory for the system to the value defined by MAXMEM:

```
/MAXMEM=32 ; system can access only 32 MB main memory.
```

The rest of the physical memory is then available for INSPECTA-2 hardware and the application that uses INSPECTA-2 driver. Main memory must be larger than the MAXMEM switch by at least 1MB

The setup programme asks for the value of MAXMEM. The difference between the amount of physical memory and MAXMEM is then the size of the image memory. MAXMEM must be a multiple of 16MB.

The setup procedure does not change the entry in the BOOT.INI file. This must be done by hand. BOOT.INI is hidden and read only. Use the DOS attrib command to allow editing.

```
cd \  
attrib boot.ini -r -h -s
```

---

Then add the switch /MAXMEM=Mbytes decimal to the end of line with the desired WinNT boot partition.

### 2.1.2.3 Registry

INSPECTA-2 uses the following keys:

HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\MpfgNt

with the below values (example):

```
BlockSize 0x00800000 ; requested length of image memory (example:8MB)
                  ; compatibility mode will return the amount of memory
                  ; the system has actually allocated.
DeviceID 0x00001234 ; Inspecta-2/3 Device ID, 0x00004D41 for Inspecta-4
MemStartPage 0x02000000 ; image memory starts at 32 MB 0 if compatibility mode
MemoryAllocationMode 00000000 ; compatibility mode =0, maxmem mode = 1
DeviceNumber ; 0 if first Inspecta,
              ; 1 for second Inspecta in Key MpfgNt1
              ; 2 for third Inspecta in Key MpfgNt2
              ; 3 for fourth Inspecta in Key MpfgNt3
```

Do not change any othe value!

### 2.1.3 Multiple Inspectas and WinNT

With Windows NT (**WinNT Only!**) up to four Inspecta can be installed in one PC.

#### 2.1.3.1 Multiple Inspectas, WinNT in compatibility mode

Equip your system with up to four Inspectas. Run Inspecta setup and enter the number of Inspectas when asked.

If Inspectas device drivers were previously installed, delete all occurrences of MpfgNt.sys, MpfgNt1.sys, MpfgNt2.sys, MpfgNt3.sys, in \WinNT\system32\drivers and mvfg\*.dll in \WinNT\system32.

Delete an evtl. inserted /MAXMEM switch in file boot.ini.

In Registry delete all occurrences of keys:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\MpfgNt
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\MpfgNt1
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\MpfgNt2
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\MpfgNt3
```

Then run Inspecta setup and enter the number of Inspectas when asked.

#### 2.1.3.2 Multiple Inspectas, WinNT in „maxmem“ mode

Basic installation is done as described above. Some additional changes are then necessary:

Directory:

WinNT\system32\drivers

Copy mpfgnt.sys to mpfgnt1.sys and mpfgnt2.sys etc., so that as many copies as Inspectas exist.

Then in WinNT Registry (Regedit.exe) export the key:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\MpfgNt
```

to disk.

Then rename the existing key to:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\MpfgNt1
```

Now import the previously exported key from disk.

Repeat this procedure until as many keys as Inspectas exist.

Now change MemStartPage and BlockSize parameters in all keys so that:

MemStartPage of the first driver corresponds to the begin of image memory (as set with /MAXMEM switch in boot.ini) and MemStartPage of the next driver corresponds to the begin of image memory plus length given by BlockSize. Make shure that BlockSize times number of Inspectas is not greater than total image memory size.

Now change Devicenumber parameters in all keys so that the Devicenumber of the first key is 0, next key 1 etc.

Example for two Inspectas with 16MB total image memory, starting at 48MB

Key MpfgNT:

MemStartPage=0x03000000 ; 48MB image buffer start address  
BlockSize=0x00800000 ; 8MB image buffer length  
DeviceID=0x00001234  
Devicenumber=0x00000000 ; for first Inspecta

**Key MpfgNT1:**

MemStartPage=0x03800000 ; 56MB image buffer start address  
BlockSize=0x00800000 ; 8MB image buffer length  
DeviceID=0x00001234  
Devicenumber=0x00000001 ; for second Inspecta

Do not change any other value!

## 2.1.4 Installation of INSPECTA-2 for DOS Extender

The DOSX diskette has no separate setup programme. The MPFGxxx!.EXE is an archive which should be expanded and copied to the following directory structure. (to be compatible with Windows)

```

\Programme\Inspecta-2\DOC\REVxxx.TXT;Revisiondescription
\Programme\Inspecta-2\MPFGEQU.ASM           ;Cameraparameters and internal definitions
\Programme\Inspecta-2\INSP2LIB.LIB         ;Phar-Lap INSPECTA-2 Library
\Programme\Inspecta-2\ICAM.EXE             ;DOS testprogramme for INSPECTA-2
\Programme\Inspecta-2\MVFG.H               ;DOSX headers
\Programme\Inspecta-2\MVFGFNT.H           ;DOSX font-definitions
\Programme\Inspecta-2\MVFGTEXT.H          ;DOSX text-definitions
\Programme\Inspecta-2\MPFGxxx!.EXE         ;these files
\Programme\Inspecta-2\DOC\INSPMAD!.EXE     ;INSPECTA-2 manuals Winword6 german
\Programme\Inspecta-2\DOC\INSPMAE!.EXE     ;INSPECTA-2 manuals Winword6 english

```

### 2.1.4.1 Definition of image memory

With Phar Lap DOS Extender use the /INT15= switch with the HIMEM.SYS extended memory manager.

Example:

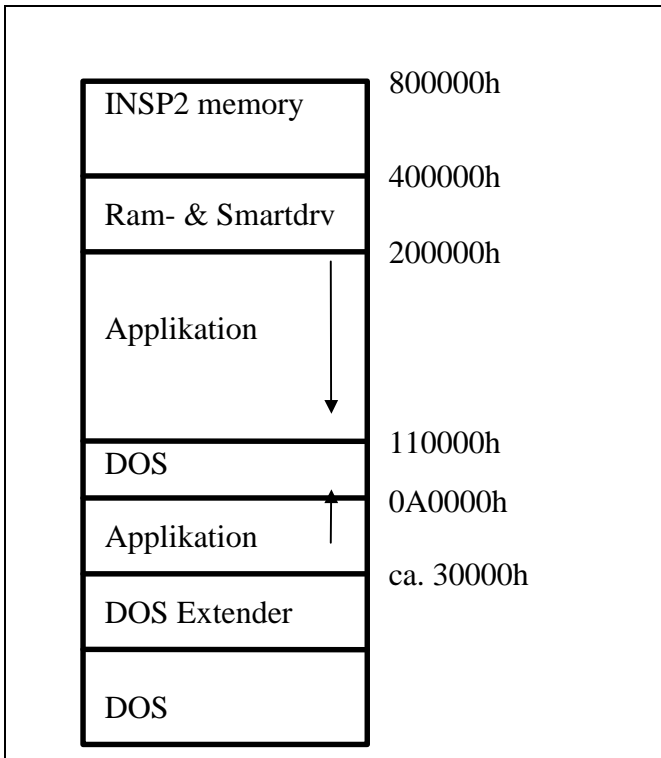
```
DEVICE=C:\DOS\HIMEM.SYS /INT15=XXXX
```

The value XXXX = main memory (excluding MVFG image memory) in Kbytes - 1088 - [length of SMARTDRV in KB] - [ length of RAMDRIVE in KB].

TNT uses extended memory starting with address XXXX+1088 (KB) downwards.

If SMARTDRV is used, the Phar-Lap exe's TNT, 386ASM, 386LINK, 386LIB, and HCD3861.EXE, HCD3862.EXE must be configured with the switch -extlow 110000h. (with the Phar Lap configuration programme: CFIG386).

See also DOS-Extender documentation.



This is an example for 4MB main memory and 2\*2MB INSP2 memory. There is also 1MB SMARTDRV and 1MB RAMDRIVE installed.

Use following entries in CONFIG.SYS:  
 DEVICE=C:\DOS\HIMEM.SYS /INT15=1024  
 DEVICE=C:\DOS\RAMDRIVE.SYS 1024 /E

....  
 and in AUTOEXEC.BAT:  
 LH C:\DOS\SMARTDRV.EXE 1024

....  
 Configure TNT DOS-Extender as follows:  
 CFG386 -clear TNT -extlow 110000h

## 2.2 Level1 Funktionen

With the Level1 API, only a few functions are necessary to initialize the Inspecta, select a specific camera and obtain an image. Level1 functions are implemented in WinNT/2k only.

### 2.2.1 mvfg\_open ( pcCameraProfile, DeviceNumber )

**Synopsis:** LONG mvfg\_open ( char \* pcCameraProfile, LONG DeviceNumber)

**Description:**

The selected Inspecta (one of four, only WinNT/2k) is initialized and a camera according to the selected configuration file and section within that configuration file is chosen. If a camera-string-file is indicated in the configuration file, the bytes in this file are send to the camera via the Inspecta 4D/4C serial interface. It is recommended to use the InspectaXX.cam configuration file which is maintained by the interactive camera setup tool VCAM95.EXE.

**Parameters:**

Entry:	char * pcCameraProfile	pcCameraProfile points to a string, containing the name of the camera configuration file and the specific section within this file. (e.g.: "Inspecta-3.cam;TestMode") Name and section are seperated by ;
	LONG DeviceNumber	Grabber-Number (0 to 3)
Result:	MVFG_OK	Init successful
	EMVFG_NO_VXD	Error: could not init Inspecta
	EMVFG_CAMFILE_NOTFOUND	Error: config file not found
	EMVFG_CAMSECTION_NOTFOUND	Error: section in config file not found
	EMVFG_CAMSTRG_FILE_NOTFOUND	Error: camera-string-file indicated but not found
	Sonstige	General error
Exit:		

**Remarks:** Use at least Driver release 3.00

### 2.2.2 mvfg\_setparam ( pcParamName, pcParamValue, DeviceNumber )

**Synopsis:** LONG mvfg\_setparam ( char \* pcParamName, char \* pcParamValue,  
LONG DeviceNumber )

**Description:**

mvfg\_setparam changes specific Inspecta parameters after being initialized by mvfg\_open ().

**Parameters:**

char * pcParamName	Parameter name to be changed
char * pcParamValue	Parameter value
LONG DeviceNumber	Grabber-Number (0 to 3)
Possible consts. for pcParamName:	
MVFGPAR_CAMMODE	Magic number for a specific camera
MVFGPAR_LINELEN	Length of one line in bytes
MVFGPAR_NUMLIN	number of lines of one image
MVFGPAR_BLACKLINESBEGIN	number of lines at the beginning of the image which are not for use
MVFGPAR_BLACKLINESEND	number of lines at the end of the image which are not for use
MVFGPAR_BLANKTIME	pixel without image-data between two lines
MVFGPAR_WHITELEVEL	white level "0" .... "255"
MVFGPAR_BLACKLEVEL	black level "0" .... "255"
MVFGPAR_VIDEONORM	videonorm ( "1" = CCIR, "0" = SECAM )
MVFGPAR_INTERLACED	"1" = interlaced, "0" = non-interlaced
MVFGPAR_REQFRAME	number / position of the images to grab
MVFGPAR_CAMSEL	"0" .... "7": camera input
MVFGPAR_DIGSEL	camera number at a multiplexer
MVFGPAR_PHOTO	"0" .... "n" shutter time
MVFGPAR_PHOTOFLAG	enable async shutter
	"0" disabled
	"1" enabled
MVFGPAR_SCANPERIOD	Scanperiod of a linescan camera
MVFGPAR_CONTINUOUSFLAG	Grab behaviour
	"0": Stop after grab
	"1": Continuous grab (only if camera is not shuttered)

MVFGPAR_EXTPHOTOFLAG	Flag for asynchronous shutter triggered by an external signal after being
	"0": Disable the external signal.
	"1": Enable the external signal.
MVFGPAR_TIMEOUT	Timeout in milliseconds
MVFGPAR_ENCODERFLAG	Encoderflag for Linescan cameras
MVFGPAR_DIVIDER	Divider for Linescan cameras

**Returnvalues:**

MVFG_OK	Parameter has been set
EMVFG_NO_VXD	Error: Inspecta not initialized
EMVFG_CAMPARAM_BADVALUE	Error: Value not allowed for this parameter
EMVFG_CAMPARAM_UNKNOWN	Error: Parameter unknown
EMVFG_NOT_OPEN	Error: Not opened with mvfg_open



## 2.2.4 mvfg\_grab ( iCommand, DeviceNumber )

**Synopsis:** LONG mvfg\_grab ( DWORD iCommand, LONG DeviceNumber )

**Description:**

Start an exposure, check status and exchange image memory block if double-buffer (standard).

**Parameters:**

DWORD iCommand	Select behaviour of mvfg_grab ()	
LONG DeviceNumber	Grabber-Number (0 to 3)	
Possible constants for iCommand:		
GRAB_WAIT	Grab image, wait for done:	
	Returnvalues:	
	MVFG_GRAB_READY	Grab successful
	EMVFG_NO_VXD	Error: Inspecta not initialized
	EMVFG_TIMEOUT	Error: Grab not complete within timeout time.
	EMVFG_NOT_OPEN	Error: Not opened with mvfg_open
GRAB_NOWAIT	Grab image, do not wait for done:	
	Returnvalues:	
	MVFG_OK	Grab has been started
	EMVFG_NO_VXD	Error: Inspecta not initialized
	EMVFG_NOT_OPEN	Error: Not opened with mvfg_open
GET_STATUS	Check status after start of grab:	
	Returnvalues:	
	MVFG_GRAB_READY	Grab success
	MVFG_NOT_READY	Grab started, but not yet completed
	EMVFG_NO_VXD	Error: Inspecta not initialized
	EMVFG_TIMEOUT	Error: Grab not completed within timeout time
	EMVFG_NOT_OPEN	Error: Not opened with mvfg_open
GET_STATUS_WAIT	Wait for done after start of grab:	
	Returnvalues:	
	MVFG_GRAB_READY	Garb success
	EMVFG_NO_VXD	Error: Inspecta not initialized
	EMVFG_TIMEOUT	Error: Grab not completed within timeout time

---

	EMVFG_NOT_OPEN	Error: Not opened with mvfg_open
--	----------------	-------------------------------------

To get the pointer to the image data use the function `mvfg_getbufptr()`  
DeviceNumber is the number of the grabber (0 to 3)

