Imagenation
PXC200 Precision Color Frame Grabber
## Contents

1. **Introduction** ......................................................... 1  
   Precision Capture Hardware ........................................... 2  
   Video Inputs and Formats ............................................. 3  
   Video Capture Modes and Resolution .............................. 3  
   Image Capture Modes .................................................. 3  
   Capture Resolution .................................................... 4  
   Real-Time Image Data Transfer ...................................... 5  
   PCI Bus Master Design ................................................ 5  
   Selectable Destination for Image Captures ...................... 5  
   Trigger Input ............................................................. 6  
   Programming Libraries and DLLs .................................... 6  
   The PXCVU Program ..................................................... 7  
   Utility Programs ......................................................... 7  
   PXCREV .................................................................. 7  
   VGACOPY .................................................................. 7  
   PXCLEAR .................................................................. 8  
   Next Steps... ............................................................. 8
2. Installing Your Frame Grabber ................................. 9
   Do You Need a Cable? ......................................................... 9
   Standard PCI-Bus Cables .................................................. 9
   PC/104-Plus Cables ......................................................... 9
   Installing Your Board ...................................................... 10
   Installing the Software .................................................... 12
   DOS, DOS/4GW, and Windows 3.1 Software Installation ...... 12
   Windows 95 Software Installation .................................. 15
   PXC200 Software Directories ........................................ 17
   Troubleshooting .............................................................. 17
   Error Loading DLL ......................................................... 18
   Error Loading VxD .......................................................... 18
   Problems Running PXCVU or PXCREV ............................ 18
   Slow Video Display Performance ................................... 19
   Windows Hangs or Crashes on Boot .............................. 19
   Technical Support ......................................................... 20

3. The PXCVU Application ............................................. 21
   Setting Up PXCVU ......................................................... 21
   Starting PXCVU ............................................................ 22
   Running PXCVU with More Than One Frame Grabber ....... 22
   Using PXCVU ............................................................... 23

4. Programming the PXC200 ............................... 25
   Library Organization ..................................................... 25
   Operating System Specifics ........................................... 26
   DOS Programming ....................................................... 27
   Windows 3.1 Programming .......................................... 28
   Windows 95 Programming ............................................ 28
   Programming Language Specifics .................................. 30
   Programming in C ........................................................ 30
   Visual Basic Programming ........................................... 31
   Typical Program Flow .................................................. 33
Frame Grabbing and PCI Bus Performance ............................... 63
Accessing Captured Image Data................................................. 64
Frame and File Input/Output....................................................... 65
  BMP Files ................................................................................ 65
  Binary Files ............................................................................. 66
Using the Video Display DLL....................................................... 66

5. PXC200 Library Reference ......................................................... 69

6. Frame Library Reference .......................................................... 101

7. The VGA Video Display Library .............................................. 119
   Initializing and Exiting the Library ........................................... 120
   Entering and Exiting VGA Graphics Mode............................... 120
   Displaying VGA Text and Graphics......................................... 121
     VGA Memory Addressing....................................................... 122
   Menu Creation, Configuration, and Display............................. 122
   Menu Structures and Types .................................................... 123
   Function Reference .................................................................. 125

A. Cables and Connectors ............................................................ 137
   Standard PCI Bus Cables......................................................... 137
     26-pin D Connector............................................................... 137
   Connecting the +12V Output................................................... 138
   PC/104-Plus Cables ................................................................ 138

B. Hardware Specifications ......................................................... 139

C. Block Diagram ........................................................................ 141

Index ......................................................................................... 143
Introduction

The Imagenation PXC200 frame grabber features precision video capture hardware for applications that require high color accuracy. Features of the precision hardware design include:

- High color accuracy with low pixel jitter
- PCI bus master design for real-time image capture to system memory or directly to the VGA display
- Image capture resolution up to full-size: 640 x 480 (NTSC) and 768 x 576 (PAL and SECAM)
- Horizontal and vertical cropping and scaling of captured images to minimize system memory and bus bandwidth requirements
- Common color output formats, including YCrCb, RGB, and Y8 (grayscale)
- Continuous, software-initiated, and triggered image captures
- Four multiplexed composite video inputs (one input can be S-Video) with automatic video format detection of NTSC and PAL/SECAM formats
- Digital TTL-level trigger input
- +12V output for powering cameras or other devices
The PXC200 is available in two hardware configurations:

- PCI bus, short card—for typical desktop PC systems
- PC/104-Plus bus—for embedded-systems applications based on the PC/104-Plus format

To make it easy to tap these hardware features, the PXC200 includes an elegant software interface that supports developing applications for 16-bit DOS, Watcom 32-bit DOS/4GW, Windows 3.1, and Windows 95:

- C libraries for building DOS applications
- DLLs for building Windows applications
- DOS VGA Video Display library for building a menu-based user interface
- Sample DOS and Windows source code
- PXCVU—a DOS image capture application

This chapter will give you an introduction to these features. More detailed technical information on features is included in Chapter 4, Programming the PXC200, on page 25.

**Precision Capture Hardware**

The design of the PXC200 video capture hardware produces high color accuracy and low pixel jitter:

- **Grayscale noise**—1.0 LSB RMS maximum
- **Pixel jitter**—±4 ns maximum

This accuracy makes PXC200 frame grabbers ideal for demanding scientific and industrial applications.
Video Inputs and Formats

The PXC200 frame grabber handles multiple camera inputs and video formats:

**Connect up to Four Cameras.** Switch between camera inputs in software. On the PXC200 standard PCI-bus configuration, BNC and S-Video connectors are provided for video inputs 0 and 1, respectively, and all four inputs are available through the 26-pin D connector. All four inputs can accept composite video signals, and video input 1 can be used for S-Video.

A PXC200 frame grabber automatically synchronizes to the selected video source.

**Use NTSC, PAL, or SECAM Video Formats.** PXC200 frame grabbers support the 60 Hz North American NTSC color and RS-170 monochrome formats, and 50 Hz European PAL and SECAM color and monochrome formats.

Video Capture Modes and Resolution

When you capture images with a PXC200 frame grabber, you can specify how you want to start the capture process, and whether you want to work with all or with just a subset of the total image data.

**Image Capture Modes**

There are three ways to capture images with a PXC200 frame grabber:

**Software-initiated grab.** On a command from an application program, the board grabs a single frame or field.
Imagenation

Triggered grab. The board waits for an external trigger and then grabs the frame.

Continuous acquire. In this mode, the board grabs one image after another. Continuous acquire is useful for applications that need to watch for changes between successive images, and for sending video data directly to other PCI devices.

With any of these modes, you can start the capture at the next field in the incoming video signal, or you can specify that the capture will start with field 0 or field 1.

Capture Resolution

PXC200 frame grabbers use a crystal-controlled pixel clock to sample horizontal lines of video at 14.32 MHz for NTSC or 17.73 MHz for PAL/SECAM. At these frequencies the frame grabber acquires more pixels per line than are required for the standard video formats and then uses interpolation to reduce the number of pixels to the specified value. On a typical display monitor with a 4 x 3 aspect ratio, a 640-pixel horizontal resolution results in approximately square pixels for images in NTSC video mode; a 768-pixel horizontal resolution results in square pixels for images in PAL and SECAM video modes; and a 720-pixel horizontal resolution supports the rectangular video pixels of conventional video displays.

If you don’t need to work with all of the image data, you can further scale the image horizontally and vertically. You can also crop the image horizontally and vertically, retaining just a rectangular subset of the image. By transferring only a subset of the image, you save memory and bandwidth on the bus, leaving more of both resources available to other parts of your application and to other applications.

Common color formats are supported for output, including YCrCb, RGB, and Y8 (8-bit grayscale).
Chapter 1  Introduction

Real-Time Image Data Transfer

The PCI bus master design of the PXC200 frame grabber lets you achieve real-time performance for captures to main memory or directly to the display.

PCI Bus Master Design

The bus master design of the PXC200 frame grabber lets the frame grabber directly control the transfer of image data to main memory or to another PCI device, such as a display controller. While the frame grabber is transferring data, the main CPU is free to run other parts of your application or other applications.

Data transfers can take advantage of the maximum 132 MB per second burst transfer rate of the PCI bus. Although actual throughput is typically well below the maximum burst rate, a properly-designed system can support real-time transfer and display of full-size, 8-bit-per-pixel video image data. At 16 or 24 bits per pixel, you might not be able to achieve real-time display of full-size images, depending on the design of the system.

Selectable Destination for Image Captures

You can choose the destination for the image capture data:

A buffer in main memory. The data is transferred via direct memory access (DMA) to a buffer in the computer’s main memory. The transfer is fast, and the data is available in memory for further processing.

Another memory-mapped device. The data is transferred via DMA directly to another PCI device. For example, some PCI VGA cards support such transfers, which can be used to display live video.
Imagenation

Trigger Input

PXC200 frame grabbers have an external TTL-level trigger input that can be used to trigger an image capture. A simple push button switch attached to this input can be used like a camera shutter button. The trigger input can be programmed to respond to either low or high logic levels, or to rising or falling edges.

Programming Libraries and DLLs

For custom applications, the PXC200 software includes support for writing your own frame grabber programs. The library and DLL functions take care of the details of low-level hardware control for you, letting you concentrate on getting your application working.

C Libraries for DOS—Write 16-bit DOS programs using the 16-bit library with Borland, Microsoft, or Watcom C compilers, or write 32-bit DOS programs using the Watcom DOS/4GW library.

DLLs for Windows—Write 16-bit or 32-bit Windows programs for Windows 3.1 and Windows 95 with C compilers from Borland and Microsoft, or with Visual Basic. The PXC200 DLLs are standard Windows DLLs, and you should be able to use them with most Windows development tools that can make calls to Windows DLLs.

DOS VGA Video Display Library—Use the Video Display library to create a menu-based user interface for your 16-bit DOS and 32-bit DOS/4GW applications that allows you to simultaneously display graphics and text.

Sample source code—Sample source code is provided, for both DOS and Windows, to show you how to use various features of the libraries and DLLs.
Chapter 4, Programming the PXC200, on page 25, describes the main features of the PXC200 hardware and software and how to use them to build applications. For reference information on all PXC200 library functions, see Chapter 5, PXC200 Library Reference, on page 69, and Chapter 6, Frame Library Reference, on page 101. The DOS VGA Video Display library and its functions are described in Chapter 7, The VGA Video Display Library, on page 119.

The PXC200 software includes a DOS frame grabber application called PXCVU. Using PXCVU, you can capture images, save images to disk, and adjust many of the image capture features of a PXC200 frame grabber—all without writing a single line of code. For more information, see Chapter 3, The PXCVU Application, on page 21.

Utility Programs

The PXCVU Program

Utility Programs

The PXC200 software also includes several utility programs.

PXCREV

If you need to contact Imagenation Technical Support, you’ll be asked for your board’s revision number. PXCREV is a DOS program that displays the revision number for any frame grabbers it finds in your system. You must run this program from DOS, not from a DOS window in Windows.

VGACOPY

VGACOPY is a test program that lets you evaluate the performance of your computer for grabbing images and copying the data to the VGA dis-
play in DOS. For similar tests in Windows, see the Windows sample programs PXCDRAW1 and PXCDRAW2.

PXCLEAR

The PXCLEAR utility for Windows 3.1 and Windows 95 frees frame grabbers when a program terminates unexpectedly and does not call the required exit procedures. PXCLEAR tells you which frame grabbers are currently in use, and gives you the option of freeing all of them. It cannot be used to free individual frame grabbers; it frees all frame grabbers in the system or none of them. For more information, see The PXCLEAR Utility, on page 38.

Next Steps...

<table>
<thead>
<tr>
<th>For...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installing your PXC200 frame grabber</td>
<td>Chapter 2, Installing Your Frame Grabber, on page 9</td>
</tr>
<tr>
<td>Operating your PXC200 with the PXCVU program</td>
<td>Chapter 3, The PXCVU Application, on page 21</td>
</tr>
<tr>
<td>Writing your own frame grabber applications</td>
<td>Chapter 4, Programming the PXC200, on page 25</td>
</tr>
<tr>
<td>Connector and cabling specifications</td>
<td>Appendix A, Cables and Connectors, on page 137</td>
</tr>
<tr>
<td>Specifications for the PXC200 board</td>
<td>Appendix B, Hardware Specifications, on page 139</td>
</tr>
<tr>
<td>A PXC200 board block diagram</td>
<td>Appendix C, Block Diagram, on page 141</td>
</tr>
</tbody>
</table>
Installing Your Frame Grabber

Do You Need a Cable?

Standard PCI-Bus Cables

The BNC composite video connector and the S-Video connector on the standard PCI-bus configurations of the PXC200 board let you attach up to two video sources. Additional video sources (you can connect a total of four), a trigger input, and a +12V power source are also available by using the 26-pin D connector. To use the 26-pin connector, you’ll need a cable with the correct mating connector and pinouts. For information on making cables, see Appendix A, *Cables and Connectors*, on page 137.

PC/104-Plus Cables

You’ll need a cable to attach to the connector on frame grabbers with the PC/104-Plus configuration. For information on making cables, refer to the release notes that came with the frame grabber.
Installing Your Board

Follow the instructions below to install your board:

1  Turn off and unplug your computer, then remove its cover.

   **Caution**
   *Static electricity can damage the electronic components on the PXC200 board. Before you remove the board from its antistatic pouch, ground yourself by touching the computer’s metal back panel.*

2  Install the PXC200 board as follows:

   **For a standard PCI-bus board:**

   a  Locate an unused PCI expansion slot that is enabled for bus mastering. On some systems, you must enable a PCI slot for bus mastering by using a switch or jumper on the system board, or by changing the BIOS settings. Refer to the manual that came with your computer for more information.

   b  Remove the cover plate. Save the screw.

   c  Insert the PXC200 board into the slot and seat it firmly.

   d  Secure the board’s cover plate using the screw you saved.

   **For a PC/104-Plus board:**

   a  Set the four-position rotary switch on the PXC200 board to an unused number. Each PC/104-Plus plug-in module must be set to a unique number.

   b  Insert the PXC200 board into the connector and seat it firmly.
3 Following the instructions below, connect your board to the video input and, optionally, to other I/O:

For a standard PCI-bus board:

**BNC and S-Video connectors.** Connect your video source to the S-Video connector or to the composite video BNC connector (see diagram at left). The composite connector is video input 0, and the S-Video connector is video input 1.

**26-pin D connector.** If you’re using the 26-pin D connector, connect your cable to that connector. If you need to purchase or make a cable, see Appendix A, *Cables and Connectors*, on page 137.

For a PC/104-Plus board:

Attach your cable to the connector on the PXC200 board. For information on making cables, see Appendix A, *Cables and Connectors*, on page 137.

4 Replace the cover on the computer, plug it in, and turn on the power.

5 **This step applies to Windows 95 only.** When you restart your system, you might see the message “Found new multimedia PCI device,” and the *Add New Hardware Wizard* is displayed. If this happens, follow the steps below:

a Insert the **Windows 95** PXC200 software installation disk in the drive.

b In the wizard, click the Have Disk button.

c In the *Install from Disk* dialog, specify the drive letter for the floppy disk drive and click OK.

You should see a single option, *PX Precision Frame Grabber*, listed in the wizard.
d  Select *PX Precision Frame Grabber* and click Next.

e  Click Next again to let Plug and Play complete the installation.

You should see a message that Windows hasn’t finished installing the necessary software. You’ll install the software in the next section.

f  Click Finish.

6  That completes the hardware installation. Next, you’ll install the PXC200 software.

### Installing the Software

PXC200 frame grabbers can be used with DOS, DOS/4GW, Windows 3.1, and Windows 95. Refer to the appropriate section below for the operating system you are running.

#### DOS, DOS/4GW, and Windows 3.1 Software Installation

1  **This step applies only to DOS;** if you’re not using DOS, skip to the next step. The frame grabber needs a vacant 4 KB block of system memory in segment 0xD000 or in segment 0xE000. The 4 KB block of memory must be aligned on a 4 KB boundary; that is, it must be of the form 0xD?00-0xD?FF or 0xE?00-0xE?FF, where ? is the same hexadecimal digit in both the beginning and ending numbers of the range. For example, 0xD200-0xD2FF or 0xEA00-0xEAFF.
To make a memory block available for the frame grabber:

a Make sure the block is not used by any other hardware devices. You can use the Microsoft diagnostics program MSD to display memory usage. (MSD comes with DOS and Windows.)

b Modify the entry in CONFIG.SYS for your memory manager to prevent it from using the block. For example, if you are using EMM386, and you want to use 0xE000-0xE0FF for the frame grabber, add \xe000-e0ff to the end of the EMM386.EXE entry in your CONFIG.SYS:

\device=c:\dos\emm386.exe noems \xe000-e0ff

If you’re using another memory manager, like QEMM or 386MAX, consult your manual.

2 Insert the DOS/Windows 3.1 installation diskette in the floppy drive.

3 The diskette includes two installation programs, one for DOS and another for Windows. The DOS INSTALL.EXE program installs only the DOS and DOS/4GW software, not the Windows software; the Windows SETUP.EXE program installs all three. Decide which installation program you want to use, and follow the appropriate instructions below:

**DOS and DOS/4GW only**

a At the DOS prompt, type (substitute the appropriate drive letter for “a”) \a:\install and press Enter.

b When the INSTALL program has completed, reboot your computer.

c After rebooting your system, you can use the PXCVU program to verify that your frame grabber is correctly installed. For instructions on running PXCVU, see Chapter 3, *The PXCVU Application*,
on page 21. If an error message appears when you try to start PXCVU, see *Troubleshooting*, on page 17.

**Windows, DOS, and DOS/4GW**

**a** From the Program Manager in Windows, choose the File menu and select Run.

**b** In the Command Line box, type `a:\setup`, and click OK.

**c** When the SETUP program has completed, restart Windows.

Setup creates a new program group called `PX`.

**d** After restarting Windows, you can run one of the PXCDRAW sample programs to verify that your frame grabber is correctly installed. The sample programs are in the `c:\pxc2\win31` directory. If you have problems running the sample programs, see *Troubleshooting*, on page 17.