```
char * pcBuffer = mvfg_getbufptr( DeviceNumber );
```

### 2.2.5 mvfg\_getbufptr ( DeviceNumber )

**Synopsis:** LONG mvfg\_getbufptr ( LONG DeviceNumber )

**Description:**

Returns a pointer to the latest image. This image has to be grabbed by mvfg\_grab() before.

**Parameters:**

Entry	LONG DeviceNumber	Grabber-Number (0 to 3)
Result	Pointer to the latest image	

### 2.2.6 mvfg\_errmessage ( iCode )

**Synopsis:** int mvfg\_errmessage ( int iCode )

**Description:**

This function handles a returnvalue on another MVFG-function (which can return error-codes) and shows a Windows-messagebox with the error. If iCode was MVFG\_OK, nothing happens.  
iCode is returned unchanged.

**Parameters:**

Entry	int iCode	Returnvalue of a Level1 function (not mvfg_getbufptr)
Result	iCode (unchanged)	

### 2.2.7 mvfg\_close ( DeviceNumber )

**Synopsis:** LONG mvfg\_close ( LONG DeviceNumber )

**Description:** Stop and deactivate driver.

**Parameters:**

Entry	LONG DeviceNumber	Grabber-Number (0 to 3)
Result	MVFG_OK	success
	EMVFG_NO_VXD	Error: Inspecta not initialized

## 2.3 Level0 Functions

Although they provide the same functionality, the function prefixes for Phar Lap DOS-Extender and Windows differ. Windows calls start with **mvfg\_**functionname. DOS-Extender calls start in general with **mfg\_**functionname. The following manual covers both descriptions. Some functions are available only with DOS-Extender ( e.g: display functions ), others only with Windows.

### 2.3.1 INSPECTA-Initialization

A basic initialization ( here with DOSX notation ) consists of the following functions:

```
mfgtest();           /* tests MVFG INSPECTA */
vmfg_isr(0x10);      /* the interrupt vector provided by PCI-BIOS is
connected with the MPFGDRV provided service routine */
mfg_int(0x410);      /* select VD falling edge as interrupt source */

/* software is now ready to grab frames, if they should be displayed, the display functions must be
set up. */

init928(0);          /* use VGA resolution 640 * 480 * 8 bpp */
palette928(0);       /* 8 Bit greyscale with 4 overlay colors */
mfg_clip(0);         /* clear any pending clip-window */

/* now functions to select input devices must be called. See the next section for further examples.
*/

mfg_modcam (0x51);   /* select internal grey scale as video source */
```

To terminate the application. the interrupt must be deselected.

```
mfg_int(0);          /* deselect PCI INTA */
vmfg_isr(0);         /* restore original interrupt vectors */
```

Conforming to this structure, the description of functions is divided into the sections: initialization, device selection, frame memory management, utilities and history

### 2.3.1.1 mvfg\_test (), mfgtest()

**Synopsis:** LONG mvfg\_test ( void );  
**Description:** This function initializes INSPECTA  
**Returns:** == 0 OK  
!= 0 Errorcode:  
-2 memory base address not found ; DOSX only  
-3 cant map phys -> lin ; DOSX only  
-6 memory banks do not exchange ; INSPECTA 1 only  
-8 no clock from self test  
-10 no frame data valid from self test  
-11 no line data valid from self test  
-13 WRONG CPU (386) ; DOSX only  
Any error stops further activity.

**Example:** LONG IMvfgError;  
  
if (IMvfgError = mvfg\_test())  
{  
  vsprintf( acBuffer, "Error %lh with initialization of INSPECTA",  
IMvfgError );  
  ...  
}

**Remarks:** Error codes -2, -3, and -6 can only occur in DOSX platforms.  
Error code -6 does not occur with INSPECTA 2.

### 2.3.1.2 mvfg\_isr (irqmode)

**Synopsis:** void mvfg\_isr ( DWORD irqmode );  
**Description:** this function defines interrupt status (on/off) and source

**Parameters:** Bits 0 - 7 = 0 : interrupt off  
Bits 0 - 7 > 0 : interrupt on  
Bits 8 - 15 interruptsource  
possible values for interrupt source see next section

**Returns:**

**Example:** /\* select VD falling edge as interrupt source and switch on  
the interrupt \*/  
mvfg\_isr ( 0x0400 | 0x10 );

**Remark:** windows only

---

### 2.3.1.3 vmfg\_isr (int\_mode)

**Description:** The PCI-BIOS defined IRQ is connected with the interrupt service routine in module MPFGDRV.

**Parameters:** int\_mode > 0 : vector for service routine installed  
int\_mode = 0 : previous vector restored

**Remark:** DOSX only

### 2.3.1.4 mfg\_int (int\_mode)

**Description:** mfg\_int selects interrupt source and activates/deactivates the INTA output from INSPECTA.

**Parameters:** 'int\_mode' can have the following values:

Bits:	0..7	= 0:	deactivate INTA
		> 0:	activate INTA
	8..9	= 0	
	10	= ISRC0	interrupt source 0
	11	= ISRC1	interrupt source 1
	12	= ISRC2	interrupt source 1
	13..32	= 0	

int_mode =	0x010	; activate hor. freq/128
	0x410	; activate falling edge of VSYNC analog
	0x810	; activate start/stop flipflop is set
	0xC10	; activate falling edge of HSYNC analog
	0x1410	; activate falling edge of VSYNC digital
	0x1C10	; activate falling edge of HSYNC digital

**Remark:** DOSX only

### 2.3.1.5 mvfg\_maskint (onoff, mask), mfg\_maskint

**Synopsis:** void mvfg\_maskint ( onoff, mask );

**Parameter:** onoff = 0: chosen interrupt will be masked.  
 onoff = 1: chosen interrupt will be enabled except short timeperiods defined by the driver.  
 mask: selected IRQ

**Description:** mfg\_maskint deselects or selects chosen system interrupts. When using camera control functions with a HSYNC defined timing, ( e.g: variable shutter ) this function assures that no higher prioritized interrupts distores the timing.

If the application enables a specific interrupt once (mvfg\_intmask (1, mask)), the driver software disables and reenables this interrupt automatically only when necessary.

If the application disables a specific interrupt (mvfg\_intmask (0, mask)), this will be done immediate. The application must then reenable this interrupt (mvfg\_intmask (1, mask )) as soon as possible.



---

### 2.3.1.7 mvfg\_alloc( void );

**Synopsis:** HGLOBAL mvfg\_alloc( void );

**Description:** This function finds a global memory handle for the MVFG - image memory.

**Returns:** != 0 memory handle.  
== 0 function failed.

**Example:** HGLOBAL MvfgHandle;