**Changes to System Files for DOS, DOS/4GW, and Windows 3.1**

The installation programs will, at your option, modify your AUTOEXEC.BAT and SYSTEM.INI (SETUP only) files. The changes are listed below so that you can make your own modifications, if you prefer. The installation programs do not look for their own modifications; if you run the installation programs more than once, don’t let them modify your system files unless you have removed the previous modifications.

**AUTOEXEC.BAT Changes for DOS, DOS/4GW, and Windows 3.1**

```
REM Imagenation’s Modifications
set path=c:\pxc2\dos;c:\pxc2\win31;%path%
set imagenation=c:\pxc2
REM Imagenation’s Modifications End
```
Adding c:\pxc2\dos and c:\pxc2\win31 to your PATH makes the samples and utilities easier to execute. If you do a DOS-only installation, c:\pxc2\win31 is not added. The IMAGENATION environment variable specifies the location of files required by the PXCVU application. PXCVU won’t run unless this variable is correctly defined.

After your AUTOEXEC.BAT file is modified, you must reboot your computer for the changes to take effect.

**SYSTEM.INI Changes for Windows 3.1**

```ini
[386Enh]
device=c:\pxc2\win31\pxc2.vxd
; Imagenation’s Modifications
device=c:\pxc2\win31\pxc2.vxd
; Imagenation’s Modifications End
```

The PXC200 Windows Virtual Device Driver (VxD), PXC2.VXD, is added to the [386Enh] section. The VxD will be loaded only when you start Windows. The PXC200 DLL, PXC2.DLL, requires this VxD; the DLL will not run unless the VxD is installed. After running Setup, you must restart Windows to load the VxD.

**Windows 95 Software Installation**

1. If you previously installed the Windows 3.1 PXC200 driver, you must edit the [386Enh] section of the SYSTEM.INI file to remove the lines that load the 16-bit VxD, PXC2.VXD. For more information, see **SYSTEM.INI Changes for Windows 3.1**, on page 15.

2. Put the Windows 95 installation disk in the floppy drive.

3. Click the Start button and click Run.

4. For the name of the program, type `a:\setup` and click OK.
5 Follow the instructions in the Install wizard to complete the installation.

Setup creates a new program group called PX.

When you have competed installing the software, you must reboot Windows 95 before the drivers that you have installed will be accessible.

6 Click the Start button and click Shut Down.

7 In the Shut Down Windows dialog, click Restart the computer and click Yes to restart Windows 95.

After restarting Windows, you can run one of the PXCDRAW sample programs to verify that your frame grabber is correctly installed. The sample programs are in the c:\pxc2\win31 directory. If you have problems running the sample programs, see Troubleshooting, on page 17.

Windows 95 Registry Changes

If you need to uninstall the PXC200 driver, you must edit the Windows 95 Registry by using the REGEDIT.EXE program in your Windows 95 directory.

The installation program adds the following key to the Windows Registry:

HKEY_LOCAL_MACHINE\System\Services\VxD\PXC2

The value assigned to this key is:

StaticVxD. A string key that contains the complete path of the VxD file, such as c:\pxc2\win95\pxc2.vxd.
PXC200 Software Directories

The installation programs create the LIB and INCLUDE directories, and directories for the appropriate operating systems:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\pxc2\lib</td>
<td>DOS and Windows libraries.</td>
</tr>
<tr>
<td>\pxc2\bin</td>
<td>Executable sample programs, DLLs, and drivers.</td>
</tr>
<tr>
<td>\pxc2\include</td>
<td>Header files.</td>
</tr>
<tr>
<td>\pxc2\dos</td>
<td>DOS and Watcom DOS/4GW sample source code.</td>
</tr>
<tr>
<td>\pxc2\win31</td>
<td>Windows 3.1 sample source code.</td>
</tr>
<tr>
<td>\pxc2\win95</td>
<td>Windows 95 sample source code.</td>
</tr>
</tbody>
</table>

These directories are structured to make program execution, compiling, and linking convenient.

You can run the Windows sample programs to control the frame grabber, write BMP files, and run the timing tests (don’t forget to first restart Windows to load the VxD). The sample programs are PXCDRAW1 and PXCDRAW2.

Troubleshooting

This section contains troubleshooting information for the following:

- Error loading DLLs
- Error loading VxDs
- Running PXCVU or PXCREV
- Slow video display performance
- Windows hangs or crashes on reboot
Error Loading DLL

The system can’t locate the PXC200 DLL. Either edit your PATH environment variable to include the path to the PXC200 DLL (see PXC200 Software Directories, on page 17) or move the DLL to the \WINDOWS\SYSTEM directory.

Error Loading VxD

When booting Windows 3.1, you might see the error “PXC2.VXD Requires a PCI compatible BIOS.” This means your BIOS lacks the BIOS32 Service Directory feature of the PCI BIOS Specification, Revision 2.0.

First, make sure you are using the version of the PXC2.VXD that came with your PXC200. If you’re using an older version, upgrade to the latest version. If you still get this error message with the latest version of PXC2.VXD, you’ll need to upgrade your BIOS; contact the manufacturer of your system for an upgrade.

Problems Running PXCVU or PXCREV

PXCVU and PXREV are DOS programs. You can’t run these programs in a DOS window in Windows. If your system hangs when you run PXCVU or PXREV, this is the most likely cause.

If the program hangs when you start it, you might have an IRQ conflict or a compatibility problem with the PCI chip set in your PC. Check for possible IRQ conflicts first. For the latest compatibility information, contact Imagenation Technical Support (see Technical Support, on page 20).

Make sure that you are excluding a 4 KB block of upper memory in your CONFIG.SYS file (see Step 1 on page 12 of the installation instructions).
If you see the message *This graphics card is not VESA compatible* when you run PXCVU, you aren’t using a VESA-compatible display driver. Check the documentation for your display controller board to see if a VESA-compatible driver is available.

If you see only a few lines of video at the top of the picture in PXCVU, the PCI bus is being overloaded or errors are occurring. Most Intel 486-based systems don’t have a PCI bus that is fast enough for the PXC200 frame grabber. Run the VGACOPY program to check for errors on the PCI bus.

If you haven’t set the IMAGENATION environment variable, PXCVU will display an error and won’t run. For information on the IMAGENATION environment variable, see *AUTOEXEC.BAT Changes for DOS, DOS/4GW, and Windows 3.1*, on page 14.

PXCVU will fail to run if the file DOS4GW.EXE is not accessible through your PATH environment variable.

**Slow Video Display Performance**

When you’re displaying video on the screen, the amount of memory on the VGA display controller card can affect the performance. With some display controllers, adding memory to the display controller will improve the performance.

**Windows Hangs or Crashes on Boot**

This can be caused by an interrupt conflict. Check to make sure you have an IRQ available and that no ISA device is trying to use the same IRQ that any PCI device is trying to use.
Imagenation offers free technical support to customers. If the PXC200 board appears to be malfunctioning, or you’re having problems getting the library functions to work, please read the appropriate sections in this manual. If you still have questions, contact us, and we’ll be happy to help you.

When you contact us, please make sure that you have the following information available:

- The revision number of your board. You can get this number by using the PXCREV program in DOS or either of the PXCDRAW programs in Windows. You must run the PXCREV program from DOS, not from a DOS window in Windows.

- The operating system you’re running: DOS, DOS/4GW, Windows 3.1, or Windows 95 (16-bit or 32-bit).

- The compiler you’re using, including the name of the manufacturer and the version number (for example, Borland C version 5.0).

**Voice:** 503-641-7408  
**Toll free:** 800-366-9131  
**Fax:** 503-643-2458  
**BBS:** 503-626-7763  
**CompuServe:** 75211,2640  
**Internet:** support@Imagenation.com  
www.imagenation.com

The 24-hour BBS and the Imagenation World Wide Web site (www.imagenation.com) always have the latest versions of the Imagenation software. Check anytime for software updates.
The PXCVU Application

This chapter describes the PXCVU application program for DOS. PXCVU is a basic frame grabber application that lets you control the features of your PXC200 frame grabber without writing your own application program. You can use PXCVU to capture frames or fields, write frames to disk files, change the video source, and to set the brightness, contrast, hue, and saturation.

Setting Up PXCVU

To run PXCVU, you must have the IMAGENATION environment variable set to point to the directory containing PXCVU.HLP and PXCVU.INI. PXCVU.HLP contains the text of the help screens you can access from PXCVU. PXCVU.INI is an optional file that contains initialization values for the application.

If you let the DOS Install or Windows Setup programs copy the files from the diskette and make the required changes to your system files, you’re ready to run PXCVU. If not, see AUTOEXEC.BAT Changes for DOS, DOS/4GW, and Windows 3.1, on page 14, for the required settings.
Starting PXCVU

Make sure you have a video source connected to your PXC200 board before starting the PXCVU program.

To run PXCVU, execute the following at the DOS command line (do not run PXCVU in a DOS window in Windows 3.1):

```
c:\pxc2\dos\pxcvu
```

If you see a display like that shown on page 23, the PXCVU program has started correctly. Otherwise, see Troubleshooting, on page 17.

Running PXCVU with More Than One Frame Grabber

If you have more than one frame grabber installed in your system, PXCVU will use the first frame grabber that it finds. To specify a particular frame grabber, follow the command with the number of the frame grabber:

```
c:\pxc2\dos\pxcvu  n
```

Frame grabbers are numbered sequentially starting with $n = 0$. Due to the nature of the PCI bus, the number of the frame grabber won’t necessarily correspond to the PCI bus slot in which the frame grabber is installed. To determine the correct number, $n$, of each frame grabber, you’ll just have to try the PXCVU application with different values for $n$ and observe the video displayed to identify the source.
Using PXCVU

The screen for the PXCVU application looks similar to the picture below:

If you have an active video source when you start PXCVU, the video should appear in the Video Window as soon as you start the program.

The Status Line below the video window shows you the current selections for the image displayed in the Video Window, the type of grab, and the starting field.

Definitions for functions keys are shown in the lower left corner:

- **F1 HELP**—Press F1 to get help on the currently-selected menu item.
- **F2 GRAB**—Press F2 to grab a frame using the current grab mode.
• **F3 ACQUIRE**—Press F3 to turn continuous acquire mode on or off.

• **F4 INFO**—Press F4 to display the hardware revision number and serial number for the board, the image size, and the screen size.

The **Main Function Menu** gives you more detailed control of the board. A short explanation of the currently-highlighted menu item is shown at the bottom of the screen. For help on a menu item, move the highlight to the item using the arrow keys, and press F1 for Help. The features listed in the menu are also explained in more detail in Chapter 4, *Programming the PXC200*, on page 25.
Programming the PXC200

This chapter describes how to write your own software programs for the PXC200 using the functions provided in the PXC200 software libraries. The chapter begins with an overview of how the libraries are organized, followed by information about programming for specific operating systems, and about using specific programming languages. The remainder of the chapter describes how to use the functions in the libraries to perform the basic steps required to capture images and access the image data, plus optional features you can use.

Library Organization

The PXC200 software is implemented as a set of libraries:

**PXC200 Frame Grabber Library**—Includes the functions you’ll use to control the frame grabber, including capturing images, setting image resolution, switching video inputs, and setting image contrast, brightness, hue, and saturation. Chapter 5, *PXC200 Library Reference*, on page 69, describes the syntax and other details for each function.
Frame Library—Includes the functions you’ll use to access captured image data and to read and write image files. Chapter 6, Frame Library Reference, on page 101, describes the syntax and other details for each function.

DOS VGA Video Display Library—A DOS-only library that includes functions for controlling the VGA display, creating a menu-style user interface, and drawing basic graphic primitives. This library is not included in the current chapter, but is described in Chapter 7, The VGA Video Display Library, on page 119.

Operating System Specifics

Follow the guidelines in this section for compiling, linking, and running PXC200 programs.

You can put c:\pxc2\lib and c:\pxc2\include in your environment variables for Microsoft, or in your TURBOC.CFG file for Borland, or in your integrated development environment (IDE) search list.
DOS Programming

The following table summarizes operating system specifics for compiling, linking, and running C programs under DOS:

### DOS 16-bit Programs

<table>
<thead>
<tr>
<th>Header Files</th>
<th>Libraries</th>
<th>Runtime, Memory, and Installation Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>PXC200.H</td>
<td>Borland:</td>
<td>For required changes to AUTOEXEC.BAT, see Changes to System Files for DOS, DOS/4GW, and Windows 3.1, on page 14.</td>
</tr>
<tr>
<td>FRAME.H</td>
<td>PXC2_LB.LIB</td>
<td></td>
</tr>
<tr>
<td>VIDEO.H*</td>
<td>FRAME_LB.LIB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VIDEO_LB.LIB*</td>
<td></td>
</tr>
<tr>
<td>Microsoft 7+:</td>
<td>PXC2_LM.LIB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FRAME_LM.LIB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VIDEO_LM.LIB*</td>
<td></td>
</tr>
</tbody>
</table>

* The VIDEO files are described in Chapter 7, The VGA Video Display Library, on page 119.

### Watcom DOS and DOS/4GW Programs

<table>
<thead>
<tr>
<th>Header Files</th>
<th>Libraries</th>
<th>Runtime, Memory, and Installation Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>PXC200.H</td>
<td>16-bit: PXC2_LW.LIB</td>
<td>For required changes to system files, see Changes to System Files for DOS, DOS/4GW, and Windows 3.1, on page 14.</td>
</tr>
<tr>
<td>FRAME.H</td>
<td>PXC2_LW.LIB</td>
<td></td>
</tr>
<tr>
<td>VIDEO.H*</td>
<td>FRAME_LW.LIB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VIDEO_LW.LIB*</td>
<td></td>
</tr>
<tr>
<td>32-bit:</td>
<td>PXC2_FW.LIB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FRAME_FW.LIB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VIDEO_FW.LIB*</td>
<td></td>
</tr>
</tbody>
</table>

* The VIDEO files are described in Chapter 7, The VGA Video Display Library, on page 119.
Windows 3.1 Programming

The following table summarizes operating system specifics for compiling, linking, and running C programs under Windows 3.1:

<table>
<thead>
<tr>
<th>Header Files</th>
<th>Libraries</th>
<th>Runtime, Memory, and Installation Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>PXC200.H</td>
<td>ILIB_31.LIB</td>
<td>PXC2.VXD, PXC2_31.DLL, and FRAME_31.DLL needed for runtime. For VxD installation, see DOS, DOS/4GW, and Windows 3.1 Software Installation, on page 12.</td>
</tr>
<tr>
<td>FRAME.H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Windows 95 Programming

The following tables summarize operating system specifics for compiling, linking, and running C programs under Windows 95:

---

Windows 95 16-bit programs

<table>
<thead>
<tr>
<th>Header Files</th>
<th>Libraries</th>
<th>Runtime, Memory, and Installation Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>PXC200.H</td>
<td>ILIB_31.LIB</td>
<td>PXC2.VXD, PXC2_31.DLL, and FRAME_31.DLL needed for runtime. For VxD installation, see Windows 95 Registry Changes, on page 16.</td>
</tr>
<tr>
<td>FRAME.H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Any DLLs your application uses should be in the Windows SYSTEM directory or in your path.

Programming in a Multithreaded, Multitasking Environment

Windows 95 is a multithreaded, preemptive multitasking operating system. In such systems, using empty loops to wait for events slows the system dramatically by wasting processing time that could be used by other threads. For example, an empty loop like this might be used in a Windows 3.1 program:

```c
while (!IsFinished(fgh,qh))
    ;
```

In Windows 95, such an empty loop is not very efficient, so an alternate function, `WaitFinished()`, is included in the library for such applications:

```c
WaitFinished(fgh,qh);
```

The `WaitFinished()` function uses system synchronization objects to prevent the current thread from executing while the wait is in progress. Since all queued operations finish executing during vertical blank, polling only once per vertical blank is just as accurate as polling more often, but significantly improves system performance. `WaitVB()` can be used to add delays to polling loops to improve system performance.

---

**Windows 95 32-bit programs**

<table>
<thead>
<tr>
<th>Header Files</th>
<th>Libraries</th>
<th>Runtime, Memory, and Installation Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>PXC200.H</td>
<td>ILIB_95.LIB</td>
<td>PXC2.VXD, PXC2_95.DLL, and FRAME_95.DLL needed for runtime. For VxD installation, see <em>Windows 95 Registry Changes</em>, on page 16.</td>
</tr>
<tr>
<td>FRAME.H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

For VxD installation, see *Windows 95 Registry Changes*, on page 16.
Scheduling multiple threads to handle complicated image processing tasks might make programming significantly easier, and the PXC200 library does allow multithreading with one important exception. A program should not allow two different threads of execution to access the same frame grabber at the same time. Doing so could put the frame grabber into an unpredictable state, and possibly cause DMA transfers to be misdirected. This limitation can’t be fixed by simply wrapping each frame grabber control function in a mutual exclusion object, since many functions permanently change the state of the frame grabber. In general, you should make sure that only one thread is responsible for each frame grabber. Functions that do not directly access the frame grabber, such as the file I/O functions and the buffer manipulating functions, are safe to multithread as long as the usual care is taken to be sure that the data they access does not become invalid.

Programming Language Specifics

This section discusses specific information about writing programs in C and in Visual Basic.

Programming in C

If you’re using third-party libraries or multiple frame grabber libraries in developing your programs, the same function name might exist in more than one library, causing a symbol collision. The PXC200 software libraries are designed to help you avoid symbol conflicts.

When you initialize a library, you can specify a unique library name that you’ll use for calling all functions in that library. When you make function calls to that library, you call a function as a member of a structure. The name of the structure is the library name you used to initialize the library. The following example shows how you might initialize the
Chapter 4  Programming the PXC200

PXC200 frame grabber library using the library name \texttt{fg} and then call the \texttt{AllocateFG()} function, which is used to get a handle for a frame grabber:

\begin{verbatim}
imagenation_OpenLibrary("pxc2_95.dll", fg, sizeof(fg));
handle = fg.AllocateFG(0);
\end{verbatim}

The first line initializes the frame grabber library. The second parameter, \texttt{fg}, is the library name you have chosen. The second line calls the \texttt{AllocateFG()} function as a member of a structure called \texttt{fg}.

The same technique works with the Frame library and the DOS VGA Video Display library. Just be sure to choose unique library names for each library you initialize.

Visual Basic Programming

The Windows DLLs were designed to make the function calls as uniform as possible, whether you’re programming in C or in Visual Basic. Since the syntax and keywords in Visual Basic differ from those of C, before you start programming in Visual Basic, you should look at the Visual Basic function definitions in the .BAS file.

There are a few things you should keep in mind when using Visual Basic with the DLL functions:

\textbf{Accessing frame data}—In C, you can use the pointer returned by \texttt{FrameBuffer()} to access the image data in the frame. Visual Basic doesn’t use pointers, so you must use the functions \texttt{GetPixel()}, \texttt{GetColumn()}, \texttt{GetRectangle()}, and \texttt{GetRow()} to access the data in a frame. The \texttt{FrameBuffer()} function exists in Visual Basic for situations where you need to get a pointer to pass to other Windows API functions that are designed to work with pointers.

\textbf{.BAS File}—You must include the appropriate .BAS file in all projects you build using the PXC200 DLL functions. The .BAS file includes all the declarations you’ll need to work with the DLLs. For 16-bit pro-
grams in Windows 3.1 and Windows 95, include WPXC2_31.BAS; for 32-bit programs in Window 95, include WPXC2_95.BAS.

Buffers: Strings vs. Integers in Visual Basic 3.0

A Visual Basic 3.0 application can pass buffers to functions as a string by value (ByVal buf As String) or as an integer array by reference (buf As Integer). If you pass a buffer as a string, it’s easy to put values into the buffer or take them out. To insert an element into a string, use the Chr$() function on that element, and insert the result in the string with the Mid$() function. The disadvantage to this method is that Visual Basic string manipulation is fairly slow.

If you pass a buffer as the first element of an integer array, you must pack two pixel values into each integer as you insert the values into the array, and unpack them when you remove elements from the array. This is faster, but somewhat more complicated.

The interfaces of the following functions have been defined in WPXC2_31.BAS using strings.

- GetColumn()  PutColumn()
- GetPixel()   PutPixel()
- GetRectangle() PutRectangle()
- GetRow()     PutRow()

If you want to change the interface, you should edit the WPXC2_31.BAS file and replace occurrences of ByVal buf As String with buf As Integer.

Buffers in Visual Basic 4.0

Visual Basic 4.0 includes a Byte type, which is equivalent to the unsigned char type that the DLLs expect for buffers. Thus, the
WPXC2_95.BAS file uses `buf As Byte` in the function definitions. To pass a buffer to the DLL, just pass the first element of your declared `Byte` array.

**Using the Visual Basic Development Environment**

**Caution**

*Do not use the End button in the Visual Basic development environment to terminate your application. The End button terminates a program immediately, without executing the Form_Unload function or any other functions. If you use the End button to exit a program, you must use the PXCLEAR utility to free any frame grabbers that your program allocated.*

**Displaying Video in Visual Basic Applications**

The PXC200 software includes a Video Display DLL that makes displaying captured images in a window quite simple. For more information, see *Using the Video Display DLL*, on page 66.

**Typical Program Flow**

A program for capturing an image with the frame grabber contains at least the following basic tasks:

1. Initialize the libraries.
2. Request access to the frame grabber.
3. Set up the destination for the captured image data.
4. Capture the image.
5. Release the frame grabber.
6. Exit the library.
In addition, a program might include:

- Selecting a video source, if you have more than one.
- Adjusting attributes of the image, such as hue and saturation.
- Specifying scaling and cropping for the image.
- Using the trigger signal to initiate a capture.
- Queuing functions so the program can do other work while the frame grabber is busy.
- Accessing the captured image data for analysis or processing.

The following sections describe these features in more detail and show you how to use the library functions to accomplish each of these tasks.

**Initializing and Exiting Libraries**

Before calling any other library functions, you must explicitly initialize each library by calling the appropriate `OpenLibrary()` function. Following your last call to a library, before your program terminates, you must call the appropriate `CloseLibrary()` function. The actual function names are specific to the operating system and language you are using, and are described in the following sections.

**C and Windows Programs**

The `OpenLibrary()` and `CloseLibrary()` functions for the PXC200 Frame Grabber library under Windows 3.1 and Windows 95 are:

```c
imagenation_OpenLibrary("pxc200.dll", &iface,
sizeof(iface))
imagenation_CloseLibrary("pxc200.dll", &iface,
sizeof(iface))
```
The OpenLibrary() and CloseLibrary() functions for the Frame library under Windows 3.1 and Windows 95 are:

```c
imagenation_OpenLibrary("frame.dll", &iface, sizeof(iface))
imagenation_CloseLibrary("frame.dll", &iface, sizeof(iface))
```

Where `interface` is the name you will use for the structure for calling library functions. For more information on this calling convention, see *Programming in C*, on page 30.

In the Windows versions of the libraries, the interrupt handlers are installed by the low-level device drivers; the virtual device drivers (VxDs) in Windows 3.1 and Windows 95. The low-level device driver is loaded when you start Windows, and is uninstalled when you exit Windows.

### C and DOS Programs

The OpenLibrary() and CloseLibrary() functions for the PXC200 Frame Grabber library and the Frame library for C programs under DOS are:

```c
PXC200_OpenLibrary(&iface, sizeof(iface))
PXC200_CloseLibrary(&iface, sizeof(iface))
FRAME_OpenLibrary(&iface, sizeof(iface))
FRAME_CloseLibrary(&iface, sizeof(iface))
```

Where `interface` is the name you will use for the structure for calling library functions. For more information on this calling convention, see *Programming in C*, on page 30.

In the DOS and DOS/4GW versions of the library, initializing the library installs an interrupt handler that is needed for frame grabber communication, and exiting the library uninstalls the interrupt handler. If your pro-
gram crashes or terminates without calling CloseLibrary(), you will probably need to reboot your system, as it may be in an unstable state.

Visual Basic and Windows Programs

The OpenLibrary() and CloseLibrary() functions for the PXC200 Frame Grabber library for Visual Basic programs under Windows 3.1 and Windows 95 are declared and called as:

```vba
declare function OpenLibrary lib "pxc200.dll" (ByVal iface as Long, ByVal count as Long) as Integer

declare sub CloseLibrary lib "pxc200.dll" (ByVal iface as Long)
```

```vba
OpenLibrary(0,0)
CloseLibrary(0)
```

For the Frame library, substitute “frame.dll” for “pxc200.dll” in the declarations.

Troubleshooting OpenLibrary()

Check the return value from OpenLibrary() to make sure the function was successful (non-zero = success). OpenLibrary() functions will fail under Windows if the DLLs or VxDs are not present.

The OpenLibrary() functions for the Frame library and the DOS VGA Video Display library should fail only when the system has insufficient memory; each function allocates a small amount of memory for internal data structures.

OpenLibrary() for the PXC200 Frame Grabber library can fail under the following conditions:

- The PCI BIOS does not exist or is malfunctioning. Your computer probably has a hardware problem.
• The PCI BIOS was unable to assign an IRQ to the frame grabber. You may need to modify your CMOS settings to make more IRQs available to the PCI BIOS.

• There is no suitable memory block in upper memory. In DOS, each frame grabber requires a contiguous 4KB block of upper memory, and OpenLibrary() will try to find such a block. For more information, see, DOS, DOS/4GW, and Windows 3.1 Software Installation, Step 1 on page 12.

• There is insufficient conventional memory. OpenLibrary() allocates a small amount of storage for internal data structures.

• There are no Imagenation frame grabbers in your computer, or they are malfunctioning.

Some of these error conditions can be detected by calling the CheckError() function.

**Requesting Access to Frame Grabbers**

A process must have a handle to a frame grabber to communicate with it. The AllocateFG() function returns a handle to the specified frame grabber if it exists and hasn’t already been allocated to another process.

FreeFG() frees the specified frame grabber, so it can be allocated by other processes.

Any process with a valid frame grabber handle can communicate with that frame grabber. One process can get a handle to the frame grabber using AllocateFG(), and other processes can use the same handle. Keep in mind that any process can change the state of the frame grabber, so a given process can’t assume the state of the frame grabber will remain as that process last left it. When more than one process has simultaneous
access to the same frame grabber, you must coordinate the processes accordingly.

If you’re using multiple frame grabbers in a single system, you’ll need to determine which frame grabber is which. Due to the design of the PCI bus, bus slot zero doesn’t necessarily correspond to frame grabber zero, and the number of the frame grabber in a particular bus slot can vary between different operating systems. You can determine which frame grabber is which by connecting a video source to only one frame grabber and then using the PXCVU program (or your own program) to switch between frame grabbers.

When the AllocateFG() function fails, it is often because another process is using the frame grabber, or because a program terminated unexpectedly, leaving a frame grabber allocated. In Windows 3.1 and Windows 95, you can use the PXCLEAR program (described below) to free all frame grabbers. For other operating systems, you might need to reboot your system.

### The PXCLEAR Utility

PXCLEAR is a utility that frees frame grabbers. It works with both Windows 3.1 and Windows 95. If a program terminates unexpectedly and does not call its exit procedures, any frame grabbers that it had allocated will still be allocated, preventing any other programs from using them. PXCLEAR tells you which frame grabbers are currently in use, and gives you the option of freeing all of them. It can’t be used to free individual frame grabbers; it frees all frame grabbers in the system or none of them.

You should not use PXCLEAR as a general tool for freeing frame grabbers in preference to freeing them in your program’s exit procedures. You also should not use PXCLEAR while any program that is using a frame grabber is still running.
Note
The Visual Basic development environment End button terminates a running program immediately, without executing the Form_Unload function (or any other). If you use the End button to exit a program, you must use the PXCLEAR utility to free any frame grabbers that your program allocated.

Setting the Destination for Image Captures

Library functions send the captured image data to frames. Don’t confuse this use of the term frame with the term video frame, which refers to a video image consisting of two fields. A frame stores an image and some basic information about it, including the image height, width, and number of bits per pixel.

Allocating and Freeing Frames

You can create a frame for capturing images in two ways: with AllocateBuffer() or with AllocateAddress(). The Frame library (see Chapter 6, Frame Library Reference, on page 101) includes two additional functions for allocating frames for uses other than grabbing frames: AllocateFlatFrame(), and AllocateMemoryFrame().

AllocateBuffer() allocates storage for a frame in main memory and calculates the physical address for the storage location, so the frame grabber can send image data directly to the buffer via DMA. AllocateAddress() is discussed in Sending Images Directly to Another PCI Device, below.
When you allocate storage for a frame you specify the type of pixel data that will be stored in the frame using one of the types listed below.

<table>
<thead>
<tr>
<th>Pixel Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBITS_Y8</td>
<td>8-bit grayscale.</td>
</tr>
<tr>
<td>PBITS_Y16*</td>
<td>16-bit grayscale.</td>
</tr>
<tr>
<td>PBITS_Yf*</td>
<td>Floating point grayscale.</td>
</tr>
<tr>
<td>PBITS_RGB15</td>
<td>5 bits each for red, green, and blue, plus one bit for the alpha value.</td>
</tr>
<tr>
<td>PBITS_RGB16</td>
<td>5 bits each for red and blue; 6 bits for green.</td>
</tr>
<tr>
<td>PBITS_RGB24</td>
<td>8 bits each for red, green, and blue.</td>
</tr>
<tr>
<td>PBITS_RGB32</td>
<td>8 bits each for red, green, and blue, plus 8 bits for the alpha value.</td>
</tr>
<tr>
<td>PBITS_RGBf*</td>
<td>A floating point number each for red, green, and blue.</td>
</tr>
<tr>
<td>PBITS_YUV422</td>
<td>8 bits for Y and 8 bits for CrCb.</td>
</tr>
<tr>
<td>PBITS_YUV444*</td>
<td>8 bits each for Y, Cr, and Cb.</td>
</tr>
<tr>
<td>PBITS_YUV422P</td>
<td>YUV422 in planar format.</td>
</tr>
<tr>
<td>PBITS_YUV444P*</td>
<td>YUV444 in planar format.</td>
</tr>
</tbody>
</table>

* These types aren’t supported by the PXC200 frame grabber and can’t be allocated with AllocateBuffer(). However, they can be useful in image processing. For more information, see Accessing Captured Image Data, on page 64.

Captured video is digitized by the frame grabber in YCrCb 4:2:2 format and then converted to the specified pixel type before being transferred to the frame.

For most pixel data types, the buffer is organized as an array of pixels, where each pixel is represented by the data structure described above. (See the PXC200.H file for the actual structure declarations.) The YUV422P and YUV444P are both planar types. In these planar types, the
data is organized in three planes: plane 0 for the Y component, plane 1 for the Cr component, and plane 2 for the Cb component.

When the `AllocateBuffer()` function fails, it means that you don’t have enough memory allocated for frame buffers. Try freeing any frame buffers that you don’t need. If calls to `AllocateBuffer()` still fail, try rebooting your system.

When you want to free memory previously allocated by `AllocateBuffer()` or `AllocateAddress()`, use the `FreeFrame()` function. Do not try to free a buffer when data is being transferred to it by queued functions or by `GrabContinuous()`.

**Sending Images Directly to Another PCI Device**

Some devices, such as high-end PCI video cards, have a physical address where they can receive data via direct memory access (DMA). (Don’t confuse this physical address with the logical addresses or pointers that software normally uses. A physical address is a low-level construct that the hardware uses in its internal communication, and is independent of the operating system.) This provides a high-performance path for capturing images directly to the device. For example, some PCI video cards have a flat addressing mode that allows DMA transfers to the card without having to swap pages of video memory in and out. With such a card, you should be able to display video in real time. To find out if your video card supports flat addressing, and how to determine the physical address for the card, refer to the documentation that came with the card or contact the manufacturer.

Use `AllocateAddress()` to create a frame for a specified physical address, where the frame grabber will copy the image data. `AllocateAddress()` does not allocate any storage for an image buffer, since the data will be sent directly to the physical address.
Caution

Use transfers to PCI devices only if you are familiar with DMA data transfers. DMA transfers bypass the operating system, so there is no opportunity to check for an incorrect address, and no protection faults are issued. An incorrect address could cause the operating system to crash. Since you are bypassing the window management routines of Windows, you can also corrupt the windows of other programs.

AllocateAddress() doesn’t allocate any storage for an image buffer, so the FreeFrame() function frees only the memory used by the frame structure.

Grabbing Images

The library includes two functions for grabbing images to frames: Grab() and GrabContinuous().

Grab() digitizes video and copies the data to the specified frame. You can specify which video field the capture should start on, whether to digitize one field or both, and when to execute (see Using Flags with Function Calls, on page 57).

Grab() starts digitizing as soon as the command is processed by the frame grabber.

GrabContinuous() continuously digitizes and transfers video to the specified frame.

The frame grabber automatically changes to the correct pixel format for the destination frame whenever a Grab(), GrabContinuous(), or SwitchGrab() function is executed. Switching to a different pixel format takes about one field time. When the change occurs because of a Grab, this delay becomes part of the latency for the Grab. You can use the SetPixelFormat() function to preset the expected pixel format and minimize the latency in the Grab function.
If the PCI bus is overloaded, it’s possible for captured data to be corrupt. Although the Grab functions can’t determine when data is being corrupted, CheckError() will return the value ERR_CORRUPT.

The most common reasons the Grab functions fail are:

- The frame grabber handle or the frame buffer handle is invalid.

- The image specified by SetWidth() or SetHeight() (or the default image size) is too large in width or height for the frame buffer.

If the Grab functions execute successfully, but don’t produce the image you expect, the most common reasons are:

- If the captured image is all black or all blue, be sure to check that your video source is attached to the frame grabber and that the iris on the video camera is open.

- If you’re using a system with an Intel Pentium Pro processor, you might not be able to read valid data from a frame buffer in system memory immediately after grabbing the image. This is due to the processor caching the data, rather than writing the data immediately to memory. Try inserting a delay in your program before reading the data.

- If you get only a few lines of valid video at the top of an image you’ve grabbed to a frame buffer in system memory, the PCI bus is being overloaded or errors are occurring on the bus. Most Intel 486-based systems don’t have a PCI bus that is fast enough for PXC200 frame grabbers. Run the VGACOPY program to check for errors on the PCI bus.

- The frame grabber can’t produce the image specified by SetHeight(), SetWidth(), SetXResolution(), and SetYResolution() (see Scaling and Cropping Images, on page 49).
Selecting Video Inputs

Each frame grabber can have up to four video sources connected directly to it. The \texttt{SetCamera()} function selects one of the four video inputs to be digitized. The \texttt{GetCamera()} function returns the currently selected input.

By default, PXC200 frame grabbers automatically detect the video format (NTSC or PAL/SECAM) on the active camera input. If you need to determine the video format of the current video source for use in your program, you can use the \texttt{VideoType()} function.

When you switch from one video input to another, there may be a delay before the frame grabber can synchronize to the new video input. Three factors determine the time that it takes to synchronize to a video input once you've switched to it: input video type, whether the cameras are genlocked or not, and brightness levels. If the cameras are all the same video type, there should be a delay of no more than one field time before re-synchronization occurs; if they are also genlocked, there will be no appreciable delay. (Cameras of different video types can’t be genlocked.) If the cameras are not of the same video type, there may be a delay of as much as 2.5 seconds before re-synchronization occurs. If the brightness level differs between two cameras of the same video type, there may be some additional delay when switching.

If the delay in detecting a video format change is too long, you can set the video type directly by using the \texttt{SetVideoDetect()} function to specify the type of video the frame grabber should expect. This forces the frame grabber to digitize the incoming video based on the video format you specify. You can specify the video format from a list of optional formats for NTSC, PAL, and SECAM. The \texttt{GetVideoDetect()} function returns the currently set video format.
Adjusting the Video Image

The PXC200 provides a variety of adjustments you can make to the video signal to change the way the signal is processed and the appearance of the resulting captured image.