```
if ( !(MvfgHandle = mvfg_alloc()) )
{
    exit ( 1 );
}
```

**Remark:** Windows only

---

### 2.3.1.8 mvfg\_lock( HGLOBAL sel);

**Synopsis:** LPVOID mvfg\_lock( HGLOBAL sel);

**Description:** This function places a lock for the MVFG memory and returns a pointer to this area.

**Returns:** != 0 pointer  
== 0 function failed.

**Example:** LPVOID lpMvfgSpeicher;

```
if ( !( lpMvfgSpeicher = mvfg_lock( MvfgHandle )))
{
    exit ( 1 );
}
```

**Remark:** Windows only

---

---

### 2.3.1.9 mvfg\_unlock( HGLOBAL sel );

**Synopsis:** BOOL mvfg\_unlock( HGLOBAL sel );

**Description:** releases the Lock for the MVFG memory

**Returns:** == 0 lock released.  
!= 0 function failed.

**Example:** mvfg\_unlock ( MvfgHandle );

**Remark:** Windows only

---

### 2.3.1.10 mvfg\_size ( HGLOBAL sel );

**Synopsis:** DWORD mvfg\_size ( HGLOBAL sel );

**Description:** This function returns the size of the MVFG frame memory.

**Returns:** memory size in bytes.

**Example:** --

**Remark:** Windows only

---

### 2.3.1.11 mvfg\_contWriteInit ( DWORD PhysAddr );

**Synopsis:** mvfg\_contWriteInit ( DWORD PhysAddr);

**Description:** INSPECTA-2 is initialized for continuous recording of long sequences up to 4Gbyte. DWORD PhysAddr is physical start address of image memory. Image memory is physical contiguous memory. Use together with mvfg\_contWrite ( NrOfFrames, Circular).

**Returns:** --

**Example:** --

**Remark:** Use driver > 1.65 and INSPECTA-2 hardware Rev. >4, IMP Nr. >317. No multiplane camera modes. All lines except those while VSYNC active are recorded. Interlace even & odd fields are not combined to frames.

### 2.3.2 Camera selection

These functions define video-source, sync-source, clock-source and white and black levels. The frame format (linelength, number of lines etc ) are also covered in this section. Use a Panasonic WV CD-50 with intern tact and Sync ( both signals provided by INSPECTA ) as an example.

```

mfg_modcam(0x28);      /* analog camera without pixel clock and with
sync                  input*/

/* use the next 8 function calls only if the default settings from
mfg_modcam ( 0x28 ) need to be changed. */

mfg_camsel(0);        /* choose the upper D-connector, channel red as
input*/
mfg_whitelevel(192);  /* white level is 192 * 1,2V/256 = 0,9V */
mfg_blacklevel(32);   /* black level is 32 * 1,2V/256 = 0,15V */
mfg_synclevel(1);     /* Sync-level is 125 mV above sync tip, has
effect
*/
                        only, if a composite sync mode has been selected
mfg_blank (84);       /* blank time from HD falling edge to start of
line = 2*84 * Pclk = 11,84 usec */
mfg_black(26);        /* number of black lines at begin of frame = 26
*/
h_start(603,472,1,0); /* 603 == linelen, 472 == numlin: including
number
                        of black lines, 1 == interlace,
                        0 == req_frm : this frame is stored only on the
first
                        place in memory */
mfg_pal(1);           /* 625 lines / 50 Hz european format */

```

### 2.3.2.1 mvfg\_modcam ( mode ), mfg\_modcam ( mode );

- Synopsis:** LONG mvfg\_modcam ( DWORD mode );
- Description:** define video, sync and clock source, A/D converter levels and image format. The mode is usually unique for each camera.
- Parameters:** The table in APPENDIX B shows the possible parameter sorted by value and function.
- Returns:**
- = 0 OK
  - 1 unknown mode
  - 2 a camera with external clock was selected, but no clock was found.

**Example:** LONG IMvfgError;

```
if ( IMvfgError = mvfg_modcam ( 0x8C )) /* PULNIX 9700, digital */
{
    wsprintf( acBuffer, "no clock from TM9700");
    ...
}
```

mvfg\_modcam matches INSPECTA to the connected camera. Bits 4..30 of mode are a „magic number“ which represents a specific camera. Bits 0..3 are copied to the camera control bits mc0..mc3.

If bit 31 of mode is not set, the global variables dwords linelen, numlin, interlace and req\_frm are written with mode-specific values. The function h\_start which uses these variables is called to define the format of the frame.

The variable DWORD seq\_color defines the frame memory organisation according to the following table:

```
seq_color    ; 0: one plane
              ; 1: three consecutive pages, offset is defined by variable [pel_frm]
              ; 2: two frames interlaced (sony xc7500, Kodak ES 1.0)
              ; 3: two frames non-interlaced
```

The variable DWORD colour\_type defines the pixel representation in frame memory according to the following table:

```
colour_type  ; 1 = 3:3:2 8bpp, 1-plane, rgb
              ; 2 = 5:6:5 16bpp, 1-plane, rgb
              ; 3 = 8:8:8 8bpp, 3-planes, rgb
              ; 4 = 8:8:8:8 32bpp, 1-plane, xrgb
              ; 5 = 8:8:8 24bpp, 1-plane, rgb
              ; 6 = 16:16:16 48bpp, 1-plane, rgb
              ; 7 = 8:8:8:8 32bpp, 1-plane, 4-cameras b&w
              ; 8 = 8:8 16bpp, 1-plane, 2-cameras b&w
```

Also mfg\_blank, mfg\_whitelevel, mfg\_blacklevel and mfg\_sync are called.

### 2.3.2.2 mvfg\_camsel (camnr), mfg\_camsel (camnr)

**Synopsis:** void mvfg\_camsel ( DWORD camnr );

**Description:** 'mvfg\_camsel()' selects one of six (eight cameras with the additional high-speed A/D adapter MAD 1020) cameras as video, clock and sync source or destination.

camnr	Video	Connector
0	green	P1
1	blue	P1
2	red	P1
3	green	P1 (MAD1020)
4	green	P2
5	blue	P2
6	red	P2
7	green	P2 (MAD1020)

The definition of green input for camnr 3=0 and 7=4 changes if the additional high-speed A/D converter board is used with the INSPECTA-2. Camnr 3 or 7 selects the connector 1 or 2 on the add. adapter board.

With modes for multi-channel cameras ( e.g.: 0x78, 0xC0, 0xEC, 0x148, colorcameras ) camnr: 0..3 selects P1 and camnr: 4..7 selects P2.

**Returns:** --

**Example:** --  
mfg\_camsel writes the global variable mfg\_camnr.

### 2.3.2.3 mvfg\_digsel (camnr)

**Synopsis:** void mvfg\_whitelevel ( DWORD camnr );

**Description:** selects one of several digital cameras which are connected to the digital multiplexer MUX 5000. See the MUX 5000 documentation for the description of valid camnr.

**Returns:** --

**Example:** --

---

#### 2.3.2.4 mvfg\_whitelevel (whitelevel), mfg\_whitelevel (whitelevel)

**Synopsis:** void mvfg\_whitelevel ( DWORD level );  
**Description:** sets the whitelevel of the video A/D  
whitelevel = 1.2V/ 256 \* whitelevel  
**Returns:** --  
**Example:** --

---

#### 2.3.2.5 mvfg\_blacklevel (blacklevel), mfg\_blacklevel (blacklevel)

**Synopsis:** void mvfg\_blacklevel ( DWORD level );  
**Description:** sets the blacklevel of the video A/D  
blacklevel = 1.2V/256 \* blacklevel  
**Returns:** --  
**Example:** --

The video signal is clamped to 0 Volt during the front porch.

---

#### 2.3.2.6 mvfg\_synclevel (synclevel), mfg\_synclevel (synclevel)

**Synopsis:** void mvfg\_synclevel ( DWORD level );  
**Description:** defines the threshold for sync separation  
  
**Parameters:** 0 25 mV  
1 125 mV  
**Returns:** --  
**Example:** --

---

---

### 2.3.2.7 mvfg\_blank (blank\_time) , mfg\_blank (blank\_time)

**Synopsis:** void mvfg\_blank ( DWORD blank );

**Description:** the parameter blank\_time is used to skip black pixels at the begin of a line. It defines the number of pixelclocks \*2 to lengthen the internal HSYNC signal. It starts with the falling edge of the incomig HSYNC and is adjusted to stop with the first valid pixel of the line. Thus blank\_time defines the length of the stored line. As this length must be divisible by 4, there are blank\_time values that result in an misaligned image.

Blank\_time depends on the camera type and can be manually adjusted with the programme ICAM.EXE or VCAM95.EXE.

**Returns:** --

**Example:** --

---

### 2.3.2.8 mvfg\_pal (0/1), mfg\_pal (0/1)

**Synopsis:** void mvfg\_pal ( DWORD pal );

**Description:** defines number of lines per frame if an internal sync mode is selected or the generation of an internal VSYNC signal if a linescan mode is selected  
pal = 0 --> 525 lines, NTSC/RS-170 or 128 lines if linescan  
pal = 1 --> 625 lines, CCIR or 256 lines if linescan  
The above parameter description is valid for **INSPECTA-2 Rev. 4,5,6 with firmware code > IMP317.**

**Returns:** --

**Example:** --

### 2.3.3 Memory-management

Functions `mvfg_hstart()`, `h_start` and `mvfg_selframe()`, `mfg_selframe` und `mvfg_lines()`, `mfg_lines` define, how the video lines are stored in image-memory.

`mvfg_xchg()`, `mfg_xchg` flip image-memory.

`mvfg_input()` flip image-memory with the next vertical sync.

`mvfg_set_vflag()`, `mvfg_get_vflag()` set or read the `das_fgtv_valid` flag.

The `_fgtv_valid` flag controls the automatic flipping of image-memory within the VSYNC interrupt service routine. It is checked within the `mfg_sync` routine ( which is called through the interrupt service routine ) and if = 0 , the function `mfg_xchg` is called and `_fgtv_valid` is set to 1 again.

The following example is a full function demo programm for the MVFG frame grabbers:

```

/* module      demo.c
=====
example program for MVFG demonstration
=====*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <ctype.h>
#include <mvfg.h>

main (int argc, char * argv[])      /* MAIN PROGRAM
=====*/
{

printf ("Result of mfgtest: %d\n",  mfgtest());

mfg_camsel(0);      // select camera 0
mfg_modcam(0x68);  // analog interlace, composite sync
mfg_pal(0);        // NTSC frames, mfg_modcam defaults to CCIR

/* the following function calls are not necessary, because mfg_modcam defines
all values to fit to standard RS-170 camera.

mfg_blank(0x45);   // blank time
mfg_black(0x1c);  // number of black lines from vertical
                  // sync rising edge to first visible line

//h_start  (linelen,  numlin,  interlace,  req_frame);
h_start    (0x2d4,    0x25e,    1,          0);

*/

vmfg_isr(0x10);
mfg_int(0x410);    // use VSYNC falling edge as interrupt source

init928(0);        // select s3 mode 103 for 640 x 480 with
                  // 8 bit per pixel
mfg_clip(0);       // clear any clip window
palette928(0);     // b/w palette with 4 overlay colors for bit 0..1

```

```
while (!_kbhit ()) // as long as not key is hit
{
    _fgtv_valid = 0 ;           // set flag to image-memory flip within next
                                // vertical interrupt
    while( _fgtv_valid == 0 ); // wait until cleared by interrupt service

/* use funktion vmfg_put2win for display
   vmfg_put2win (ptr,      x, y, width, skippix, height, skipline, pitch,
ram)*/
   vmfg_put2win (mfg_lin, 0, 0, 640,  0,      480,  0,      0x2d8, 1);
}
mfg_int(0);           // deselect mvfg-interrupt
vmfg_isr(0);         // restore original interrupt-vector
init928(-1);         // select vga character mode before exit

}                       /* end "main" */
```

Example for multiframe-recording:

There are three analog RS-170 cameras synchronized by sync output from INSPECTA. A frame from each camera should be recorded one after the other. Recording should stop after the last frame. Frame memory should be flipped after the application has ended the previous calculations.

```

...
int status;
mfg_modcam (0x28);      // analoge camera, clock and sync to camera
mfg_camsel(0);         // select camera 0
mfg_selframe(0);       // write to image 0 in frame-memory
mfg_startstop(1);      // start recording

/* wait until the bits fdv == 0 and odd == 1 returned by mfg_stat,
   e.g: wait for next full frame */

    while (( status = ( mfg_stat() && 0xa00 ) == 0x800 );

/* now frame 0 is recording */

/* wait until the bits fdv == 0 and odd == 1 returned by mfg_stat,
   e.g: wait for next full frame */

    while (( status = ( mfg_stat() && 0xa00 ) == 0x800 );

/* now the image at position 0 is ready, select next camera and next image
position */

mfg_camsel(1);         // select camera 1
mfg_selframe(1);       // select image position 1

/* wait until the bits fdv == 0 and odd == 1 returned by mfg_stat,
   e.g: wait for next full frame */

    while (( status = ( mfg_stat() && 0xa00 ) == 0x800 );

/* now the image at position 1 is ready, select next camera and next image
position */

mfg_camsel(2);         // select camera 2
mfg_selframe(2);       // select image position 2

/* wait until the bits fdv == 0 and odd == 1 returned by mfg_stat,
   e.g: wait for next full frame */

    while (( status = ( mfg_stat() && 0xa00 ) == 0x800 );

/* now the image at position 2 is ready, stop further recording */
mfg_startstop(0);

....

/* flip image-memory to get access to the previous recorded sequence */
mfg_xchg();            // now frames are accessible

```

This form of sequence-recording has the disadvantage, that the frame immediately following the selection of the new camera cannot be recorded, because writing the line-address table ( mfg\_selframe(..) ) takes too much time.

The functions `m(v)fg_multisequence()`, `m(v)fg_multiframe()` avoid this problem, because all controlling is done in the background through the `mvfg` interrupt service routine, and the line-address table is set up for more than one frame.

### 2.3.3.1 `mvfg_black (black_lines)`, `mfg_black (black_lines)`

**Synopsis:** `void mvfg_black( DWORD black_lines );`

**Description:** Defines how many black-lines at the beginning of a frame will be written to the end of the frame-memory and thus not be stored to the actual frame.

Call this function **before** '`mvfg_hstart()`' !

Use the test-tool `PCAM.EXE` to find out, how many black lines an actual camera has.

**Returns:** --

**Example:** --

`mfg_black` writes the global variable `black_lines`. Funktion `h_start` uses it to store all black lines to the end of the frame-memory.

---

### 2.3.3.2 `mvfg_blackend (black_linesend)`, `mfg_blackend (black_linesend)`

**Synopsis:** `void mvfg_blackend ( DWORD black_linesend );`

**Description:** Defines how many black-lines at the end of a frame will be written to the end of the frame-memory and thus not be stored to the actual frame.

Call this function **before** '`mvfg_hstart()`' !

Use the test-tool `PCAM.EXE` to find out, how many black lines an actual camera has.

**Returns:** --

**Example:** --

`mfg_black` writes the global variable `black_lines`. Funktion `h_start` uses it to store all black lines to the end of the frame-memory.

### 2.3.3.3 mvfg\_hstart, h\_start (linelen, numlin, interlace, req\_frm)

**Synopsis:** void mvfg\_hstart (DWORD linelen, DWORD numlin,  
DWORD interl, LONG req\_frm);

**Description:** this function defines the format of the frame in memory and where in memory the actual frame is to be stored.

**Parameters:**

linelen:	Number of pixel per line.
numlin:	Number of lines per frame <b>including</b> lines that should not be stored (invisible lines, defined by 'mvfg_black()' and mvfg_blacklinesend()). INSPECTA does not count HSYNCS while VSYNC is low, but it counts all serration and equalizing pulses which are recovered from a composite video signal just before and after the VSYNC is low. Therefore numlin is usually less then the 525/625 lines expected from a standard RS-170 or CCIR video signal. Linescan cameras use either 128 or 256 lines depending on the parameter of mvfg_pal (videomode).
interl:	Type of frame 1 = interlaced, 0 = non interlaced.
req_frm:	Position in frame-memory, where the actual frame has to be stored. Each position consumes as many bytes as the variable pel_frm shows. The position number starts with 0. If req_frm == -1, line-table is set up to accomodate as many frames as fit into frame-memory. (max. 8192 lines per plane or 32768 lines if singleplane.) If req_frm == -2, -3, -4.... frame memory is set up for 1, 2, 3 ... consecutive frames.

The above parameter description is valid for **INSPECTA-2 Rev. 4,5,6 with firmware code > IMP317.**

Use the test-tool ICAM.EXE or VCAM95.EXE (PCAM.EXE for INSPECTA-1) to find the actual parameters of the selected camera.

**Returns:** --

**Example:** /\* define the geometry of a frame of a PULNIX TM9700 when using digital video. The TM9700 delivers non-interlaced video. Write only to position 0 in frame-memory. \*/

```
mvfg_hstart( 0x300, 0x20d, 0x0, 0x0 );
```

#### Global variables:

Calling-parameters linelen, numlin, interlace und req\_frm are stored to global variables with the same name.

The global variable dword `pel_frm` is calculated as:  $(= \text{linelen} * (\text{numlin} - \text{black}))$ , also the global variable dword `frame_nr` as:  $(= [\text{mfg\_len}] / \text{pel\_frm})$ , number of frames per image memory size)

The function `h_start` uses the global variable `black_lines`, to store unwanted lines at the begin of a frame to the end of frame-memory:  $(= [\text{mfg\_len}] - \text{linelen})$ . The Funktion `mfg_black` writes this variable, `h_start` processes it.

The function `h_start` uses the global variable `black_linesend`, to store unwanted lines at the end of a frame to the end of frame-memory:  $(= [\text{mfg\_len}] - \text{linelen})$ . The Funktion `mfg_blacklinesend` writes this variable, `h_start` processes it.

---

#### 2.3.3.4 `mvfg_selframe (framenr)`, `mfg_selframe (framenr)`

**Synopsis:**                `void mvfg_selframe ( LONG nr );`

**Description:**            `mvfg_selframe` uses `h_start` (see previous function ) in that only the parameter `req_frm` is passed. For all other parameters the contents of the appropriate global variables is used.

**Parameters:**            'framenr' must be: **0** <= **nr** <= **MaxFrames**.

If 'framenr' == -1, line-table is set up to accomodate as many frames as fit into frame-memory. (max. 2048 lines per channel)

**Returns:**                --

**Example:**                --

### 2.3.3.5 mvfg\_PhysBuffer (\*framestruc), mfg\_PhysBuffer (\*framestruc)

**Synopsis:** void mvfg\_PhysBuffer ( \*framestruc );

**Description:** mfg\_PhysBuffer(\*framestruc) includes within the structure \*framestruc all parameters necessary to define image format.

\*framestruc: +0 physical buffer address ( byte address )  
 +4 physical buffer length  
 +8 single/double buffer: 0/1  
 +12 linelen  
 +16 numlin  
 +20 interlace 0/1  
 +24 req\_frame -1, 0..frame\_nr  
 +28 blacklines  $\geq 0 < \text{numlin}$   
 +32 blacklinesend  $\geq 0 < (\text{numlin} - \text{blacklines})$   
 +36 seq\_color 0,1,2

double\_buffer: 0: only one image buffer  
 1: two image buffers, length = physical\_buffer/2

seq\_color: 0: single plane b&w  
 1: three planes for three b&w or one RGB camera  
 2: two planes interlaced into each other, for two-tap sensors like SONY XC-7500

**Returns:** --

**Example:** --

**Remark:** --

### 2.3.3.6 mvfg\_input ( DWORD timeout )

**Synopsis:** LONG mvfg\_input ( DWORD timeout );

**Description:** This function waits for the next VSYNC of one or as many full frames as defined in mvfg\_hstart with parameter req\_frm <0). Then it flips the frame-memory within the vertical service routine.  
If the 'timeout' counter expired before the VSYNC was encountered, the function returns with an error code. The internal timeout counter is the Windows msec timer. For compatibility, the parameter: (DWORD timeout) is expressed in microseconds.

**Returns:** == 0 success!  
== EMVFG\_TIMEOUT timeout!

**Example:**