Setting Contrast and Brightness

The contrast adjustment lets you lighten or darken the image. It’s like a gain control on the monochrome part of the video signal. Contrast can be adjusted from 0.0 to 2.0. A contrast value of 1.0 leaves the signal unchanged. You set the contrast adjustment using the SetContrast() function. The GetContrast() function returns the current contrast adjustment.

The brightness adjustment acts as an offset for the monochrome part of the video signal. The brightness can be adjusted from -0.5 to +0.5. A value of +0.5 increases the digitized value of black to medium gray, and a value of -0.5 brings the digitized value of white to medium gray. A value of 0.0 leaves the digitized value unchanged. You set the brightness adjustment using the SetBrightness() function. The GetBrightness() function returns the current brightness adjustment.

Setting Hue and Saturation

The hue adjustment lets you shift the colors in the image. Adjusting the hue is like rotating the color wheel, shown below. Positive values for the
hue adjustment shift colors displayed as red toward yellow and green; negative values shift reds toward magenta and blue.

You set the hue adjustment using the SetHue() function. The GetHue() function returns the current hue adjustment. For NTSC video, you can adjust the hue from -90° to +90°. Because of the nature of PAL/SECAM signals, hue adjustments can’t be made.

The saturation adjustment lets you change the intensity of the colors in the image. It’s like a gain control on the color part of the video signal. Saturation can be adjusted from 0.0 to 2.0, with a value of 1.0 being normal. A saturation value of zero removes all color, leaving a monochrome image. You set the saturation adjustment using the SetSaturation() function. The GetSaturation() function returns the current contrast adjustment.

### Setting the Video Level

The video level adjustment lets you set the expected amplitude range of the video signal from the bottom of the video sync portion of the signal to bright white. (See the drawing, below, of a video signal for a single horizontal line of video.) This value is normally 1.3 V, but can be set to any value in the range zero to 2.5 V for video sources that don’t produce signals at the normal value. You set the video level using the
The `SetVideoLevel()` function. The `GetVideoLevel()` function returns the current video level adjustment.

**Setting Luma Controls**

The term *luma* refers to the monochrome part of the video signal. The luma control lets you specify several features the frame grabber can apply to processing the monochrome part of the video signal:

- **Low Filter**—A low-pass filter that reduces high-frequency information in the video signal.

- **Core Function**—Causes all video below a specified level to be digitized to black. Coring can improve the apparent contrast of some dark images.

- **Gamma Correction**—Provides gamma correction for RGB video output. For NTSC, a gamma value of 2.2 is used; for PAL, the gamma value is 2.8.

- **Comb Filter**—Activates a comb filter to reduce artifacts in the monochrome signal caused by crosstalk from the color signal.
**Imagenation**

**Peak Filter**—Activates a filter that amplifies high frequencies. This filter can sharpen edges in a blurry image, but might also cause artifacts on edges that are already sharp.

You set the luma control features using the `SetLumaControl()` function. The `GetLumaControl()` function returns the current setting for each luma control feature.

**Setting Chroma Controls**

The term *chroma* refers to the color part of the video signal. The chroma control lets you specify several features the frame grabber can apply to processing the color part of the video signal:

- **S-Video**—Tells the frame grabber that the video signal is an S-Video signal with separate color and monochrome channels, rather than a composite video signal. This causes the frame grabber to extract the color information from the separate video signal rather than extracting it from the composite signal.

- **Notch Filter**—Activates a filter to remove the color burst signal from the video signal before the signal is digitized. This prevents color artifacts from appearing in composite video, while still allowing the color information to be digitized.

- **Automatic Gain Control**—Activates automatic gain control (AGC) for color saturation to compensate for non-standard color signals.

- **Monochrome Detect**—Sets the color signal to zero when the board detects a missing or weak color burst signal.

- **Comb Filter**—Activates a comb filter to reduce color artifacts.
You set the chroma control features using the `SetChromaControl()` function. The `GetChromaControl()` function returns the current setting for each chroma control feature.

**Scaling and Cropping Images**

The resolution of full-size digitized images depends on the video format and the aspect ratio of your screen and pixels. Typical computer monitors have an aspect ratio of 4 x 3 and use square pixels. Conventional television monitors use rectangular pixels. Typical resolutions for several common formats are given below:

<table>
<thead>
<tr>
<th>Video Format</th>
<th>Resolution for Full-Size Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTSC square pixels</td>
<td>640 x 486</td>
</tr>
<tr>
<td>NTSC rectangular pixels</td>
<td>720 x 486</td>
</tr>
<tr>
<td>PAL/SECAM square pixels</td>
<td>768 x 576</td>
</tr>
<tr>
<td>PAL/SECAM rectangular pixels</td>
<td>720 x 576</td>
</tr>
</tbody>
</table>

You can digitize images at these maximum resolutions, or you can scale and crop the images, which saves memory and bandwidth for transferring and processing images.

**Scaling Images**

PXC200 frame grabbers can scale the video image by interpolating pixel values along both the horizontal and vertical axes. To scale an image, you simply specify the number of pixels you want along the horizontal and vertical axes using the `SetXResolution()` and `SetYResolution()` functions. The `GetXResolution()` and `GetYResolution()` functions return the current values. You can scale images down to approximately 1/16 size.
Note
When working with small values for Y resolution, you can often get better image quality by specifying twice the desired Y resolution and using the SINGLE_FLD flag with the Grab() function. This eliminates field blur and other problems related to interlacing.

Cropping Images

In addition to scaling images, you can crop images vertically and horizontally. You crop an image in width by specifying the starting column and number of columns to keep, using the SetLeft() and SetWidth() functions. You crop an image in height by specifying the starting row and number of rows to keep using the SetTop() and SetHeight() functions. You can get the current values with GetLeft(), GetWidth(), GetTop(), and GetHeight().

The figure on page 51 below shows an example of an NTSC image that has been scaled to 32 pixels by 26 pixels. If you want to crop the image to get a rectangular image 16 pixels by 16 pixels from the center of the scaled image, you would specify the cropping parameters as left = 8, width = 16, top = 5, and height = 16.

For all video formats, the default starting row is row four, and the default number of rows is 480. For PXC200 frame grabbers, row zero of the video image is the first row of valid video.

Note
NTSC and PAL/SECAM video signals have only a half row of valid video on the first and last rows of each frame. The first line (row zero for both formats) contains valid video for only the last half of the row. The last line (row 485 for NTSC, row 575 for PAL/SECAM) contains valid video for only the first half. If you include either of these rows in your image data, the entire row will be sampled.
Timing the Execution of Functions

The PXC200 software library includes some advanced features for applications that are time-critical. These features let you determine whether functions should be executed immediately, or if they should be placed in a queue to execute asynchronously while the program proceeds.
_QUEUEED Functions

Frame grabber applications often include a loop that repeatedly grabs a frame and then processes the information in it. For example:

```c
for (;;)
{
    Grab(fgh, fbuf, 0);
    Process_Image(fbuf); /* your function */
}
```

where `fgh` identifies the frame grabber, `fbuf` specifies the frame handle, and 0 indicates that Grab() is to use the default settings.

This technique of serially grabbing and processing frames is straightforward and easy to implement using the PXC200 library. However, there are disadvantages to this serial process:

- While the image is being processed, the frame grabber can’t grab images, and much of the video image data that the camera is receiving never gets processed.

- While the frame grab is occurring, the computer’s CPU can’t do any image processing and sits idle waiting for the next frame.

PXC200 frame grabbers transfer image data to a frame using direct memory access (DMA), which bypasses the computer’s operating system. DMA makes it possible to have the frame grabber moving data to one frame, while at the same time the application is processing image data in another frame. The library has been designed to take advantage of this parallel activity. Certain functions can be designated as _queued_, by specifying the QUEUED flag in the function call (see *Using Flags with Function Calls*, on page 57). A queued function will return as soon as it puts the necessary information in the queue, without waiting for the operation to execute. This frees the application to continue processing.
Here’s an example of how you might use this capability:

```c
int grab1, grab2;
grab1 = Grab(fgh, fbuf1, QUEUED);
grab2 = Grab(fgh, fbuf2, QUEUED);
WaitFinished(fgh, grab1)
    ; /* wait until grab 1 has completed */
for (;;)
{
    ProcessImage(fbuf1);
    grab1 = Grab(fgh, fbuf1, QUEUED);
    WSaitFinished(fgh, grab2)
        ;/* wait until grab 2 has completed */
    ProcessImage(fbuf2);
    grab2 = Grab(fgh, fbuf2, QUEUED);
    WaitFinished(fgh, grab1)
        ;/* wait until grab 1 has completed */
}
```

The `WaitFinished()` function is used to pause until a function has completed. In the example above, once `WaitFinished()` indicates that the first `Grab()` is complete, the program starts processing the first image. `WaitFinished()` can check on a specific function in the queue (as in this example), or check to see if all functions in the queue are complete.

If your system has more than one frame grabber installed, each frame grabber has a separate queue, and `WaitFinished()` checks the appropriate queue based on the handle `fgh` that you specify.

## Synchronizing Program Execution to Video

The library has two functions, `Wait()` and `WaitVB()`, that can be used to synchronize program execution to incoming video:

- `WaitVB()` pauses until the end of the next vertical blank before returning. This is the most efficient way to synchronize program execution to video for non-queued functions.
**Imagenation**

Wait() can wait for the end of the next field, the end of the next frame (two complete fields), or the end of a specific field before returning. Wait() takes exactly as much time as a Grab() with the same parameters. Since the Wait() function can be queued, it is most useful for synchronizing queued functions to video.

You can also synchronize program execution based on the state of I/O lines (see *Digital I/O*, on page 57).

**Purging the Queue**

The KillQueue() function purges any pending functions in the queue and terminates any that are executing. This function is designed for error recovery and should only be used when the queue appears to have stopped processing functions.

The results of any functions in the queue when KillQueue() is called are undefined. For example, if a call to Grab() is in the queue when KillQueue() is called, the image data in the frame might not be valid.

**Immediate Functions**

You can specify that a function should only execute if there is nothing in the queue. The library provides the flag IMMEDIATE for this purpose. If a function specified as immediate executes when functions are in the queue, it will return failure without doing anything. Otherwise, the function will return when it has completed.
Function Timing Summary

The *queued* and *immediate* settings are not mutually exclusive. A function can be declared to be either one, neither, or both. The behavior of each setting is summarized below:

**Neither queued nor immediate.** executes when all functions in the queue have completed, and returns when execution is completed. This is the default.

**Queued.** Execution is deferred until previously queued functions have executed. The function returns immediately, and the program continues to the next statement. The frame grabber executes the queued instructions concurrently with the program’s execution of any non-frame grabber functions.

**Immediate.** Only executes if there are no functions in the queue. The function returns when execution is completed.

**Queued and Immediate.** Only executes if there are no functions in the queue. The function returns immediately, and program continues to the next statement. The frame grabber executes the queued instructions concurrently with any non-frame grabber functions. If there is a non-queued function in progress, the application doesn’t proceed until that function is complete.

Many applications don’t require the QUEUED and IMMEDIATE flags. If you don’t use either flag, the function executes as soon as the frame grabber has finished the previous operation, and the function returns when the frame grabber has finished executing it.
You can use the QUEUED and IMMEDIATE flags with any of the following functions:

- Grab()
- GrabContinuous()
- SetBrightness()
- SetContrast()
- SetCamera()
- SwitchCamera()
- SwitchGrab()
- Wait()
- WaitAllEvents()
- WaitAnyEvent()

These functions return a handle that can be used by IsFinished() and WaitFinished() to check their progress.

The following functions always wait until all functions in the queue have completed before executing:

- SetHeight()
- SetLeft()
- SetTop()
- SetWidth()
- SetXResolution()
- SetYResolution()
- SetChromaControl()
- SetLumaControl()
- SetIOType()
- SetPixelFormat()
- SetVideoDetect()
- SetVideoLevel()

All functions not listed here will execute when they are called and return when they have completed. They may execute concurrently with functions in the queue.
Using Flags with Function Calls

Several of the frame grabber control functions take a set of flag bits as one of their parameters. The possible flags are:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EITHER</td>
<td>Operation will start on the next field.</td>
</tr>
<tr>
<td>FIELD0</td>
<td>Operation will start on an even video field.</td>
</tr>
<tr>
<td>FIELD1</td>
<td>Operation will start on an odd video field.</td>
</tr>
<tr>
<td>SINGLE_FLD</td>
<td>Operation will only apply to one field.</td>
</tr>
<tr>
<td>IMMEDIATE</td>
<td>Operation will fail if the frame grabber is busy.</td>
</tr>
<tr>
<td>QUEUED</td>
<td>Operation will be queued for later processing.</td>
</tr>
</tbody>
</table>

Flags can be combined with the bitwise OR operator.

The default behavior ($flags = 0$) for a function that uses flags is:

- Wait until the frame grabber is not busy.
- Start on the next field.
- Process a two-field, interlaced frame (if the function processes an image).
- Return after the operation is complete.

Not all flags are relevant to each function that has a $flags$ parameter. For example, some functions, such as SetBrightness() and SetHue(), ignore the FIELD choice flags and always operate as if the EITHER flag was specified.

Digital I/O

The PXC200 frame grabber includes a single TTL-level digital input that lets you synchronize the frame grabber with other devices in the system.
To provide for future enhancements, the library functions include support for multiple input and output lines, even though currently only a subset of this functionality can be used with the PXC200.

**Controlling the Input Lines**

You can use the input lines to read information from an external device and to initiate actions in your program. For example, you could use an input line to trigger the frame grabber to capture an image on a signal from a camera or other external device.

**Setting Up and Reading the Input Lines**

You use the `SetIOType()` function to set up the input lines. You can set up an input line so that the state of the line will be set for any of the following conditions:

- **Rising signal**—signal changed from low to high.
- **Falling signal**—signal changed from high to low.
- **Input signal**—signal is high (the default).

The `GetIOType()` function returns the current type of an I/O line, as set by the `SetIOType()` function.

The `ReadIO()` function returns the current state of all I/O lines. Bit 0 represents the single input line on the PXC200.

**Using an Input Line as a Trigger**

Using an input line to initiate some action typically involves the following steps in a program:

1. Set up the line to change state when the signal on the line changes.
2 Queue a WaitAnyEvent() or WaitAllEvents() function to wait for the state of the line to change.

3 Queue a follow-on action to take place when the event has been detected.

You can program the **WaitAnyEvent()** function to watch the state of one or more input lines. When WaitAnyEvent() reaches the top of the queue, processing of the queue pauses until at least one of the watched lines is in the specified state; then, the next function in the queue is processed. For example, if the frame grabber had four input lines, numbered 0 - 3, and you set up WaitAnyEvent() to watch for lines 0 and 3 to be set, the program will pause as long as the states of both lines are clear. As soon as the state of either (or both) lines is set, the program will resume processing the queue.

A common use for a trigger input on the PXC200 is to initiate a capture when the trigger signal is detected. You can accomplish this with these two lines of code:

```c
WaitAnyEvent(fgh, fgh, 1, 1, QUEUED);
Grab(fgh, frh, flags);
```

The **WaitAllEvents()** function pauses processing of the queue until all of the watched lines are at the specified state. For example, if the frame grabber had four input lines, numbered 0 - 3, and you set up WaitAllEvents() to watch for lines 0 and 3 to be clear, the state of both lines must be clear before processing the queue resumes. As long as the state of at least one of the lines is set, processing the queue remains suspended.

You designate which lines WaitAnyEvent() and WaitAllEvents() should watch, and which state to watch for, by setting a *state* parameter and a
mask parameter in the function call. Both functions read the I/O lines, as `ReadIO()` would, and evaluate the expression:

\[(\text{ReadIO()} \ ^ \ !\text{state}) \ & \ \text{mask}\]

where “^” is the bitwise exclusive OR operator, and “&” is the bitwise AND operator. This lets you designate the state (0 or 1) to watch for on each line and limits the lines watched to those with a value of 1 in the mask. Bit 0 in both state and mask represents the single input line 0 on the PXC200.

The `WaitAnyEvent()` and `WaitAllEvents()` functions also let you use the I/O lines on one frame grabber to trigger events on another frame grabber by specifying the handles for the two frame grabbers in the function call.

When processing continues, `WaitAnyEvent()` and `WaitAllEvents()` set a switch and clear the state of the input line. The switch is set to the number of the highest line that had a state of 1. If the state of more than one of the watched lines is a 1, `WaitAnyEvent()` clears only the state of the highest-numbered line, while `WaitAllEvents()` clears all lines. For example, if the frame grabber had four input lines, numbered 0 - 3, and if both lines 0 and 3 have a state of 1, the switch will be set to 3; `WaitAnyEvent()` will clear only the state of line 3, while `WaitAllEvents()` will clear both lines 0 and 3.

The follow-on operation in the queue can be a `Grab()` or any other function that can be queued (see the list on page 56).

Several functions are specifically designed to work with the switch value set by the `WaitAnyEvent()` and `WaitAllEvents()` functions:

- `GetSwitch()`—Returns the current value of the switch. You can use the value returned to control the flow of your program.
SwitchGrab()—Performs a Grab() to capture an image, but sends the image to one of four possible frames depending on the value of the switch.

SwitchCamera()—Performs a SetCamera(), selecting one of the four possible the video input sources based on the value of the switch.

The switch value is cleared when the frame grabber is reset by calling the Reset() function.

Controlling the Output Lines

While the initial version of the PXC200 has no output lines, the Frame Grabber library functions support output lines for future expansion.

You can use output lines to send timing signals or other information to an external device. For example, you could use output lines to send a programmed sequence of strobe pulses to a camera or other device.

Writing to the Output Lines

You can set the state of the output lines using the WriteImmediateIO() function. You designate which lines to set, and which state to set for each, by setting a state parameter and a mask parameter in the function call. Any line for which the mask bit is set to 1 will have its state set to the value of the corresponding bit in state. The function will fail if all mask bits are zero.

On boards with latched input lines, you can use the WriteImmediateIO() function to clear the input line after reading the line.