```
if ( mvfg_input ( 0x100000 ) == EMVFG_TIMEOUT )
{
    wsprintf( acBuffer, "timeout !" );
    ...
}
```

**Remark:** Windows only

### 2.3.3.7 mvfg\_inputEx( int flag, DWORD timeout )

**Synopsis:** LONG mvfg\_inputEx( int flag, DWORD timeout )

**Description:** ,mvfg\_inputEx()' extends the funktion ,mvfg\_input()' with those controls described in function ,mvfg\_set\_vflag()'.  
While this function waits for an image to be ready, the associated thread is suspended, so that no CPU time is wasted.

int	flag	Same meaning as with ,mvfg_set_vflag()'
DWORD	timeout	Timeout to return if no video is present.
		<b>If a sequence of frames is grabbed, take care for long enough timeout!</b>
		This values is given in [? s] .

**Returns:** == MVFG\_OK image ready  
== EMVFG\_TIMEOUT Timeout happened

**Example:** mvfg\_inputEx( 3, 1000000 ); // grab image

**Remark:** this funktion is only available with WiNT/2K/XP

### 2.3.3.8 mvfg\_get\_event (DWORD event\_id )

**Synopsis:** Handle mvfg\_get\_event( DWORD event\_id )

**Description:** mvfg\_get\_event() returns the handle of an Inspecta events. Inspectas driver serves this event in its interrupt service routine and signals the status of the capture process. This event is a „manual reset event object“, which will be set but not reset by the driver.

Following Event-ID's are defined:

ID	Description
MVFGEVENT_VALID_FLAG	The event is set (signaled), as soon as an image is completely captured.

**Returns:**

!= NULL	Handle of the event.
== NULL	Handle not available.

**Example:**

```
// capture a frame with the Valid-Flag Event:

// get handle for the Valid-Flag Event
hValidFlagHandle = mvfg_get_event( MVFGEVENT_VALID_FLAG );

ResetEvent( hValidFlagHandle ); // clear Valid-Flag Event

mvfg_set_vflag( 3 ); // trigger input of next frame

// Wait for frame ready, timeout 1s
WaitForSingleObject( hValidFlagHandle, 1000 );
```

**Remark:** this funktion is only available with WiNT/2K/XP

### 2.3.3.9 mvfg\_set\_vflag ( int flag );

**Synopsis:** void mvfg\_set\_vflag ( int flag );

**Description:** This function sets the \_fgtv\_valid flag. This flag is checked in every VSYNC service routine.

Depending on the value for 'flag' the following action is taken:

int flag = 0 : image memory pointer is loaded pointing to the other memory block after the next VSYNC of a full frame.

int flag = 1 : no more action regarding the memory pointer or the grabbing process is taken.

int flag = 2 : image memory pointer is toggled pointing to the other memory block after every VSYNC of a full frame.

int flag = 3 : image memory pointer is loaded pointing to the other memory block after the next VSYNC of as many full frames as defined in mvfg\_hstart with parameter req\_frm <0.

int flag = 4 : grabbing is stopped after the next VSYNC of one or as many full frames as defined in mvfg\_hstart with parameter req\_frm <0.

**Returns:**

**Example:** See function 'mvfg\_get\_vflag ()'.

**Remark:** Windows only, for DOSX use the global variable DWORD \_fgtv\_valid directly.

### 2.3.3.10 mvfg\_get\_vflag ( int iFlag );

**Synopsis:** LONG mvfg\_get\_vflag ( int iFlag );

**Description:** Read the actual value of the \_fgtv\_valid flag.

**Returns:** == 1 action according to request parameter ( see: mvfg\_set\_vflag) is taken.  
 == 0,2,3,4 action not yet been taken.

**Example** mvfg\_set\_vflag ( MVFG\_VALID ); // request flipping on next VSYNC.  
 // wait for done

while ( !mvfg\_get\_vflag ( MVFG\_VALID ));

**Remark:** Windows only, for DOSX use the global variable DWORD \_fgtv\_valid directly.

---

**2.3.3.11** **mvfg\_xchg (), mfg\_xchg ()****Synopsis:** void mvfg\_xchg ( void );**Description:** This function flips frame-memory immediately.  
If this should be synchronized with a VSYNC, wait for VSYNC with mvfg\_stat () and the bit MVFG\_FRAME\_VALID. ( or better: use the mvfg\_set\_vflag(..) or the mvfg\_input () functions)**Returns:** --**Example:**

```
/* wait for VSYNC */
while (!(mvfg_stat ( void ) & MVFGS_FRAME_VALID ));

/* flip frame-memory */
mvfg_xchg();
```

---

**2.3.3.12 mvfg\_ActualDmaPointer (), mfg\_ActualDmaPointer()****Synopsis:** DWORD mvfg\_ActualDmaPointer( void );**Description:** This function reads the actual writeposition of the PCI-DMA and returns the next lower linenummer.**Returns:** actual write position modulo linelength.**Example:** --**Remark:** this function is available for INSPECTA-2 rev. > 1.50.

**2.3.3.13** **m(v)fg\_DefineNextImage ( ImageNr, ShutterTime, DoubleBuffer)**

**Synopsis:** `DWORD m(v)fg_DefineNextImage(DWORD ImageNr, DWORD ShutterTime, DWORD DoubleBuffer );`

**Beschreibung:** Inspecta-2 linetable is loaded with new lineaddressees. Thus the next image is written to a position defined by ImageNr. DoubleBuffer 1/0 activates or deactivates the hardware controlled exchange of memory blocks.

For cameras with asynchronous shutter a hardware controlled shutter time in units of linetimes can be selected. ShutterTime 0 is used for cameras with camera controlled shutter-time.

ImageNr defines the position of the image in image memory. The return value is the offset within the image meory for th selected image. For ImageNr == -1 the returnvalue is the index of the last image that fits completely into image memory.

Image memory structure:

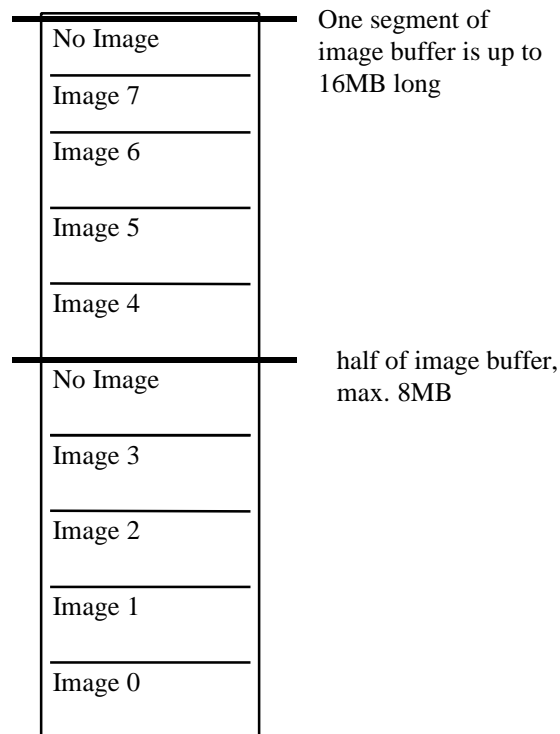
Inspecta-2 hardware divides image memory into 16MB segments. A segment can be less than 16Mbytes, the physical start address plus buffer length must not cross a 16Mbyte memory boundary. There are as many 16MB segments as fit into the amount of memory defined as image memory.

DoubleBuffer = 1 divides the image memory into two parts of same length.

### Distribution of images in image memory:

For one image there is (numlin \* linelen) image memory reserved. There are as many images reserved as fit into one half of image memory or into 8MB. Image memory beyond the last completely defined image and the 8/16MB boundary is not accessed by Inspecta-2 hardware.

If image memory is longer than 16MB the memory structure as shown in the illustration below is repeated, the next image with ImageNr = 8 starts at image memory offset 16MB



**Returns:** Offset in image memory for the selected ImageNr. Or index of the last image that fits completely into memory if ImageNr == -1.

**Example:** --

**Remark:** Processing time is ca. 0.5 usec per line, e.g.: an image with numlin = 508 the function consumes ca. 1msec. If DoubleBuffer = 1 consumed time also doubles. If shutter times >0 are selected, each increment (linetime) adds also 0.5 usec.  
If request parameters are the same as with the last call, no time is consumed. Use m(v)fg\_DefineNextImage with Rev.>= 1.77 and INSPECTA-2 hardware

### 2.3.4 Cameracontrol

The following functions control specific features of a camera. For example: 'async shutter' which means an exposure time which is shorter than the frame time, or 'integration' which means an exposure time which is a multiple of frame time.

```
mvfg_startstop          // enable or disable recording
mvfg_photo(), mfg_photo // open and close the 'shutter' of a camera
```

Camera control functions can be called at any time by the application, or being called through the interrupt service routine every VSYNC or every HSYNC. The application provides 'descriptors' which are used by the service routine to do the requested action.

---

#### 2.3.4.1 mvfg\_startstop ( DWORD flag ); mfg\_startstop ( DWORD flag)

**Synopsis:** void mvfg\_startstop ( DWORD flag );

**Description:** This function starts or stops grabbing immediately.

Possible values for 'flag':

MVFG\_STOP = 0 stop recording

MVFG\_START = 1 start recording

**Returns:** --

**Example:** --

### 2.3.4.2 mvfg\_stat (), mfg\_stat ()

**Synopsis:** DWORD mvfg\_stat ( void );

**Description:** this function returns the actual status of the MVFG.

**Returns:** mfg\_stat () returns four byte status code:

**Bits:**

0=MC0	camera special fct. control 0
1=MC1	camera special fct. control 1
2=MC2	camera special fct. control 2 / VINIT
3=MC3	camera special fct. control 3 / integration
4=CAM0	select one of two a/d input triples
5=RGBX0	data width
6=RGBX1	data width
7=RGBX2	data width
8=LDV	1=linedata valid
9=FDV	1=frame data valid (analog or digital)
10=PHO	1=Photo has been started and is not ready 0=Photo is ready (or not started)
11=ODD	1=Odd field
12=pixel clock/4	(>= rev. 4): internal pixel clock/4
13= undefined	(>= rev. 4)
14=FDV digital	1=digital camera frame data valid
15=undefined	
16	1= connects video red data buffer to red channel
17	1= connects video green data buffer to red channel
18	1= connects video blue data buffer to red channel
19	1= connects digital video data buffer to red channel
20	1= interrupt enabled
21	undefined
22	undefined
23 DIS_WR	1= disable writing to fifo
24=AM0	camera mode bit 0
25=AM1	camera mode bit 1
26=ISRC0	interrupt source 0
27=ISRC1	interrupt source 1
28=DIS_EOL	disable end of line processing if set
29=PAL	525/625 lines
30=STEST1	camera mode bit 2
31=STEST	camera mode bit 3

ISRC [0..2] =		; interrupt occurs on
0/0		; hor. freq/128
1/0		; disable write to fifo
0/1		; falling edge of VSYNC
1/1		; falling edge of HSYNC

---

### 2.3.4.3 mfg\_fdvhi ()

**Synopsis:** DWORD mfg\_fdvhi ( void );  
**Description:** this function waits for fdv ( VSYNC ) going high.  
**Returns:** 0 if o.K, -1 if timeout ( app. 0.5sec )  
**Example:** --  
**Remark:** DOSX only

---

### 2.3.4.4 mfg\_fdvlo ()

**Synopsis:** DWORD mfg\_fdvlo ( void );  
**Description:** this function waits for fdv ( VSYNC ) going low.  
**Returns:** 0 if o.K, -1 if timeout ( app. 0.5sec )  
**Example:** --  
**Remark:** DOSX only

---

### 2.3.4.5 mvfg\_photo (time), mfg\_photo (time)

**Synopsis:** LONG mvfg\_photo ( LONG shuttertime );

**Description:** Use this function for cameras with an ( electronic ) shutter. This function controls the shutter, but it does never flip the frame-memory. Depending on the type of connected camera. one or more additional VSYNCs are generated. mfg\_photo provides a vsync\_skip counter, so that the service routine „knows“ which VSYNC terminates the shuttered frame. mfg\_photo () will not return immediately, the variable shutter times are timed out within the mfg\_photo () function. mfg\_photo will disable system interrupts as long as the timing runs if an appropriate mask with mfg\_maskint () is set.

**Returns:** 0 if o.K.  
-2 if a Kodak Megaplug should be shuttered while in „continuous“ mode.

**Example:**

```

/* wait for an external trigger. As soon as the trigger is encountered, process
mfg_photo (), then call mvfg_input (). Use a TM9700 camera with shutter
mode 4. */

/* wait for trigger, use opto-coupled input, bit 0 */
while ( mvfg_ppin() & 0x01 );

mvfg_photo ( -4);    /* process shutter    */

return mvfg_input ( timeout );

```

The following table shows details of shutter-time and processing-time of mfg\_photo depending on various camera types and modes:

Kameratyp PULNIX	mfg_mo dcam ()	calling para- meter	shuttertime in usec	processing time in 'mfg_photo()'	Remark
TM620, TM720, TM9700, TM6700, TM1000	0x9C 0x9C 0x8C/9C 0xBC 0xAC	> 0	1-210*64 1-210*64 1-420*64 1-420*32 1-840*64	app. shuttertime	Sutterswitch position 9 (variable shuttertime).
TM620, TM720, TM9700, TM6700, TM1000	0x9C 0x9C 0x8C/9C 0xBC 0xAC	= 0	see camera description	app. 9 ms	Sutterswitch 1 - 8 ( internal fixed shuttertime ).
TM620, TM720, TM9700, TM6700, TM1000	0x9C 0x9C 0x8C/9C 0xBC 0xAC	- 1 bis -8	see camera description	app. 128 usec	Sutterswitch 1 - 8 (internal fixed shuttertime ).

'shuttertime' = -1 will be no-op if any other than the above mentioned camera mode is set.

With camera modes 0x16 or 0x17 ( Kodak Megaplast ) shutter time is timed by the camera, if the calling parameter was 0. If >0, timing is defined by a DOS call and thus not very precise. For these cameras, the bit PHO in mfg\_stat is set until the frame is ready.

---

### 2.3.4.6 mvfg\_ScanPeriod (ScanPeriod), mfg\_ScanPeriod (ScanPeriod)

**Synopsis:** void mvfgScanPeriod( DWORD ScanPeriod );

**Description:** Sets the frequency of the Start of Scan Signal for a linescan camera. Parameter ScanPeriod counts in pixel clocks. ScanPeriod is greater or equal  $N \times (\text{physical number of pixels per scanline})$ , and less than  $N \times (\text{physical number of pixels per scanline}) + 8192$ .  $N=1$  for black&white cameras and  $N=3$  for color cameras with serial data output.“ ( e.g.: T.V.I.)

Beginning with hardware version number 0x0e the maximum of ScanPeriod <  $N \times \text{linelength} + 65536$ . Hardware versionnumber is available with DOSX: global variable mfg\_rev, with Windows: datptr[MVFGD\_REV].

If the linescan camera is triggered by an external SOS (exposure) signal, (e.g.: from an external rotary encoder) use: mvfgScanPeriod( 128 ) to suppress any internally generated SOS signal. ( IMP Nr >= IMP418 ), Driver version >= 2.01

**Returns:** --

**Example:** --

mvfg\_ScanPeriod writes the global variable start\_scan.

---

### 2.3.4.7 mvfg\_contWrite ( DWORD NrOfFrames, DWORD Circular );

**Synopsis:** mvfg\_contWrite (DWORD NrOfFrames, DWORD Circular);

**Beschreibung:** Starts grabbing for sequences of frames up to the requested number of frames.

With interlaced cameras, every field is recorded seperatedly. There are double as many fields recorded as defined by DWORD NrOfFrames.

DWORD Circular = 0: one time recording, \_fgtv\_valid = 1 after stop of grab.

DWORD Circular = 1: recording restarts automatically after NrOfFrames expired. Stop recording with mvfg\_startstop ( 0 ).

**Returns:** global variable (frm\_cnt) or DatPointer[MVFGD\_FRM\_CNT] counts number of recorded frames. If frm\_cnt = 0 the sequence is ready.

**Example:** --

**Remark:** Use driver > 1.65 and INSPECTA-2 hardware Rev. >4, IMP Nr. >317. No multiplane camera modes. All lines except those while VSYNC active are recorded. Interlace even & odd fields are not combined to frames. Use mvfg\_blank (blanktime) to suppress black pixel at begin of line. Before using m(v)fg\_ContWrite (..) an initialization with mvfg\_contWriteInit ( DWORD PhysAddr) is necessary.

### 2.3.4.8 m(v)fg\_Snap ( DWORD mode, DWORD stop/exchange );

**Synopsis:** m(v)fg\_Snap ( DWORD mode, DWORD stop/exchange );

**Description:** m(v)fg\_Snap starts grabbing with hardware controlled shutter, or enables a external trigger signal on input 0 of the opto-coupled inputs to start exposure of a single frame. On end of frame grabbing can be stopped or memory blocks can be exchanged.

Parameter mode:

- = 0: *start grab on next vsync, stop or toggle and stop then, \_fgtv\_valid = 1 (not implemented.)*
- = 1: *start grab on next vsync, stop or toggle and stop then, \_fgtv\_valid = 1 (not implemented.)*
- = 2: *start toggling on next even field, continue until vsync after requested stop, \_fgtv\_valid = 1 (not implemented.)*
- = 3: *start random shutter, stop or toggle and stop when ready; \_fgtv\_valid = 1 when ready*
- = 4: *enable external shutter on opto-in bit 0 high pulse, stop or toggle and stop when ready, \_fgtv\_valid flag = 1 when ready*

Parameter stop/exchange:

- = 0: *stop if action done*
- = 1: *stop and exchange (except mode 2) if action done*

**Returns:** --

**Example:** --

**Remark:** Use driver >= 1.77 and INSPECTA-2 hardware rev. >4, IMP Nr. >=383

### 2.3.4.9 mvfg\_GrabberSelect ( DWORD number );

**Synopsis:** DWORD mvfg\_GrabberSelect ( DWORD number );

**Description:** mvfg\_GrabberSelect selects one of four Inspectas in an WinNT PC. All subsequent Inspecta calls are directed to the selected Inspecta.

Function mvfg\_datpnt () will return the pointer to the selected Inspecta driver data area.

Image memory start address is defined in WinNT Registry through „MemStartPage“ parameter in key: [HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\MpfgNtx] whereas MpfgNtx (x=0..3) is the driver for the corresponding Inspecta. (see also: 2.1.3)

**Parameter number:** = 0..3: one of four Inspectas

**Returns:** the number of the last selected Inspecta or -1 if selected Inspecta is not present.

**Example:** --

**Remark:** Use driver >= 2.12 and INSPECTA-2 hardware rev. >4, IMP Nr. >=445

### 2.3.4.10 mvfg\_Linescan ()

**Synopsis:** void mvfg\_Linescan (DWORD dwScanRate, DWORD dwExpTime, DWORD dwEnaEnc, DWORD dwDivider);

**Description:** This function combines all parameters that are necessary to define an operation mode of a linescan camera that is connected to an Inspecta-4D (RS-644 parallel) or Inspecta-4C (CameraLink).

**dwScanRate:** divides the pixelclock of a camera by this value and thus defines the horizontal frequency of a linescancamera, as long as dwEnaEnc == 0 . This value must be greater than the linelength of the camera and less than 65535 .

**dwExpTime:** divides the pixelclock of a camera by this value and thus defines the exposure time of the camera, if this camera supports exposure time control. This value must be greater than the linelength of the camera and less than 65535.

**dwEnaEnc:** Deactivates or activates the input of an external encoder.

= 0 Encoder is inactive

- = 1 a single phase encodersignal on bit 2 of the opto-coupled input divided by: *dwDivider* triggers a line.
- = 2 a bi-phase encodersignal on bit 1 and bit 2 of the opto-coupled input divided by: *dwDivider* triggers a line if the encoder runs in forward direction. In backward direction the pulses are counted without triggering a line. As many pulses in forward direction as previously counted in backward direction must be input before a new line is triggered. Forward/backward direction is defined by connecting the encoder outputs A/B to bit1/2 or bit2/1.
- = 3 a bi-phase encodersignal on bit 1 and bit 2 of the opto-coupled input divided by: *dwDivider* triggers a line if the encoder runs in forward direction. In backward direction no pulses are counted. Forward/backward direction is defined by connecting the encoder outputs A/B to bit1/2 or bit2/1.
- = 1..3 If no encoder signal is present for more than (pixelclock/65535) a new line is triggered but not captured. An incoming encoderpulse is delayed while this dummy line is running. This avoids an over-exposure of the first lines after longer periods of inactivity.

**dwDivider:** divides the encoder output by this value and triggers a new line. The divisor is:  $1+dwDivider$  for  $dwDivider = 0...255$ .

**Returns:** --

**Example:** --

**Reamrks:** For constant image brightness with changing encoder speeds the linescan camera has to have the ability of exposure control.

If the camera has a programmable exposure time the parameter: *dwExpTime* is set to shortest possible pulse width that the camera can accept. (In general: 80). Highest linefrequency is then:  $1/(\text{ExposureTime}+\text{Line Output time})$

If Inspecta-4DC controls exposure time (depending on camera manufacturer often called: „level controlled exposure time“), the parameter: *dwExpTime* is set to the desired length. Maximum linefrequency =  $1/(\text{Line Output time})$  is reached if exposure time is less Line Output time.

Driverversion  $\geq 2.75$  und Inspecta-4D/C hardware is necessary.

### **2.3.5 Display**

The powerful display-functions use VGA-boards with an S3 VGA controller. All resolutions and color depths need 1MB display RAM max.

There are functions which use the S3 graphic engine and others which provide direct access to video RAM.

The graphic engine supports overlays, zoom, hardware BitBlts, polylines with hardware Bresenham, clip and fill functions.

Display functions with direct access to video memory allow data transfer rates up to 40 MB / sec. Thus they provide real time display with more than 25 full-frames per second or color displays with 24 or 32 bit color. these functions display windows with selectable position and size.

### 2.3.5.1 init928 (gmode)

**Description:** The graphic controller will be initialized for the requested resolution and color. 1MB of video RAM is sufficient.

**Parameters:**