The ReadIO() function returns the current state of all I/O lines.
Error Handling

The `CheckError()` function returns a flag if any of the following errors have occurred:

**Invalid frame grabber handle**—`CheckError()` was called using an invalid handle for the frame grabber.

**Corrupt data**—A captured image was transferred incorrectly and might contain bad data.

**Overflow**—The incoming video signal exceeds the range of the digitizer.

Error flags get cleared every time `CheckError()`, `AllocateFG()`, or `Reset()` are called.

You can use the `Reset()` function to restore the frame grabber to its default state. `Reset()` aborts any operations pending in the queue and the digital I/O.

Reading Frame Grabber Information

**Board Revision Number**

The frame grabber has a revision number encoded in it, which can be read using the `ReadRevision()` function. In most cases you won’t need
this function. If you need your revision number for calling Imagenation Technical Support, use one of these easy methods:

**DOS or DOS/4GW**—Run the PXCREV program.

**Any version of Windows**—Run either of the PXCDRAW sample programs. The revision number appears in the title bar.

### Hardware Protection Key

You can request to have your frame grabbers encoded with a unique protection key that your software can read using the `ReadProtection()` function. Checking for the key in software gives you some protection against software piracy, since you can prevent the software from running on systems that you have not supplied.

### Serial Number

You can request to have your frame grabbers encoded with a serial number, which can be used to identify a specific board. The `ReadSerial()` function returns the encoded serial number, if any.

### Frame Grabbing and PCI Bus Performance

Data transfers can take advantage of the maximum 132 MB per second burst transfer rate of the PCI bus. Although actual throughput is typically well below the maximum burst rate, a properly-designed system can support real-time transfer and display of at least 8-bit-per-pixel video image data. Actual throughput is affected by the PCI implementation on the motherboard, the design of the PCI video controller or other PCI device, and the load on the bus due to all PCI devices using it.
If the PCI bus is overloaded, it’s possible for captured data to be corrupt. Although the Grab functions can’t determine if data is being corrupted, CheckError() will return the value ERR_CORRUPT.

Accessing Captured Image Data

You can access image data stored in a frame in main memory in two ways:

- Use the FrameBuffer() function to get a *logical* address (a pointer) to the data and use the pointer to operate directly on the data. You can use FrameBuffer() only on frames you create with AllocateBuffer(), AllocateFlatFrame(), and AllocateMemoryFrame(); frames you create with AllocateAddress() can’t be read by the library, so you can’t use FrameBuffer() to get a logical address to those frames.

- Use the GetPixel(), GetRectangle(), GetRow(), and GetColumn() functions to copy parts of the image data from a frame to a buffer you have created in memory. Use the PutPixel(), PutRectangle(), PutRow(), and PutColumn() functions to copy parts of the image data a buffer you have created in memory to a frame. For languages, such as Visual Basic, that do not have pointers, these functions are the only way to access the data in a frame buffer. These functions will cause unpredictable results if the buffer you are copying to isn’t large enough to hold the data.

The following functions are also useful in working with frame data:

- **CopyFrame()**—Copies a rectangular region of pixels from one frame to another frame.

- **ExtractPlane()**—Returns a frame containing one of the planes from a frame containing planar data, such as YUV422P or YUV444P.
FrameHeight(), FrameWidth(), and FrameType()—Return, respectively, the height, width, and type of pixel data for the specified frame.

AllocateMemoryFrame()—Can allocate frames for any of the pixel data types, including the floating point types PBITS_Yf and PBITS_RGBf. The memory for the frame is not guaranteed to be in one contiguous block.

AllocateFlatFrame()—Can allocate frames for any of the pixel data types, including the floating point types PBITS_Yf and PBITS_RGBf. The memory is guaranteed to be in one contiguous block.

You can use FrameAddress() to get the physical address for a buffer, but don’t try to use this physical address to access data in an application program; use the logical address returned by FrameBuffer() instead. FrameAddress() is provided only for special situations in which a physical address might be needed, as in writing device drivers.

**Frame and File Input/Output**

The library provides functions for writing and reading image data to and from files. You can read and write unformatted (binary) files and Windows BMP formatted files. Formatted files include information about the image, including the width, height, and number of bits per pixel, while binary files include only the pixel values.

**BMP Files**

The BMP routines ReadBMP() and WriteBMP() read and write frames to image files on disk using the Windows BMP formats. Y8 images are written and read as 8-bit-per-pixel BMP files with a grayscale palette. RGB images are written and read as 24-bit, true-color BMP files. In RGB32, the alpha data is ignored.
If a BMP file is read into a frame that does not have room to store the entire BMP image, the image is clipped on the right and bottom edges. If the BMP file image is smaller than the frame, the image is padded on the right and bottom with zeros.

**Binary Files**

The routines **ReadBin()** and **WriteBin()** read and write unformatted image data to and from files. Unformatted files contain no information on an image’s height, width, or pixel type, so you must keep track of that information. For example, nothing prevents you from saving a frame that is 320 pixels wide and 160 pixels tall in an unformatted file, and then reading that file into a frame that is 160 pixels wide and 320 pixels tall, even though each line of the original frame will occupy two lines in the new frame. If you use unformatted files, keep track of the characteristics of the stored frames.

**Using the Video Display DLL**

The Video Display DLL is a simple tool for displaying video images in a window. Since it is a standard DLL, it can be used with Visual Basic, C, and other languages that can call DLLs. The Video Display DLL supports only one operation: copying an arbitrary rectangle of an image frame onto an arbitrary rectangle of a window’s client area. There are two functions that are needed for this purpose:

```c
void pxSetWindowSize(int x, int y, int dx, int dy) This function specifies the position and size of the rectangle where the image will be drawn, in units of pixels relative to the client area of the window where the drawing takes place. If pxSetWindowSize() is never called, the default values are x = 0, y = 0, dx = 640, and dy = 512.

void pxPaintDisplay(HDC hdc, FRAME __PX_FAR *frh, int x, int y, int dx, int dy) This function takes the rectangular area specified
```
by \( x, y, dx, \) and \( dy \) from the frame \( frh \), stretches it to fit the rectangle set by \( pxSetWindowSize() \), and draws it into the device context \( hdc \), which should be a valid device context for the window in which the image is to appear.

The frame pointer used by \( pxPaintDisplay() \) must reference a valid frame created by a call to the Frame DLL. This means that the library must be initialized properly and a frame must be allocated before the Video Display DLL can be used.

The Video Display DLL doesn’t necessarily use the most efficient techniques to pipe the video information to a window. It is intended to be a tool to make video display as easy as possible, and may not be the best solution if you are concerned primarily with performance.

To incorporate the Video Display DLL into your programs, you will need these files:

<table>
<thead>
<tr>
<th>16-bit Windows Programs</th>
<th>32-bit Windows Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIDEO_16.H</td>
<td>VIDEO_32.H</td>
</tr>
<tr>
<td>VIDEO_16.LIB</td>
<td>VIDEO_32.LIB</td>
</tr>
<tr>
<td>VIDEO_16.DLL</td>
<td>VIDEO_32.DLL</td>
</tr>
<tr>
<td>VIDEO_16.BAS</td>
<td>VIDEO_32.BAS</td>
</tr>
</tbody>
</table>

To link to the DLL, you must include the .BAS files in a Visual Basic program. If you want to use this DLL with a C program, you must put the prototypes of the functions (as they appear on page 66) in your program’s source or header files; these prototypes do not appear in the main header files.
Imagenation
PXC200 Library Reference

The chapter is a complete, alphabetical function reference for the PXC200 Frame Grabber libraries and DLLs. For additional information on using the functions, see Chapter 4, *Programming the PXC200*, on page 25. For reference information on the Frame library, see Chapter 6, *Frame Library Reference*, on page 101.

The 16-bit Windows 3.1 PXC2_31.DLL uses the Pascal calling convention. The 32-bit Windows 95 PXC2_95.DLL uses the _stdcall calling convention.

This function reference is a general guide for using the functions with all operating systems and languages. The functions will work as written for C and Visual Basic with the header files provided.

If you need to construct your own header file, you will need to know the definitions of constants and the sizes of the parameters and the return values for the function calls. You can find the definitions of constants in the
header files for C and Visual BASIC. The following table gives the sizes of the various data types that are used by the PXC200 library.

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>8 bits</td>
</tr>
<tr>
<td>long, unsigned long</td>
<td>32 bits</td>
</tr>
<tr>
<td>void *, unsigned char *, int *, char *, LPSTR</td>
<td>32 bits</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

FRAME and FGLIB are defined types; to see how they are defined, refer to the C language header file for the appropriate operating system. Void is a special type. When it is the type for a parameter, the function has no parameters; when it is the type for the return value, the function does not return a value.

The library and DLL interface is almost identical for all operating systems. Functions that do not apply to a particular operating system or language are noted with an icon:

⚠️ Does not apply to Visual Basic.

### AllocateBuffer()

**Syntax**

```c
FRAME __PX_FAR *AllocateBuffer(short dx, short dy, short type);
```

**Return Value**

A handle to the allocated FRAME structure.

NULL on failure.

**Description**

Reserves memory for an image buffer of size $dx$ by $dy$, with the specified pixel data $type$. For the buffer to be usable by the frame grabber, $dx$ and $dy$ must be at least as large as the image being grabbed. FreeFrame() should be used to release the frame when it is no longer needed.
For more information and a list of pixel data types, see Allocating and Freeing Frames, on page 39.

See Also
FreeFrame()

AllocateFG()

Syntax
long AllocateFG(short n);

Return Value
A handle for the requested frame grabber.
0 on failure.

Description
AllocateFG() attempts to find a frame grabber and give the program access to it. The program can request a specific frame grabber in a system that has more than one by specifying a number, n. Due to the design of the PCI bus, bus slot 0 doesn’t necessarily correspond to frame grabber 0, and the number of the frame grabber in a particular bus slot can vary between different operating systems. You can determine which frame grabber is which by connecting a video source to only one frame grabber and then using the PCXVU program (or your own program) to switch between frame grabbers. To request any available frame grabber, specify n = -1.

If the frame grabber is available, AllocateFG() returns a handle that must be used in other library functions that refer to the frame grabber.

The program should call FreeFG() on the frame grabber when it is no longer needed.

For more information, see Requesting Access to Frame Grabbers, on page 37.

See Also
FreeFG()
Imagenation

CheckError()

Syntax
long CheckError(long fgh);

Return Value
0 if no errors have occurred.
1 if the handle fgh is invalid.
One or more of these flags if an error has occurred:

<table>
<thead>
<tr>
<th>Error Returned</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR_CORRUPT</td>
<td>A captured image was transferred incorrectly and might contain bad data.</td>
</tr>
<tr>
<td>ERR_IO_FAIL</td>
<td>The state of the digital I/O lines does not match the state the software set them to.</td>
</tr>
<tr>
<td>ERR_NOT_VALID</td>
<td>fgh is not a valid frame grabber handle.</td>
</tr>
<tr>
<td>WARN_OVERFLOW</td>
<td>The video signal exceeds the range of the digitizer.</td>
</tr>
</tbody>
</table>

Description
CheckError() queries the frame grabber to determine whether any of a known set of errors occurred. These errors are automatically cleared when CheckError() returns and by successful calls to AllocateFG() and Reset().

CloseLibrary()

DOS Syntax
void PXC200_CloseLibrary(FGLIB __PX_FAR *interface);

Win C Syntax
void imagenation_CloseLibrary(FGLIB __PX_FAR *interface);

Win VB Syntax
CloseLibrary(0)

Return Value
None.

Description
Returns to the system any resources that were allocated by OpenLibrary(). CloseLibrary() should be the last library function called by the program. A program that exits after calling OpenLibrary(), but before calling CloseLibrary(), will leave the computer in an unstable state and might crash the operating system.
For more information, see *Initializing and Exiting Libraries*, on page 34.

**See Also**

- OpenLibrary()

---

### FreeFG()

**Syntax**

```c
void FreeFG(long fgh);
```

**Return Value**

None.

**Description**

Releases control of a frame grabber (previously allocated with the AllocateFG() function) after the program is finished using the frame grabber.

**See Also**

- AllocateFG()

---

### FreeFrame()

**Syntax**

```c
void FreeFrame(FRAME __px_far *f);
```

**Return Value**

None.

**Description**

Returns memory associated with a FRAME handle to the system. You must free all frames allocated by AllocateBuffer() before calling CloseLibrary().

This function is identical to the FreeFrame() function in the Frame library. Either version of the function can free a frame allocated by either library.

**See Also**

- AllocateBuffer()

---

### GetBrightness()

**Syntax**

```c
float GetBrightness(long fgh);
```

**Return Value**

The current brightness setting.

- 0 on failure.

**Description**

Returns the current brightness (monochrome offset) setting for the frame grabber. This function executes concurrently with any queued functions.
Imagenation

If a SetBrightness() function is queued when GetBrightness() is called, either function might execute first, affecting the result returned by GetBrightness().

See Also SetBrightness(), SetContrast()

GetCamera()

Syntax short GetCamera(long fgh);
Return Value The currently active video input.
-1 on failure.
Description Returns the active video input of the specified frame grabber. Use SetCamera() to specify the active video input. If a SetCamera() function is queued when GetCamera() is called, either function might execute first, affecting the result returned by GetCamera().

See Also SetCamera()

GetChromaControl()

Syntax short GetChromaControl(long fgh);
Return Value A set of flags if successful.
-1 on failure.
Description Returns a set of flags for the optional features for processing the color portion of the video signal. The flag values are listed for the function SetChromaControl(), on page 86. For more information, see Setting Chroma Controls, on page 48.

See Also SetChromaControl()

GetContrast()

Syntax float GetContrast(long fgh);
Return Value The current contrast setting.
0 on failure.
**GetContrast()**

Description

Returns the current contrast (monochrome gain) setting for the frame grabber. This function executes concurrently with any queued functions. If a `SetContrast()` function is queued when `GetContrast()` is called, either function might execute first, affecting the result returned by `GetContrast()`.

See Also

`SetBrightness()`, `SetContrast()`

---

**GetHeight()**

**Syntax**

`short GetHeight(long fgh)`

**Return Value**

The currently set height if successful.

0 if `fgh` is invalid.

**Description**

Returns the height in pixels of the cropped image, as set by `SetHeight()`.

The top-most pixel in the cropped image is set with `SetTop()`.

This function waits until the frame grabber queue is empty before executing.

See Also

`SetHeight()`, `SetTop()`

---

**GetHue()**

**Syntax**

`float GetHue(long fgh);`

**Return Value**

The current hue setting if successful.

0 on failure.

**Description**

Returns the current hue setting for the frame grabber. This function executes concurrently with any queued functions. If the `SetHue()` function is queued when `GetHue()` is called, either function might execute first, affecting the result returned by `GetHue()`.

For more information, see [Setting Hue and Saturation](#), on page 45.

See Also

`SetHue()`
GetInterface()

**Syntax**

```c
const void __PX_FAR *GetInterface(long handle);
```

**Return Value**

None.

**Description**

A C macro that returns a pointer to the interface structure for a given frame grabber `handle`. You should assume that the structure pointed to is read-only. It is your responsibility to know what type of object is represented by handle and to cast the return value to the correct type. Be sure the `handle` is valid, since this macro is not good at error detection. This macro is intended for advance users who want to write complicated device-independent code.

**See Also**

OpenLibrary()

GetIOType()

**Syntax**

```c
short GetIOType(long fgh, short n);
```

**Return Value**

Type of I/O line `n` if successful.

-1 on failure.

**Description**

Returns the type of I/O line number `n`, where `n = 0` for the PXC200, and the type is one of the following:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LATCH_RISING</td>
<td>The state of the line will be set to 1 if the signal changes from low to high.</td>
</tr>
<tr>
<td>LATCH_FALLING</td>
<td>The state of the line will be set to 1 if the signal changes from high to low.</td>
</tr>
<tr>
<td>INPUT</td>
<td>The state of the line is equal to the signal value.</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>The line is an output line.</td>
</tr>
</tbody>
</table>

For more information, see *Digital I/O*, on page 57.

**See Also**

SetIOType()
GetLeft()

Syntax       short GetLeft(long fgh)

Return Value The currently set left edge if successful.
              0 if fgh is invalid.

Description Returns the left-most pixel of the cropped image, as set by SetLeft(). The
width of the cropped image is set with SetWidth().

This function waits until the frame grabber queue is empty before execut-
ing.

See Also    SetLeft(), SetWidth()

GetLumaControl()

Syntax       short GetLumaControl(long fgh);

Return Value A set of flags if successful.
              -1 on failure.

Description Returns a set of flags for the optional features for processing the mono-
chrome portion of the video signal. The flag values are listed for the func-
tion SetLumaControl(), on page 90. For more information, see Setting
Luma Controls, on page 47.

See Also    SetLumaControl()

GetSaturation()

Syntax       float GetSaturation(long fgh);

Return Value The current saturation setting if successful.
              0 on failure.

Description Returns the current saturation adjustment. This function executes concur-
rently with any queued functions. If the SetSaturation() function is
queued when GetSaturation() is called, either function might execute
first, affecting the result returned by GetSaturation(). For more information, see *Setting Hue and Saturation*, on page 45.

**See Also**

SetSaturation()

---

### GetSwitch()

**Syntax**

```c
short GetSwitch(long fgh);
```

**Return Value**

- The number of the I/O line.
- 0 if neither of the Wait functions has completed.
- -1 on failure.

**Description**

When a WaitAllEvents() or WaitAnyEvent() function completes, the function sets the switch value to the number of the highest I/O line that satisfied the wait condition. GetSwitch() returns the line number, or zero if the function hasn’t yet completed. For more information, see *Controlling the Input Lines*, on page 58.

This function executes concurrently with any queued functions. If another WaitAllEvents() or WaitAnyEvent() function is queued when GetSwitch() is called, either function might execute first, affecting the result returned by GetSwitch().

**See Also**

WaitAllEvents(), WaitAnyEvent()
GetVideoDetect()

Syntax  
short GetVideoDetect(long fgh);

Return Value  
The currently-set video type if successful.
-1 on failure.