```

gmode == 0 for 640 x 480 x 8 resolution
      == 1 for 800 x 600 x 4 resolution
      == 2 for 800 x 600 x 8 resolution
      == 3 for 1024 x 768 x 4 resolution
      == 4 for 1024 x 768 x 8 resolution
      == 5 for 640 x 480 x 24 resolution
      == -1 VGA mode 3 ( 80 x 25 color characters )

```

**Returns:** --

**Example:**

```

main ();
{
init928 ( 0 );          /* Init to 640*480*8bit */

...                    // do frame grabbing & image processing here

/* end of programme */
vmfg_isr ( 0 )         // switch off interrupt & restore vector
init928 ( -1 );       // select vga mode 3
}

```

### 2.3.5.2 palette928 (palette)

The 'palette' parameter has the following meanings

```

palette == 0 -> linear 8 bit greyscale with lsb two bits overlay:
      overlay = 0 -> black
      overlay = 1 -> red
      overlay = 2 -> green
      overlay = 3 -> white
palette == 1 -> 3/3/2 red/green/blue 256 color palette
palette == 2 -> expanded 8 bit greyscale

```

If called with palette == 0, the VGA palette increments by one greyscale every four pixel-values. example:

pixel value	VGA output
0	black overlay
1	red overlay
2	green overlay
3	white overlay
4	1
16	4
255	63

the overlay colors are only accessed if the calling parameter RAM in the funtions ( vmfg\_putxwin, mfg\_c, mfg\_fill, mfg\_line, mfg\_pblt ) is set.

---

### 2.3.5.3 mfg\_clip (onoff)

**Description:** mfg\_clip enables or disables a defined clip-window. Be sure to disable mfg\_clip, if ms3\_setclipwindows has never been called.

**Example:** mfg\_clip(0); // disable a clip-window  
mfg\_clip(1); // enable a clip-window

---

### 2.3.5.4 ms3\_setclipwindow (x, y, w, h)

**Description:** ms3\_setclipwindow defines a clip-window which starts at x, y (0, 0 top left of screen) with the size w (width) and h (height).

**Example:** ms3\_setclipwindow (x, y, w, h);

---

### 2.3.5.5 mfg\_setcolor (foreground, background)

**Description:** mfg\_setcolor defines foreground and background color for mfg\_fill(), mfg\_c() und ms3\_line. The colors depend on the previous selected palette:

#### 2.3.5.5.1 Palette 0 (linear 8 bit greyscale with lsb two bits overlay):

When writing to overlay, the lowest two bits of the parameter foreground (and with mfg\_c() the parameter background) define following overlay colors:

foreground = 0: transparent  
foreground = 1: red  
foreground = 2: green  
foreground = 3: white

---

#### 2.3.5.5.2 Palette 1 ( 256 colour palette):

Following values result:  
foreground = 0: black  
foreground = 255: white. The colors are defined by a 3:3:2 / r:g:b byte.

---

#### 2.3.5.5.3 Palette 2 (expanded 8 bit greyscale):

foreground = 0: black  
foreground = 255: white Values between are shades of grey.

---

### 2.3.5.6 mfg\_fill (x, y, w, h, ram)

**Description:** mfg\_fill fills the requested window starting at x, y (0, 0 top left of screen) with the size w (width) and h (height) with that color that was defined by a previous call of mfg\_setcolor(foreground, background). The overlay is filled if RAM = 2, if RAM = 1 the image is filled.

**Parameters:** x,y: graphic position on screen (lower left zero)  
width, height: size of fill rectangle  
ram=1: to VRAM  
ram=2: to overlay

**Example:** mfg\_fill (x, y, w, h, ram)

---

### 2.3.5.7 mfg\_c ( x, y, width, height, bitmap, ram)

**Description:** mfg\_c copies a bitmap whose length is divisible by eight, to a random position on the image or in the overlay.

**Parameters:** x,y: graphic position on screen (lower left zero)  
width, height: char box size [pixel]  
bitmap: bitmap (xsize+7)/8 \* ysize bytes  
ram=1: to VRAM  
ram=2: to overlay

**Example:** mfg\_c (x, y, width, height, bitmap, ram)

---

### 2.3.5.8 ms3\_line (npoint, xyv, ram, plmode)

**Description:** ms3\_line connects pairs of points who are defined with the vector xyv ( pointer to number of npoint pairs of points ).

**Parameters:** xyv: vector of points, "npoint" points  
ram=1/2 for write line into VRAM/overlay  
plmode=0 -> connected polyline; 1->disjunct vectors

---

### 2.3.5.9 ms3\_pblt (xs, ys, smask, xd, yd, dmask, width, height, ram)

**Description:** Copies within video ram a rectangle with an arbitrary pixel aligned starting point using a source mask to a destination rectangle with a arbitrary pixel aligned starting point using a destination mask.

Use this function to move characters with arbitrary size or the mouse cursor over the screen.

**Parameters:** xs, ys: source (UL)  
smask: source bitplane mask  
xd, yd: dest (UL)  
dmask: dest bitplane mask  
width, height: size of rectangle  
ram=1/2: copy in VRAM / Overlay

### 2.3.5.10 vmfg\_put2win (ptr, x, y, width, takepix, height, takeline, pitch, ram)

**Description:** vmfg\_put2win writes image data from a starting point x, y ( 0, 0, upper left corner of screen ) to a window with the width w (width in pixel) and height h (height in lines).

The image can be zoomed down. Zooming is done by omitting pixel and/or lines and can be adjusted independent for x (takepix) or y (takeline) with a maximum of 6.

Image data starts from pointer ptr (long), the source linelength = pitch.  
„Width“ must be chosen so that it is divisibly without remainder by 2\*takepix.

**Parameters:**

ptr:	UL corner of CPU image memory window to transfer
x, y:	UL corner of window in graphics display memory
width:	width of window [pixel] in dest device
takepix:	take each "takepix" pixel per line transfer (0->take all)
height:	height of window [lines] in dest device
takeline:	take each "takeline" line in dest device (0->take all)
pitch:	ptr memory address offset between lines
example:	mapping a 2560 pixel window to 640 display makes: width=640; takepix = 4; pitch=2560
destination:	ram=1:VRAM; 2:OVLY

### 2.3.5.11 vmfg\_put4win (ptr, x, y, width, multx, height, multy, pitch, ram)

**Description:** vmfg\_put2win writes image data from a starting point x, y ( 0, 0, upper left corner of screen ) to a window with the width w (width in pixel) and height h (height in lines).

The image can be zoomed up. Zooming is done by replicating pixel and/or lines and can be adjusted independent for x (multx) or y (multy) with a maximum of 10.

Image data starts from pointer ptr (long), the source linelength = pitch.  
„Width“ must be chosen so that it is divisibly without remainder by 2\*multx.

**Parameters:**

ptr:	UL corner of CPU image memory window to transfer
x, y:	UL corner of window in graphics display memory
width:	width of window [pixel] in dest device
multx:	multiply each pixel "multx" times in X
height:	height of window [lines] in dest device
multy:	multiply each line "multy" times in Y

pitch: ptr memory address offset between lines  
 example: mapping a 512x512 pixel window 1024x1024 display  
 makes:  
 width=height=1024; multx=multy=2; pitch=512  
 destination: ram=1:VRAM; 2:OVLY

---

### 2.3.5.12 linadr3 ()

**Description:** linadr3 enables the direct access to video RAM of the S3 VGA controller.

linadr3 () maps the physical video RAM address to the actual data-segment and provides a linear address to have access to it.

linadr3() must be called before using mvfg\_dmawin.

---

### 2.3.5.13 mvfg\_dmawin (x, y, width, height, pointer, pitch, color)

**Description:** mvfg\_dmawin write video data direct to the RAM of the S3-VGA controller. It is the fastest way of writing images to screen.

The direct access of the VGA RAM must be opened by a previously called linadr3 (). Overlays are destroyed.

**Parameters:**

x, y:	UL corner of window in graphics display memory
width:	width of window [pixel] in dest device
height:	height of window [lines] in dest device
ptr:	UL corner of CPU image memory window to transfer
pitch:	ptr memory address offset between lines
color=0	-> 8-Bit grey scale
color=1	-> 3/3/2 r/g/b color mode, ptr to other color planes is ptr+[pel_frm]
color=2	-> r/g/b 24 bit color mode ptr to other color planes is ptr+[pel_frm]
color=3	-> r=g=b grey mode

---

### 2.3.5.14 `linends3 ()`

**Description:** `linends3 ()` closes direct access to S3 RAM.

---

## 2.3.6 Miscellaneous functions

---

### 2.3.6.1 `mvfg_IntCallback ()`, `mfg_IntCallback`

**Synopsis:** `void mvfg_IntCallback(* user_proc, segments );`

**Description:** `mvfg_IntCallback ()` provides a pointer to an application-programme to the MVFG interrupt Service routine. The MVFG service routine calls then this application-programme on every interrupt. ( every 64usec for interrupt on HSYNC and every 16 msec on inteerupts on VSYNC ).  
Use this function to avoid any polling loops in the application programme (e.g. waiting for an optical sensor ).  
The parameter 'segments' is always 0.

**Returns:** --

**Remark:** This function is available with DOSX and Windows95. Do not call any systems functions out of this user programme, because it runs as an extension of the protected-mode service routine.

### 2.3.6.2 mvfg\_IntSource (), mfg\_IntSource ()

**Synopsis:** void mvfg\_IntSource ( DWORD irqsource );

**Description:** 'mvfg\_IntSource()' defines the source of an interrupt. Use this function in conjunction with mvfg\_IntCallback () to define, how often the user supplied programme should be called.

The parameter irqsource is the same as in function mvfg\_isr (). Bits 10-12 define the source of an interrupt.

'irqsource' can have the following values:

Bits: 0..7 = 0

Bits: 8..9 = 0

10 = ISRC0 interrupt source 0

11 = ISRC1 interrupt source 1

12 = ISRC2 interrupt source 1

Bits: 13..32 = 0

irqsource = 0x000 ; hor. freq/128  
 0x400 ; falling edge of VSYNC analog  
 0x800 ; start/stop flipflop is set  
 0xC00 ; falling edge of HSYNC analog  
 0x1400 ; falling edge of VSYNC digital  
 0x1C00 ; falling edge of HSYNC digital

**Returns:** --

**Example:** Call the application programme with every VSYNC.