Description  
Returns the video type expected by the frame grabber, as set by SetVideoDetect().

See Also  
SetVideoDetect()

GetVideoLevel()

Syntax  
float GetVideoLevel(long fgh);

Return Value  
The current video level if successful.
0 on failure.

Description  
Returns the voltage difference between the bottom of video sync and bright white, as set by SetVideoLevel(). For more information, see Setting the Video Level, on page 46.

See Also  
SetVideoLevel()

GetWidth()

Syntax  
short GetWidth(long fgh)

Return Value  
The currently set width if successful.
0 if fgh is invalid.

Description  
Returns the width in pixels of the cropped image, as set by SetWidth().
The left-most pixel in the cropped image is set with SetLeft().
This function waits until the frame grabber queue is empty before executing.

See Also  
SetLeft(), SetWidth()
GetXResolution()

Syntax: short GetXResolution(long fgh)

Return Value: The current X resolution if successful.
0 if fgh is invalid.

Description: Returns the number of pixels the frame grabber will digitize per row of video, as set by SetXResolution(). The captured image might be fewer pixels in width if the image has been cropped with SetLeft() and SetWidth().

See Also: SetLeft(), SetWidth(), SetXResolution()

GetYResolution()

Syntax: short GetYResolution(long fgh)

Return Value: The current Y resolution if successful.
0 if fgh is invalid.

Description: Returns the number of pixels the frame grabber will digitize vertically per frame of video, as set by SetYResolution(). The captured image might be fewer pixels in height if the image has been cropped with SetTop() and SetHeight().

See Also: SetHeight(), SetTop(), SetYResolution()

Grab()

Syntax: long Grab(long fgh, FRAME __PX_FAR *frh, short flags)

Return Value: A queued operation handle if successful.
0 on failure.

Description: Captures a video image and writes it to frame buffer frh. Grab() fails if the image size is larger in either the horizontal or vertical dimension than the destination frame. For more information, see Grabbing Images, on page 42.
The parameter *flags* is a set of flag bits that can specify modes of operation for this function. If *flags* is 0, the default modes will be used. See *Using Flags with Function Calls*, on page 57.

**See Also**  
AllocateFG(), AllocateBuffer(), GrabContinuous(), SwitchGrab()

### GrabContinuous()

**Syntax**  
short GrabContinuous(long fgh, FRAME __PX_FAR *frh, short state, short flags);

**Return Value**  
Non-zero if successful.  
0 on failure.

**Description**  
Turns continuous acquire mode on (if *state* = -1) or off (if *state* = 0) for a given frame grabber. In continuous acquire mode, the buffer *frh* is continuously updated with new video data. GrabContinuous() fails if the frame is not of the correct type to hold the data.

Continuous acquire mode can be useful for software that is watching a small number of pixels in every image, or for sending video data directly to another PCI device, but also requires fast access to RAM. Using continuous acquire mode while other memory accesses or PCI accesses are occurring might require more data to be transferred than is possible on some computers, resulting in corrupt video data. The Grab functions can’t determine when data corruption is occurring, but CheckError() will return ERR_CORRUPT.

The Grab() and SwitchGrab() functions and any operations that change the type of data produced by the frame grabber or the resolution or size of the video image automatically turn off continuous acquire mode.

For more information, see *Grabbing Images*, on page 42.

The parameter *flags* can specify additional modes of operation for this function. If *flags* is 0, the default modes will be used. See *Using Flags with Function Calls*, on page 57.

**See Also**  
Grab(), SwitchGrab()
Imagenation

IsFinished()

Syntax  short IsFinished(long fgh, int handle);
Return Value  >0 if the operation is not in the queue.
               0 if the specified operation is in the queue and has not completed.
               -1 if the specified frame grabber is invalid.
Description  Can be used to check whether a queued operation has finished by passing
              the handle returned by the function that queued the operation. It can also
              check whether all operations queued for a particular frame grabber are
              finished by using handle = 0. For more information on queued functions,
              see Timing the Execution of Functions, on page 51.

Many frame grabber control functions can queue operations if they are
passed the appropriate flags. For more information, see Using Flags with
Function Calls, on page 57.

See Also  WaitFinished()

KillQueue()

Syntax  void KillQueue(long fgh);
Return Value  None.
Description  Aborts any operations in progress for the specified frame grabber. Any
operations in the queue when this function is called will be removed,
although the operations might already have executed. For instance, if a
Grab() command was in the queue, some or all of the video data might
have been written into the frame by the time the queue is killed.

This function takes several milliseconds to execute. It is intended prima-
arily for recovering from error conditions.

See Also  Reset()
OpenLibrary()

**DOS Syntax**
short PXC200_OpenLibrary(FGLIB __PX_FAR *interface, short sizeof(interface));

**Win C Syntax**
short imagenation_OpenLibrary(LPSTR dllname, __PX_FAR *interface, short sizeof(interface));

**Win VB Syntax**
integer OpenLibrary(0,0)

**Return Value**
Number of available frame grabbers. 0 on failure.

**Description**
Initializes library data structures and locates all available frame grabbers. It must be called successfully before any other library functions can be used.

OpenLibrary() will usually fail only if no frame grabbers are detected, but may also fail under conditions of extremely low memory. When OpenLibrary() fails, use CheckError() to get the error.

For more information on using OpenLibrary(), see *Initializing and Exiting Libraries*, on page 34.

See Also
CloseLibrary()

ReadIO()

**Syntax**
unsigned long ReadIO(long fgh);

**Return Value**
The state of the I/O lines if successful. 0 on failure.

**Description**
Returns a set of bit flags indicating the state of the I/O lines. Bits 0 - 7 correspond to I/O lines 0 - 7. Bits that have no associated I/O line return zero.

See Also
SetIOType(), WriteImmediateIO()
ReadProtection()

Syntax
short ReadProtection(long fgh);

Return Value
The protection key if successful.
0 on failure.

Description
Returns the hardware protection key of the frame grabber. The returned value will be zero unless the frame grabber has been programmed with a key to match your custom software.

ReadRevision()

Syntax
short ReadRevision(long fgh);

Return Value
The revision number if successful.
0 on failure.

Description
Returns the hardware/firmware revision number of the frame grabber. If $fgh = 0$, ReadRevision() returns the revision number of the software library.

You can also get the revision number using the PXCREV utility program in DOS or any of the sample programs in Windows; the sample programs display the revision number in the title bar.

ReadSerial()

Syntax
long ReadSerial(long fgh);

Return Value
The serial number of the board if successful.
0 on failure.

Description
Returns the serial number of the frame grabber. The value returned will be zero unless the frame grabber has been programmed with a serial number.
Reset()

Syntax
void Reset(long fgh);

Return Value
None.

Description
Returns the frame grabber to a default state, and aborts any queued operations and any digital I/O operations. This function takes several milliseconds to execute.

See Also
KillQueue()

SetBrightness()

Syntax
long SetBrightness(long fgh, float offset, short flags);

Return Value
A queued operation handle if successful.
0 on failure.

Description
Sets the offset value for the monochrome signal, where -0.5 \leq offset \leq +0.5. A value of +0.5 increases the digitized value of black to medium gray, and a value of -0.5 brings the digitized value of white to medium gray. For more information, see Setting Contrast and Brightness, on page 45.

The parameter flags is a set of flag bits that can specify modes of operation for this function. If flags is 0, the default modes will be used. See Using Flags with Function Calls, on page 57.

See Also
GetBrightness(), SetContrast()

SetCamera()

Syntax
short SetCamera(long fgh, short n, short flags);

Return Value
A queued operation handle if successful.
0 on failure.
Imagenation

**Description**

Selects one of the video inputs (0-3) on the frame grabber to be active. The camera attached to the selected input is the source for all subsequent video input to the frame grabber.

The parameter `flags` is a set of flag bits that can specify modes of operation for this function. If `flags` is 0, the default modes will be used. See *Using Flags with Function Calls*, on page 57.

**See Also**

GetCamera()

---

**SetChromaControl()**

**Syntax**

`short SetChromaControl(long fgh, short cf);`

**Return Value**

Non-zero if successful.

0 on success.

**Description**

Selects features for processing the color portion of the video signal. The parameter `cf` is a set of flags that can be combined with the OR operator to select specific features:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVIDEO</td>
<td>Color information is digitized from the separate chroma channel of the S-Video input. If this flag is not set, color information is extracted from the composite video signal. On the PXC200, this flag only affects video input 1, which is the only video input that supports the S-Video format.</td>
</tr>
<tr>
<td>NOTCH_FILTER</td>
<td>Activates an analog filter to remove the color burst signal from the luminance channel before brightness information is digitized.</td>
</tr>
<tr>
<td>AGC</td>
<td>Activates the automatic gain control for color saturation. If this flag is enabled, the board attempts to compensate for non-standard color burst amplitudes.</td>
</tr>
</tbody>
</table>
Chapter 5  PXC200 Library Reference

For more information, see Setting Chroma Controls, on page 48.

This function waits for the queue to empty before executing.

**See Also**

GetChromaControl()

---

### SetContrast()

**Syntax**

long SetContrast(long fgh, float gain, short flags);

**Return Value**

A queued operation handle if successful.

0 on failure.

**Description**

Sets the monochrome gain for the frame grabber, where $0.0 \leq \text{gain} \leq 2.0$. The amplitude of the input signal is multiplied by the gain, so the contrast of the input signal is unchanged for $\text{gain} = 1$. For more information, see Setting Contrast and Brightness, on page 45.

The parameter flags is a set of flag bits that can specify modes of operation for this function. If flags is 0, the default modes will be used. See Using Flags with Function Calls, on page 57.

**See Also**

GetContrast(), SetBrightness()

---

### SetHeight()

**Syntax**

short SetHeight(long fgh, short dy)

**Return Value**

The actual height set if successful.

0 if fgh is invalid.
Imagenation

**Description**  The height in pixels of the cropped image. The top-most pixel in the cropped image is set with SetTop(). The frame grabber sets the height to the closest value less than or equal to $dy$ it is capable of and returns the actual value set. For more information, see *Scaling and Cropping Images*, on page 49.

This function waits until the frame grabber queue is empty before executing.

**See Also**  SetLeft(), SetTop(), SetWidth()

---

**SetHue()**

**Syntax**  
```
long SetHue(long fgh, float h, short flags);
```

**Return Value**  A queued operation handle if successful. 0 on failure.

**Description**  Sets the hue adjustment to $h$, or the closest value the frame grabber is capable of, where $-90 \leq h \leq +90$. SetHue() is ignored for PAL/SECAM video signals.

For more information, see *Setting Hue and Saturation*, on page 45.

**See Also**  GetHue(), SetSaturation()

---

**SetIOType()**

**Syntax**  
```
short SetIOType(long fgh, short n, short type);
```

**Return Value**  Non-zero if successful. 0 on failure.
Sets the type of I/O line number $n$, where $0 \leq n \leq 3$ and the type is one of the following:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LATCH_RISING</td>
<td>The state of the line will be set to 1 if the signal changes from low to high.</td>
</tr>
<tr>
<td>LATCH_FALLING</td>
<td>The state of the line will be set to 1 if the signal changes from high to low.</td>
</tr>
<tr>
<td>INPUT</td>
<td>The state of the line is equal to the signal value. This is the default.</td>
</tr>
</tbody>
</table>

SetIOType() executes only after all functions in the queue have completed. For more information, see Digital I/O, on page 57.

See Also

GetIOType()
## SetLumaControl()

**Syntax**

```c
short SetLumaControl(long fgh, short lf);
```

**Return Value**

Non-zero if successful.
0 on success.

**Description**

Selects features for processing the monochrome portion of the video signal. The parameter `lf` is a set of flags that can be combined with the OR operator to select specific features:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOW_FILTER_AUTO</td>
<td>Activates a low-pass filter that reduces high-frequency information in the video.</td>
</tr>
<tr>
<td>LOW_FILTER_1</td>
<td>LOW_FILTER_3 has the highest level of filtering. LOW_FILTER_AUTO automatically sets the filtering based on the resolution. Set, at most, one of these flags, or omit all for no filtering.</td>
</tr>
<tr>
<td>LOW_FILTER_2</td>
<td></td>
</tr>
<tr>
<td>LOW_FILTER_3</td>
<td></td>
</tr>
<tr>
<td>CORE_8</td>
<td>Forces any video with a brightness value less than n/256 (where n is 8, 16, or 32) to be digitized as black. Set, at most, one of these flags, or omit all for no coring.</td>
</tr>
<tr>
<td>CORE_16</td>
<td></td>
</tr>
<tr>
<td>CORE_32</td>
<td></td>
</tr>
<tr>
<td>GAMMA_CORRECT</td>
<td>Activates a filter to gamma correct RGB video. For NTSC, gamma = 2.2; for PAL/SECAM, gamma = 2.8. YCrCb video is never gamma corrected.</td>
</tr>
<tr>
<td>COMB_FILTER</td>
<td>Activates digital filtering of the brightness data to reduce artifacts.</td>
</tr>
<tr>
<td>PEAK_FILT_0</td>
<td>Activates a filter that amplifies high-frequency information in the video. PEAK_FILT_0 has the highest gain. These filters will sharpen edges in a blurry video image, but might cause artifacts on edges that are already sharp. Set, at most, one of these flags, or omit all for no filtering.</td>
</tr>
<tr>
<td>PEAK_FILT_1</td>
<td></td>
</tr>
<tr>
<td>PEAK_FILT_2</td>
<td></td>
</tr>
<tr>
<td>PEAK_FILT_3</td>
<td></td>
</tr>
</tbody>
</table>
For more information, see *Setting Luma Controls*, on page 47.

This function waits for the queue to empty before executing.

See Also  
*GetLumaControl()*

### SetPixelFormat()

**Syntax**  
short SetPixelFormat(long fgh, short type);

**Return Value**  
Non-zero if successful.  
0 on failure.

**Description**  
Sets the pixel format that the frame grabber expects to digitize. Pixel types are listed in the table on page 40.

The frame grabber automatically changes to the correct format for the destination frame when a Grab(), GrabContinuous(), or SwitchGrab() function is executed, so using SetPixelFormat() explicitly is often not necessary. The frame grabber requires one field time of delay before it can digitize in a new format, whether the format change occurs due to calling SetPixelFormat() or due to the frame type for a Grab function. When the change occurs because of a Grab, this delay becomes part of the latency for the Grab. Using SetPixelFormat() to preset the expected pixel format minimizes the latency in the Grab function. For more information, see *Allocating and Freeing Frames*, on page 39.

This function waits for the queue to empty before executing.

### SetSaturation()

**Syntax**  
long SetSaturation(long fgh, float s, short flags);

**Return Value**  
A queued operation handle if successful.  
0 on failure.
Imagenation

Description
Sets the saturation adjustment to $s$, or the closest value the frame grabber is capable of, where $0.0 \leq s \leq 2.0$. For more information, see Setting Hue and Saturation, on page 45.

See Also
GetSaturation(), SetHue()  

SetTop()

Syntax
short SetTop(long fgh, short y0)

Return Value
The actual pixel position set if successful.
0 if fgh is invalid.

Description
The top-most pixel of the cropped image. The height of the cropped image is set with SetHeight(). The frame grabber sets the top-most pixel to the closest value to $y0$ it is capable of and returns the actual value set. For more information, see Scaling and Cropping Images, on page 49.

This function waits until the frame grabber queue is empty before executing.

See Also
SetHeight(), SetLeft(), SetWidth()  

SetVideoDetect()

Syntax
short SetVideoDetect(long fgh, short type);

Return Value
Non-zero if successful.
0 on failure.

Description
Sets the video format the frame grabber should expect to $type$. Calling this function may cause the X resolution and Y resolution to change if the frame grabber can't support the current resolution in the new video format. Possible values for $type$ are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTO_FORMAT</td>
<td>The frame grabber will measure the field length and adjust to either NTSC or PAL video. Detecting a format change will take about 2.5 seconds.</td>
</tr>
</tbody>
</table>
Chapter 5   PXC200 Library Reference

For more information, see Selecting Video Inputs, on page 44.

This function waits for the video queue to empty before executing.

See Also

VideoType()

SetVideoLevel()

Syntax

float SetVideoLevel(long fgh, float white);

Return Value

The video level actually set if successful.
0 on failure.

Description

Sets the expected voltage difference between the bottom of video sync and bright white, where $0.0 \leq \text{white} \leq 2.5$. The nominal level is 1.3 V. The function sets the video level to the closest value the frame grabber is capable of and returns the value actually set. For more information, see Setting the Video Level, on page 46.

This function waits for the queue to empty before executing.

See Also

GetVideoLevel()
SetWidth()

**Syntax**
short SetWidth(long fgh, short dx)

**Return Value**
The actual width set if successful.
0 if fgh is invalid.

**Description**
The width in pixels of the cropped image. The left-most pixel in the cropped image is set with SetLeft(). The frame grabber sets the width to the closest value less than or equal to \( dx \) it is capable of and returns the actual value set. For more information, see *Scaling and Cropping Images*, on page 49.

This function waits until the frame grabber queue is empty before executing.

**See Also**
SetHeight(), SetLeft(), SetTop()

SetXResolution()

**Syntax**
short SetXResolution(long fgh, short rez)

**Return Value**
The actual resolution set if successful.
0 if fgh is invalid.

**Description**
Sets the number of pixels the frame grabber will digitize per row of video. The frame grabber sets the resolution to the closest value to \( rez \) it is capable of and returns the actual value set. For more information, see *Scaling and Cropping Images*, on page 49.

This function waits until the frame grabber queue is empty before executing.

**See Also**
SetLeft(), SetWidth(), SetYResolution()
**SetYResolution()**

**Syntax**

short SetYResolution(long fgh, short rez)

**Return Value**

The actual resolution set if successful.

0 if fgh is invalid.

**Description**

Sets the number of pixels the frame grabber will digitize vertically per frame of video. The frame grabber sets the resolution to the closest value to `rez` it is capable of and returns the actual value set. For more information, see *Scaling and Cropping Images*, on page 49.

This function waits until the frame grabber queue is empty before executing.

**See Also**

`SetHeight()`, `SetTop()`, `SetXResolution()`

**SwitchCamera()**

**Syntax**

long SwitchCamera(long fgh, short flags);

**Return Value**

A queued operation handle if successful.

0 on failure.

**Description**

Sets the active video input to the switch value set by the last complete `WaitAnyEvent()` or `WaitAllEvents()` function. If the value of the switch is larger than the number of valid video inputs, the function does nothing. For more information, see *Controlling the Input Lines*, on page 58.

**See Also**

`WaitAllEvents()`, `WaitAnyEvent()`

**SwitchGrab()**

**Syntax**

long SwitchGrab(long fgh, FRAME __PX_FAR *f0, FRAME __PX_FAR *f1, FRAME __PX_FAR *f2, FRAME __PX_FAR *f3, short flags);

**Return Value**

A queued operation handle if successful.

0 on failure.
Imagenation

**Description**  
This function behaves just like Grab(), except that the image data is written to one of four frames depending on the last WaitAnyEvent() or WaitAllEvents() function that completed. Some, but not all, of the frame pointers can be NULL; if a NULL frame pointer is selected, the function completes, but does nothing. For more information, see *Controlling the Input Lines*, on page 58.

This function fails if all frame pointers are NULL or if any of the frames don’t have the correct width and height.

**See Also**  
*WaitAllEvents(), WaitAnyEvent()*

---

**VideoType()**

**Syntax**  
short VideoType(long fgh);

**Return Value**  
- 0  No video.
- 1  NTSC video.
- 2  PAL/SECAM video.
- 3  Other.
- -1 Invalid fgh.

**Description**  
Returns the type of video signal connected to the frame grabber: NTSC format, PAL/SECAM format, or other. The video source is determined by counting the number of lines per video frame. When the video line count doesn’t match either NTSC or PAL/SECAM, or the frame grabber is not auto-detecting, the function returns Other.

**See Also**  
*SetVideoDetect()*

---

**Wait()**

**Syntax**  
long Wait(long fgh, short flags);

**Return Value**  
A queued operation handle if successful.  
0 on failure.

**Description**  
Waits for the end of the next field, the end of the next frame (two complete fields), or the end of a specific field, depending on the flags you
specify. The default behavior when flags = 0 is to wait for two complete fields.

If the Wait() function is QUEUED, it does not pause program execution, but any QUEUED functions that are called immediately afterwards will not execute until the Wait() is finished.

A useful rule for understanding the Wait() function is that it always has the same timing as a Grab() function called with the same flags; that is, a Wait() takes the same time to execute as the equivalent Grab() function, but doesn’t collect any image data during that time.

The parameter flags is a set of flag bits that can specify modes of operation for this function. If flags is 0, the default modes will be used. See Using Flags with Function Calls, on page 57.

See Also  
WaitVB()

*WaitAllEvents()*

**Syntax**

```c
long WaitAllEvents(long fgh, long ioh, unsigned long mask, unsigned long state, short flags);
```

**Return Value**

A queued operation handle if successful.  
0 on failure.

**Description**

Pauses processing of the queue until an I/O event occurs.  
*WaitAllEvents()* examines the I/O lines as if by the expression  
((ReadIO(ioh) ^ !state) & mask). While the expression is not equal to mask, the queue is paused. If the expression is equal to mask, the state for the highest I/O line that was set is cleared, the switch is set to that I/O line number, and processing of the queue resumes. For more information, see  
Controlling the Input Lines, on page 58.

This function will fail when mask = 0 or when mask has any bits set that represent invalid I/O lines or lines that are output-only.
Imagenation

The parameter flags is a set of flag bits that can specify modes of operation for this function. If flags is 0, the default modes will be used. See Using Flags with Function Calls, on page 57.

See Also GetSwitch(), SetIOType(), SwitchCamera(), SwitchGrab(), WaitAnyEvent()

WaitAnyEvent()

Syntax
long WaitAnyEvent(long fgh, long ioh, unsigned long mask, unsigned long state, short flags);

Return Value
A queued operation handle if successful.
0 on failure.

Description
Pauses processing of the queue until an I/O event occurs. WaitAnyEvent() examines the I/O lines as if by the expression (ReadIO(ioh) ^ !state) & mask. While the expression is zero, the queue is paused. If the expression is non-zero, the state for the highest I/O line that was set is cleared, the switch is set to that I/O line number, and processing of the queue resumes. For more information, see Controlling the Input Lines, on page 58.

This function will fail when mask = 0 or when mask has any bits set that represent invalid I/O lines or lines that are output-only.

The parameter flags is a set of flag bits that can specify modes of operation for this function. If flags is 0, the default modes will be used. See Using Flags with Function Calls, on page 57.

See Also GetSwitch(), SetIOType(), SwitchCamera(), SwitchGrab(), WaitAllEvents()

WaitFinished()

Syntax
void WaitFinished(long fgh, long handle);

Return Value
1 if successful.
0 on failure.
Chapter 5  PXC200 Library Reference

Description  Releases the processor to execute other tasks until a specific operation in the queue has finished. You identify an operation in the queue by the handle returned by the function that queued the operation. For handle = 0, WaitFinished() waits until all operations in the queue have finished. For more information, see Programming in a Multithreaded, Multitasking Environment, on page 29.

See Also  IsFinished()

WaitVB()

Syntax  short WaitVB(long fgh);

Return Value  Non-zero if successful.
0 on failure.

Description  Waits until the next vertical blank. WaitVB() returns when the interrupt routine has completed; this is usually close to the beginning of vertical blank, but can be at any time during vertical blank depending on system loading. WaitVB() returns too late for frame grabbing functions called immediately afterward to capture the field that has just begun.

See Also  Wait()

WriteImmediateIO()

Syntax  short WriteImmediateIO(long fgh, unsigned long mask, unsigned long state);

Return Value  Non-zero on success.
0 on failure.

Description  Sets all I/O lines that have a 1 bit in the mask to the value in the associated bit of state. Lines with a zero bit in the mask are not affected. The function fails without doing anything if the mask has no 1 bits.

On boards with latched input lines, you can use the WriteImmediateIO() function to clear the input line after reading the line.

See Also  ReadIO()
Imagenation
The chapter is a complete, alphabetical function reference for the Frame libraries and DLLs. For additional information on using the functions, see Chapter 4, Programming the PXC200, on page 25. For reference information on the PXC200 Frame Grabber library, see Chapter 5, PXC200 Library Reference, on page 69.

The 16-bit Windows 3.1 FRAME_31.DLL uses the Pascal calling convention. The 32-bit Windows 95 FRAME_95.DLL uses the _stdcall calling convention.

This function reference is a general guide for using the functions with all operating systems and languages. The functions will work as written for C and Visual Basic with the header files provided.

If you need to construct your own header file, you will need to know the definitions of constants and the sizes of the parameters and the return values for the function calls. You can find the definitions of constants in the
header files for C and Visual BASIC. The following table gives the sizes of the various data types that are used by the PXC200 library.

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>8 bits</td>
</tr>
<tr>
<td>long, unsigned long</td>
<td>32 bits</td>
</tr>
<tr>
<td>void *, unsigned char *,  int *, char *, LPSTR</td>
<td>32 bits</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

FRAME and FRAMELIB are defined types; to see how they are defined, refer to the C language header file for the appropriate operating system. Void is a special type. When it is the type for a parameter, the function has no parameters; when it is the type for the return value, the function does not return a value.

The library and DLL interface is almost identical for all operating systems. Functions that do not apply to a particular operating system or language are noted with an icon:

![Does not apply to Visual Basic.](image)

---

**AliasFrame()**

**Syntax**

```c
FRAME __PX_FAR *AliasFrame(FRAME __PX_FAR *f, short x0, short y0, short dx, short dy, unsigned short type);
```

**Return Value**

A pointer to the frame structure. NULL on failure.

**Description**

Creates a new frame structure that uses the memory from the original frame’s image buffer, starting at the location of the pixel x0, y0. The frame f must not be a paged frame buffer and must not be a planar data type. The new frame treats the memory from the old frame as if it has the new data format type.
AliasFrame() fails if the memory required for the new frame does not fit completely inside the old frame. Freeing the old frame before freeing the alias frame can cause undefined behavior, since this frees the image buffer for the alias frame as well. Freeing the alias frame does not affect the original frame’s buffer.

**AllocateAddress()**

**Syntax**

```
FRAME __PX_FAR *AllocateAddress(unsigned long address, short dx, short dy, unsigned short type);
```

**Return Value**

A pointer to the frame structure.

NULL on failure.

**Description**

Creates a frame of size $dx$ by $dy$, with the specified pixel type, from the memory at the specified physical address. Both $dx$ by $dy$ must be greater than zero. AllocateAddress() can allocate any of the types listed on page 40, except the planar types. This function does not attempt to exclusively allocate the physical address space or to verify that writable memory actually exists there.

This function lets you program specialized operations, like peer-to-peer transfers between the frame grabber and another PCI device. It should not be used with linear addresses unless you know the processor’s paging mode is disabled.

FreeFrame() should be called when the frame is no longer needed. This will de-allocate memory associated with the FRAME structure, but will not attempt to free any resources associated with the given buffer address.

**See Also**

FreeFrame()

**AllocateFlatFrame()**

**Syntax**

```
FRAME __PX_FAR *AllocateFlatFrame(short dx, short dy, unsigned short type);
```

**Return Value**

A pointer to the frame structure.

NULL on failure.
**Description**

Creates a frame of size $dx$ by $dy$, with the specified pixel type, from unpaged, contiguous physical memory. Both $dx$ by $dy$ must be greater than zero. The start of the image buffer will be aligned to at least a 32-bit boundary in the program’s address space. AllocateFlatFrame() can allocate any of the types listed on page 40, including the planar types. For planar types, the memory for each plane will be contiguous, but the three planes won’t necessarily be in one contiguous block. Also, the frame structure itself is not necessarily in contiguous memory, only the image buffer.

AllocateFlatFrame() can fail if the system is not configured to allow contiguous buffers. The PXC200 doesn’t need flat frames; this function is provided for compatibility with other products.

For more information and a list of pixel types, see *Allocating and Freeing Frames*, on page 39.

FreeFrame() should be called when the frame is no longer needed.

**See Also**

AllocateMemoryFrame(), FreeFrame()

---

**AllocateMemoryFrame()**

**Syntax**

```c
FRAME __PX_FAR *AllocateMemoryFrame(short dx, short dy,
unsigned short type);
```

**Return Value**

A pointer to the frame structure.

NULL on failure.

**Description**

Creates a frame of size $dx$ by $dy$, with the specified pixel type, from the program’s memory heap. Both $dx$ by $dy$ must be greater than zero. The start of the image buffer will be aligned to at least a 32-bit boundary in the program’s address space. AllocateMemoryFrame() can allocate any of the types listed on page 40.

For more information and a list of pixel types, see *Allocating and Freeing Frames*, on page 39.
FreeFrame() should be called when the frame is no longer needed.

**See Also**

*AllocateFlatFrame(), FreeFrame*

---

### CloseLibrary()

**DOS Syntax**

```c
void FRAME_CloseLibrary(FRAMELIB __PX_FAR *interface);
```

**Win C Syntax**

```c
void imagenation_CloseLibrary(FRAMELIB __PX_FAR *interface);
```

**Win VB Syntax**

```vb
CloseLibrary(0)
```

**Return Value**

None.

**Description**

Returns to the system any resources that were allocated by OpenLibrary(). CloseLibrary() should be the last library function called by the program. A program that exits after calling OpenLibrary(), but before calling CloseLibrary(), will leave the computer in an unstable state and might crash the operating system.

For more information, see *Initializing and Exiting Libraries*, on page 34.

**See Also**

*OpenLibrary()*

---

### CopyFrame()

**Syntax**

```c
short CopyFrame(FRAME __PX_FAR *source, short sourcex, short sourcey, FRAME __PX_FAR *dest, short destx, short desty, short dx, short dy);
```

**Return Value**

Non-zero if successful.

0 on failure.

**Description**

Copies a rectangle of size $dx$ by $dy$ from the frame $source$ to the frame $dest$. Copies data only between parts of rectangles that are within the boundaries of the frames. CopyFrame() fails if the specified region is entirely outside the boundaries of the frames, if the frames can’t be read or written, if the frames are planar, or if the frame don’t have the same pixel data type. For more information, see *Accessing Captured Image Data*, on page 64.
Imagenation

ExtractPlane()

Syntax

FRAME __PX_FAR *ExtractPlane(FRAME __PX_FAR *f, short plane);

Return Value

Description

Returns a frame that contains a single plane of the planar frame \( f \). Returns NULL if \( f \) is not planar. The frame returned contains Y8 data for all the planar types generated by the Frame library. The returned frame has a width and height less than or equal to that of the source frame.

For YUV planar formats, plane 0 is the Y component, plane 1 is the Cr component, and plane 2 is the Cb component. In YUV422P format, plane 0 is the same width and height as the source frame, while both planes 1 and 2 are the height of the source frame by half the width (rounded up).

The frame returned by ExtractPlane() does not need to be freed by FreeFrame(), and calling FreeFrame() on a frame with a single plane will cause the function to return without doing anything. All planes extracted from a frame immediately become invalid when the original frame is freed.

For more information, see Accessing Captured Image Data, on page 64.

FrameAddress()

Syntax

unsigned long FrameAddress(FRAME __PX_FAR *f);

Return Value

The physical address of the frame’s image buffer.

0 on failure.

Description

Returns the physical address of the specified frame’s image buffer. If the frame’s image buffer doesn’t have a fixed physical address, the function fails.

The physical address can not, in general, be converted to a C-style pointer because of segmentation and paging of the processor’s address space. In order to get a logical address (a pointer) to this buffer, use FrameBuffer().
This function is useful for writing low-level code, such as device drivers or memory managers, that need to interact with the frame grabber libraries.

See Also  
FrameBuffer()

FrameBuffer()

Syntax  
void __PX_HW *FrameBuffer(FRAME __PX_FAR *f);

Return Value  
The logical address of the frame’s image buffer.  
0 if the frame handle is invalid.

Description  
Returns a pointer to the start of the data buffer of the specified frame, or NULL if the data is not in the program's address space. An application can use this pointer to access a frame’s image data.

See Also  
FrameAddress()

FrameHeight()

Syntax  
short FrameHeight(FRAME __PX_FAR *f);

Return Value  
The height of the frame in pixels.  
0 if the frame handle is invalid.

Description  
Returns the height of a frame created with any of the Allocate functions.  
For more information, see Accessing Captured Image Data, on page 64.

See Also  
FrameWidth()

FrameWidth()

Syntax  
short FrameWidth(FRAME __PX_FAR *f);

Return Value  
The width of the frame in pixels.  
0 if the frame handle is invalid.
Imagenation

Description
Returns the width of a frame created with any of the Allocate functions.
For more information, see Accessing Captured Image Data, on page 64.

See Also
FrameHeight()

FrameType()

Syntax
short FrameType(FRAME __PX_FAR *f);

Return Value
The pixel data type of the frame.
0 if the frame handle is invalid.

Description
Returns the pixel data type of the frame created with any of the Allocate
functions. For more information and a list of the pixel data types, see
Allocating and Freeing Frames, on page 39 and Accessing Captured
Image Data, on page 64.

See Also
FrameHeight(), FrameWidth()

FreeFrame()

Syntax
void FreeFrame(FRAME __PX_FAR *f);

Return Value
None.

Description
Returns memory associated with a FRAME handle to the system. You
must free all frames allocated by AllocateAddress(),
AllocateFlatFrame(), and AllocateMemoryFrame() before calling
CloseLibrary()

This function is identical to the FreeFrame() function in the PXC200
Frame Grabber library. Either version of the function can free a frame
allocated by either library.

For more information and a list of the pixel data types, see Allocating and
Freeing Frames, on page 39 and Accessing Captured Image Data, on
page 64.

See Also
AllocateAddress(), AllocateFlatFrame(), AllocateMemoryFrame()
GetColumn()

Syntax
short GetColumn(FRAME __PX_FAR *f, void __PX_HUGE *buf, short column);

Return Value
Non-zero if successful.
0 on failure.

Description
Copies a column of the image stored in frame f into the buffer buf. The columns are numbered starting with 0 at the left of the frame. The buffer is assumed to be an array of the correct type to hold the column of pixels. If the entire column won’t fit in the memory pointed to by buf undefined behavior and data corruption might result.

GetColumn() will fail if the specified column is outside the boundaries of the frame, if the frame can’t be read, or if the frame contains planar data.

See Also
GetRow(), PutColumn(), PutRow()

GetPixel()

Syntax
short GetPixel(FRAME __PX_FAR *f, void __PX_HUGE *pixel, short x, short y);

Return Value
Non-zero if successful.
0 on failure.

Description
Copies the pixel at (x,y) into pixel, where (0,0) is the upper-left corner of the frame. The parameter pixel is assumed to point to a variable or structure of the correct type to hold the pixel. If pixel doesn’t point to an object of sufficient size to hold the pixel, undefined behavior and data corruption might result. If the frame is planar, pixel must point to an object that can hold one pixel from each plane, appended in order (example: YUV422P frames require a byte of brightness, followed by a byte of red, followed by a byte of blue, for a total of 24 bits).

If the point specified by (x,y) is outside the boundaries of the frame, or the frame can’t be read, the function call fails.

See Also
PutPixel()
Imagenation

GetRectangle()

Syntax
short GetRectangle(FRAME __PX_FAR *f, void __PX_HUGE *buf, short x0, short y0, short dx, short dy);

Return Value
Non-zero if successful.
0 on failure.

Description
Copies a rectangular region of the frame \( f \) into the buffer \( buf \). The rectangle has upper left corner \((x_0, y_0)\) in the source frame, width \( dx \), and height \( dy \). The buffer is assumed to be an array of the correct type to hold the row of pixels. If the entire rectangle won’t fit in the memory pointed to by \( buf \) undefined behavior and data corruption might result. If the region is partially outside the boundaries of the frame, GetRectangle() will copy only the parts of the rectangle that are within the frame.