```
// define (VSYNC) as interrupt source.
```

```
mvfg_IntSource ( 0x0400 );
```

```
// initialize MVFG-Callback function.
```

```
mvfg_IntCallback( (DWORD) myCallBackFkt, 0 );
```

### 2.3.6.3 mfg\_sync ()

mfg\_sync () is the processing part of the MVFG interrupt service routine. The interrupt service routine just calls mfg\_sync, reenables the interrupt and returns from interrupt.

mfg\_sync does everything that has to be done when a VSYNC is encountered:

- looking for the odd field of an interlaced video.
- check for new camera & mode selection and the A/D converter levels within a sequence.
- flip frame-memory if the flag \_fgtv\_valid == 0
- call the application programme if mvfg\_IntCallback is activated.

mfg\_sync must be called every HSYNC ( if this is the source of the interrupt ) or every VSYNC. If an application programme has its own service routine for the MVFG interrupt, mfg\_sync has to be called first.

mfg\_sync checks & changes the following counters & pointers:

dword int_cnt	increments with every interrupt.
dword frm_cnt	increments mod frame_nr with every full-frame if rotating recording is selected ( req_frm = -1 ), shows otherwise the actual image position in frame-memory.
dword odd_flag	=1 if the just recorded field was an odd field.
dword frm_beg	offset within frame-memory points to the first pixel of the just recorded frame.
dword _fgtv_valid	=1 if frame-memory was just flipped.
dword integ_count	counter for integration time mod full-frames.
dword integration	requested number of full-frames to integrate.
dword stop_start	=1 if recording is not stopped.

#### 2.3.6.4 mvfg\_chkclk (), mfg\_chkclk (),

**Synopsis:** DWORD mvfg\_chkclk ( void );

**Description:** Checks if a pixel clock is available for the selected camera mode and input channel.

**Returns:** 0: Pixel clock found  
-2: No pixel clock found

**Remark:** Use this function after selecting a camera mode with external clock input, to check if a camera is connected and working.

**Example:** --

---

#### 2.3.6.5 mvfg\_ppin (), mfg\_ppin (),

**Synopsis:** DWORD mvfg\_ppin ( void );

**Description:** Read the opto-coupled digital input.

**Returns:** bits 0 .. 3 state of the four input bits.  
bits 1 .. 7 state of the four output bits.

**Example:** --

---

#### 2.3.6.6 mvfg\_ppout (dout), mfg\_ppout (dout)

**Synopsis:** void mvfg\_ppout( DWORD value );

**Description:** Write the opto-coupled digital output port.

**Parameters:** bits 0 .. 3 are valid.

**Returns:** --

**Example:** --

---

---

### 2.3.6.7 mfg\_bmp (parameters)

**Description:** mfg\_bmp write images or a window out of an image to disc. File format is Windows compatible uncompressed \*.BMP format.

**Parameters:**

- \*file\_name: long pointer to ASCIIZ string with filename
- \* ptr: long pointer to frame buffer
- x: x position of window, 0 = left
- y: y position of window, 0 = first line
- width: width of window in pixel
- height: height of window in lines
- pitch: source linelength
- lut: Look Up Table: 0 = 256 shades of grey  
2 = 24 bit RGB color, color planes are expected at an offset of ploff.
- \*buffer: long pointer to copy buffer
- ploff: plane offset for color planes.

**Rückgabewert:** 0 = o.K.  
x = File Error (create, write, disk full etc.)

---

### 2.3.6.8 m(v)fg\_SetVideoClock (frequency)

**Synopsis:** void mvfg\_SetVideoClock( DWORD frequency );

**Description:** Sets video clock synthesizer to requested value (Hz)

**Returns:** --

**Example:** --

**Remark:** Valid for software revision >= 1.62

---

### 2.3.6.9 mvfg\_get\_info( Z\_MVFG\_INFO \* mvfg\_info )

**Synopsis:**                DWORD mvfg\_get\_info( Z\_MVFG\_INFO \* mvfg\_info )

**Description:**        This function returns within the structure ,mvfg\_info‘ hard- und software-informations of the installed Inspecta.  
If the argument is ZERO, only the serial number of the installed Inspecta is returned.

The struture Z\_MVFG\_INFO contains the following informatios:

Element	Description
driver_version_ms	Most significant bits of fileversion of loaded device-driver.
driver_version_ls	Least significant bits of fileversion of loaded device-driver.
dll_version_ms	Upper 32 bits of fileversion of ,MVFGD32.DLL‘
dll_version_ls	Lower 32 bits of fileversion of Fileversion von ,MVFGD32.DLL‘
hw_type	Type of installed Inspecta: 0 = undefiniert 1 = Inspecta-1 2 = Inspecta-2 3 = Inspecta-3 4 = Inspecta-4
hw_vendorID	Mikrotrons Vendor ID
hw_deviceID	Inspectas Device ID
hw_revID	Inspectas Revision ID
hw_snr	Inspecta-4 serialnumber
hw_imp	Inspectas firmware versioncode

**Returns:**                != 0                        Inspecta-4 serial number  
                              == 0                        Error

**Example:**                Z\_MVFG\_INFO        mvfg\_info;  
                              .  
                              mvfg\_get\_info( &mvfg\_info );

**Remark:**                this function is only availabe in WinNT/2K beginning with Rev 2.38

### 2.3.7 Testfunctions

The two display functions mvga (parameter) and m13vga (parameter) are too slow and have not enough resolution. They are of use for test purposes, because they work with every VGA

---

#### 2.3.7.1 palette ()

**Description:** palette () initializes the VGA adapter to mode 0x12 (640x480x16) and adjusts the VGA palette to a linear greyscale with 16 shades of grey. Call it before using the display function mvga (...).

---

#### 2.3.7.2 mvga (alt\_frame\_start, frame\_start, vga\_start, granularity)

**Description:** mvga (..) displays the frame-memory to the VGA screen with graphic mode 0x12. In addition, mvga (..) has a delta function, in that the pixel values of two consecutive images can be subtracted, and the difference is displayed as a colored pixel. The difference color is:

delta = 0 : black  
1 : blue  
2 : green  
3 : cyan  
4 : red etc.

**Parameters:**

alt_frame_start:	pointer to first pixel of image to compare.
frame_start:	pointer to first pixel of image to display.
vga_start:	pointer to first pixel on VGA screen.
granularity:	zoom down, omit every second, third etc. pixel/line.

---

### 2.3.7.3 palette13

**Description:** palette13 () initializes the VGA adapter to mode 0x13 (320x240x16) and adjusts the VGA palette to a linear greyscale with 256 shades of grey. Call it before using the display function m13vga (...).

---

### 2.3.7.4 m13vga ( ptr, vga\_ptr, granularity )

**Description:** m13vga writes to standard VGA with mode 0x13. To initialize, call palette928 (0) and palette13 () before.

**Parameters:**

- ptr: pointer to first pixel of image to display.
- vga\_ptr: pointer to first pixel on VGA screen.
- granularity: zoom down, omit every second, third etc. pixel/line.