GetRectangle() will fail if the specified rectangle is entirely outside the boundaries of the frame, if the frame can’t be read, or if the frame contains planar data.

See Also
PutRectangle()

GetRow()

Syntax
short GetRow(FRAME __PX_FAR *f, void __PX_HUGE *buf, short row);

Return Value
Non-zero if successful.
0 on failure.

Description
Copies a row of the image stored in frame \( f \) into the buffer \( buf \). The rows are numbered starting with 0 at the top of the frame. The buffer is assumed to be an array of the correct type to hold the row of pixels. If the entire row won’t fit in the memory pointed to by \( buf \) undefined behavior and data corruption might result.

GetRow() will fail if the specified row is outside the boundaries of the frame, if the frame can’t be read, or if the frame contains planar data.

See Also
GetColumn(), PutColumn(), PutRow()
OpenLibrary()

**DOS Syntax**
short FRAME_OpenLibrary(FRAMELIB __PX_FAR *interface,
short sizeof(interface));

**Win C Syntax**
short imagenation_OpenLibrary(LPSTR dllname, __PX_FAR *interface,
short sizeof(interface));

**Win VB Syntax**
integer OpenLibrary(0,0)

**Return Value**
Non-zero if successful.
0 on failure.

**Description**
Initializes library data structures. It must be called successfully before
any other library functions can be used.

For more information on using OpenLibrary(), see *Initializing and Exit-
ing Libraries*, on page 34.

**See Also**
CloseLibrary()

PutColumn()

**Syntax**
void PutColumn(void __PX_HUGE *buf, FRAME __PX_FAR *f,
short col);

**Return Value**
Non-zero if successful.
0 on failure.

**Description**
Copies the data stored in the buffer "buf" into a column of frame "f". The col-
umns are numbered starting with 0 at the left of the frame. The buffer is
assumed to be an array of the correct type to hold the column of pixels. If "buf"
doesn’t point to enough data to hold an entire column, undefined
behavior and illegal memory accesses might result.

PutColumn() will fail if the specified column is outside the boundaries of
the frame, if the frame can’t be written, or if the frame contains planar
data.

**See Also**
GetColumn(), GetRow(), PutRow()
**Imagenation**

---

**PutPixel()**

**Syntax**

```c
short PutPixel(void __PX_HUGE *pixel, FRAME __PX_FAR *f, short x, short y);
```

**Return Value**

Non-zero if successful.

0 on failure.

**Description**

Copies the data pointed to by `pixel` into location `(x,y)` in the frame, where `0,0` is the upper-left corner of the frame. The parameter `pixel` is assumed to point to a variable or structure of the correct type to hold the pixel. If `pixel` doesn’t point to an object of sufficient size to hold the pixel, undefined behavior and illegal memory accesses might result. If the frame is planar, `pixel` must point to an object that holds one pixel from each plane, appended in order (example: YUV422P frames require a byte of brightness, followed by a byte of red, followed by a byte of blue, for a total of 24 bits).

If the point specified by `(x,y)` is outside the boundaries of the frame, or the frame can’t be read, the function call fails.

**See Also**

`GetPixel()`

---

**PutRectangle()**

**Syntax**

```c
void PutRectangle(void __PX_HUGE *buf, FRAME __PX_FAR *f, int x1, short y1, short dx, short dy);
```

**Return Value**

Non-zero if successful.

0 on failure.

**Description**

Copies a rectangular region from buffer `buf` into the frame `f`. The rectangle goes into the frame with its upper left corner at `(x0,y0)`, width `dx`, and height `dy`. The buffer is assumed to be an array of the correct type to hold the rectangle of pixels as a series of concatenated lines. If `buf` doesn’t point to enough data to hold the entire rectangle, undefined behavior and illegal memory accesses might result. If the specified rectangle is partly outside the frame boundaries, only the data within the frame boundaries is written.
PutRectangle() fails if the specified rectangle is entirely outside the boundaries of the frame, if the frame can’t be written, or if the frame contains planar data.

**See Also**

GetRectangle()

### PutRow()

**Syntax**

short PutRow(void __PX_HUGE *buf, FRAME __PX_FAR *f, short row);

**Return Value**

Non-zero if successful.

0 on failure.

**Description**

Copies the data stored in the buffer `buf` into a row of frame `f`. The rows are numbered starting with 0 at the top of the frame. The buffer is assumed to be an array of the correct type to hold the row of pixels. If `buf` doesn’t point to enough data to hold an entire row, undefined behavior and illegal memory accesses might result.

PutRow() will fail if the specified row is outside the boundaries of the frame, if the frame can’t be written, or if the frame contains planar data.

**See Also**

GetColumn(), GetRow(), PutColumn()
Imagenation

ReadBin()

Syntax
short ReadBin(FRAME __PX_FAR *f, char __PX_FAR *filename);

Return Value
The return values are:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>The file was read successfully.</td>
</tr>
<tr>
<td>FILE_OPEN_ERROR</td>
<td>The specified file could not be opened.</td>
</tr>
<tr>
<td>BAD_READ</td>
<td>An error occurred while a file was being read.</td>
</tr>
<tr>
<td>BAD_FILE</td>
<td>The file being read is not of the correct format.</td>
</tr>
<tr>
<td>INVALID_FRAME</td>
<td>The frame pointer is invalid or the frame data can’t be accessed.</td>
</tr>
<tr>
<td>FRAME_SIZE</td>
<td>The frame is not large enough to hold the data being read.</td>
</tr>
</tbody>
</table>

Description
Reads the unformatted binary file filename and copies it into frame buffer f. The function stores as much of the contents of the file in the buffer as will fit. If the type of data in the file does not match the data type of the frame, the data will interpreted as if it were in the frame’s data format. For planar frames, each plane is read from the file in order.

If the data in the file is too large to fit in the frame, the function reads as much data as will fit and returns the FRAME_SIZE error. If the file doesn’t contain enough data to fill the frame, the entire file is read, the remainder of the frame is set to zero, and the function returns the FRAME_SIZE error.

ReadBin() opens and closes the file.

See Also
WriteBin()
ReadBMP()

Syntax
short ReadBMP(FRAME __PX_FAR *f, char __PX_FAR *filename);

Return Value
The return values are:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>The file was read successfully.</td>
</tr>
<tr>
<td>FILE_OPEN_ERROR</td>
<td>The specified file could not be opened.</td>
</tr>
<tr>
<td>BAD_READ</td>
<td>An error occurred while a file was being read.</td>
</tr>
<tr>
<td>BAD_FILE</td>
<td>ReadBMP() attempted to read a non-BMP-formatted file.</td>
</tr>
<tr>
<td>INVALID_FRAME</td>
<td>The frame pointer is invalid or the frame data can’t be accessed.</td>
</tr>
<tr>
<td>FRAME_SIZE</td>
<td>The frame is not large enough to hold the data being read.</td>
</tr>
</tbody>
</table>

Description
Reads the image stored in the BMP file filename and copies it into frame buffer f. Y8 images are read from 8-bit-per-pixel BMP files, RGB images are read from 24-bit, true-color BMP files, with low-order bits discarded to match the RGB pixel type format as necessary. Attempting to read files with any other pixel format results in an error.

If the frame is larger than the image data in the file, the data appears in the upper-left corner of the frame with the remainder of the frame set to zero. If the frame is smaller than the image, the upper-left portion of the image is read into the frame, and the FRAME_SIZE error is returned.

ReadBMP() opens and closes filename.

See Also
WriteBMP()
WriteBin()

Syntax

short WriteBin(FRAME __PX_FAR *f, char *filename, short overwrite);

Return Value

The return values are:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>The file was written successfully.</td>
</tr>
<tr>
<td>FILE_EXISTS</td>
<td>The file already exists, but the function call did not specify that the file should be overwritten.</td>
</tr>
<tr>
<td>FILE_OPEN_ERROR</td>
<td>The file could not be opened.</td>
</tr>
<tr>
<td>BAD_WRITE</td>
<td>An error occurred while a file was being written.</td>
</tr>
<tr>
<td>INVALID_FRAME</td>
<td>The frame pointer is invalid or the frame’s data can’t be accessed.</td>
</tr>
</tbody>
</table>

Description

Writes the image in frame buffer f to the file filename. No information about the image (height, width, and bits per pixel) is written, only the pixel values. Data in the file exactly matches the format of the data in memory. Planar frames are written to the file plane by plane.

If filename already exists and overwrite is zero, the function returns an error; otherwise, the contents of filename are overwritten. WriteBin() opens and closes the file.

See Also

ReadBin()
WriteBMP()

Syntax

```c
short WriteBMP(FRAME __PX_FAR *f, char __PX_FAR *filename, short overwrite);
```

Return Value

The return values are:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>The file was written successfully.</td>
</tr>
<tr>
<td>FILE_EXISTS</td>
<td>The file already exists, but the function call did not specify that the file should be overwritten.</td>
</tr>
<tr>
<td>FILE_OPEN_ERROR</td>
<td>The file could not be opened.</td>
</tr>
<tr>
<td>BAD_WRITE</td>
<td>An error occurred while a file was being written.</td>
</tr>
<tr>
<td>INVALID_FRAME</td>
<td>The frame pointer is invalid or the frame data can’t be accessed.</td>
</tr>
<tr>
<td>WRONG_BITS</td>
<td>The file format does not accept data of the type contained in the frame</td>
</tr>
</tbody>
</table>

Description

Writes the image stored in frame buffer `f` to the file `filename` in the BMP format. Y8 images are written as 8-bits-per-pixel BMP files with a grayscale palette. RGB images are written as 24-bit, true-color BMP files. Any alpha channel data is ignored. Attempting to write floating-point formats, Y16, and YUV formats results in an error.

If `filename` already exists and `overwrite` is zero, the function returns an error; otherwise, the contents of `filename` are overwritten. WriteBMP() opens and closes the file `filename`.

See Also

ReadBMP()
The VGA Video Display Library

The VGA Video Display library is a DOS-based VGA display and menu builder. The library makes it easy to create and display a graphics menu-based interface for a program. Imagenation used this library to create the interface for PXCVU and for most of the DOS sample programs.

This library is written in C and comes in several versions:

- **VIDEO_LB.LIB**—Turbo, version 3.0 and later and Borland, version 3.1 and later.
- **VIDEO_LM.LIB**—Microsoft, version 6.0 and later.
- **VIDEO_LW.LIB**—Watcom 16-bit compiler version 10.6 and later.
- **VIDEO_FW.LIB**—Watcom DOS/4GW version 10.6 and later.

The library provides functions for the following purposes:

- Entering, configuring, and exiting graphics mode
- Selecting fonts and displaying text strings
- Drawing lines and rectangles
- Creating and displaying menus
In order to use this VGA Video Display library, your video card and monitor must be VESA-compatible.

**Initializing and Exiting the Library**

Before you call any other VGA Video Display functions, you must call `VGALIB_OpenLibrary()`. The VGALIB_OpenLibrary() function initializes the library and sets up the interface for calling functions (for more information on function calling conventions, see *Programming in C*, on page 30.)

After making the last VGA Video Display function call and before exiting your program, you must call `VGALIB_CloseLibrary()`. VGALIB_CloseLibrary() frees any resources allocated when the library was initialized.

**Entering and Exiting VGA Graphics Mode**

After initializing the VGA Video Display library, but before calling any other VGA Video Display functions, you must call `AllocateVGA()`. The AllocateVGA() function saves the current display mode, sets the display to the specified graphics mode, initializes some global data structures, and returns a pointer to a frame. You can use the frame pointer returned by AllocateVGA() to operate on the VGA display with functions from both the VGA Video Display library and the Frame library. You specify the graphics mode by specifying a resolution, (dx,dy), and a pixel data type. The valid pixel data types are PBITS_Y8, PBITS_RGB15, PBITS_RGB16, PBITS_RGB24, and PBITS_RGB32. (For more information on pixel data types, see *Allocating and Freeing Frames*, on page 39.)

After making the last VGA Video Display function call, but before calling VGALIB_CloseLibrary(), you must call `FreeFrame()`. FreeFrame()
resets the display mode to the mode that was active before the call to AllocateVGA().

Displaying VGA Text and Graphics

The color for both text and graphics can be controlled using the following library functions:

- **SetColor()**—Sets the current foreground color to the RGB values specified.
- **GetColor()**—Returns the R, G, or B value of the currently selected color.

The basic functions this library provides for displaying text are:

- **DrawTextString()**—Draws a string of text in the current color, beginning at a specified location (x, y).
- **SetFontSize()**—Selects one of the three fonts: 8x8, 8x14, or 8x16.
- **GetFontSize()**—Returns the currently selected font.

The library provides the following graphics operations:

- **DrawLine()**—Draws a line in the current color. You specify the two endpoints of the line.
- **DrawRectangle()**—Draws a rectangle in the current color. You specify the coordinates of the upper-left corner and the width and height.
- **FillRectangle()**—Draws a filled rectangle in the current color. You specify the coordinates of the upper-left corner and the width and height.
The library provides the following functions for locating the current cursor position following a text or drawing operation:

**WhereX()**—The current horizontal position of the cursor.

**WhereY()**—The current vertical position of the cursor.

### VGA Memory Addressing

Addressing the display memory on a VGA controller often requires swapping pages of memory. The library functions for the VGA Video Display library and the Frame library automatically handle any page swapping. This means that you can’t treat the VGA frame as if it were a single, contiguous block of memory. You can’t use the FrameBuffer() function to get a pointer to that memory and then operate directly on the memory using that pointer. Similarly, the AliasFrame() and FrameAddress() functions can’t be used with frames allocated by AllocateVGA().

### Menu Creation, Configuration, and Display

A menu is a data structure whose contents can be manipulated and displayed using the **MenuSelect()** and **MenuDisplay()** functions. All menus must be successfully initialized by the **MenuGenerate()** function before they are referenced by any other function; however, some fields in the **menu** and **menuitem** structures must be initialized by the application before MenuGenerate() is called. For more information, see *Menu Structure*, on page 123 and **MenuGenerate()**, on page 132.

The MenuSelect() function is used to change the currently highlighted menu option. Its return value indicates which (if any) menu option has been selected. This return value can be used, for example, to select which of a variety of functions should be executed.
Menu Structures and Types

Menu Structure

```c
struct menu
{
    short xmin, ymin, dx, dy;
    short rows, cols;
    short numitems;
    char *title;
    short highlight;
    PIX_RGB32 standardc, standardcbk;
    PIX_RGB32 highc, highcbk;
    PIX_RGB32 menuc, menucbk;
    PIX_RGB32 helpc, helpcbk;
    menuitem *data;
}
```

This structure defines a menu. All of these values must be initialized before `MenuGenerate()` is called unless otherwise specified:

- **xmin, ymin**—Define the upper left-hand corner on the screen where the menu will be drawn.

- **dx, dy**—Define the height and width of the menu.

- **rows, cols**—Define the number of rows and columns in which the menu items will be organized and displayed; these values are set by the `MenuGenerate()` function.

- **numitems**—Defines the number of items in the menu.

- **title**—Points to the title, if any, of the menu. The title appears in the menu title bar. A menu that doesn’t have a title must initialize this pointer to NULL.
highlight—Defines which of the menu items is currently selected.

standardc, standardcbk—Colors used to display all menu features except menu items and help.

highc, highcbk—Colors used to display the highlighted menu items.

menuc, menucbk—Colors used to display non-highlighted menu items.

helpc, helpcbk—Colors used to display single-line help messages for highlighted menu items at the bottom of the screen.

*data—Points to the menuitem structures and is usually set to point to an array.

Menuitem Structure

```
struct menuitem
typedef struct tagmenuitem
{
    short xoff, yoff;
    short i, j;
    char *text;
    short hotkey;
    char *help;
}menuitem;
```

This structure defines a menu item. All of these values must be initialized before calling MenuGenerate() on the associated menu, unless otherwise specified:

xoff, yoff—Define the item's display coordinates relative to the menu's upper left-hand corner; these values are set by MenuGenerate().
i, j—Define the item's (row, column) coordinates in the menu display; these values are set by MenuGenerate().

*text—Points to the text string in the menu that describes this item.

hotkey—Defines a hotkey that can be used to select this menu item. If no hotkey is desired, set this field to zero.

*help—Defines the text string that will be displayed at the bottom of the screen when this item is selected. The string should describe the function of this menu item.

Function Reference

AllocateVGA()

Syntax

FRAME __PX_FAR *AllocateVGA(short dx, short dy, unsigned short type);

Return Value

A pointer to a frame if successful.
NULL if unsuccessful.

Description

Puts the VGA display into the graphics mode with a resolution of dx x dy and a pixel type of type. Valid pixel types are PBITS_Y8, PBITS_RGB15, PBITS_RGB16, PBITS_RGB24, and PBITS_RGB32. (For more information on pixel data types, see Allocating and Freeing Frames, on page 39.)

If the VGA display doesn’t support the requested mode, the function returns NULL.

You can use the frame pointer returned by AllocateVGA() to operate on the VGA display with functions from both the VGA Video Display library and the Frame library. This means that you can use Frame library functions, such as PutRectangle() to draw to the VGA screen.
Imagenation

Programs must call VGALIB_OpenLibrary() and AllocateVGA(), in that order, before calling any other VGA Video Display library function.

Note:

*It is also possible to use the graphics functions from the VGA Video Display library on a frame allocated with AllocateBuffer(). In that case, you must call VGALIB_OpenLibrary(), but not AllocateVGA().

See Also: FreeFrame(), VGALIB_OpenLibrary(), ChangeResolution()

ChangeResolution()

Syntax

FRAME __PX_FAR *ChangeResolution(FRAME __PX_FAR *f, short dx, short dy, unsigned short type);

Return Value

Non-zero if successful.
0 on failure.

Description

Changes the VGA display to the mode with a resolution of \( dx \times dy \) and a pixel type of type. After setting the original display mode with AllocateVGA(), you can change the display mode by calling ChangeResolution() with the frame pointer \( f \) returned by AllocateVGA(). If the resolution is changed successfully, the frame \( f \) is no longer valid; you must use the new frame returned by this function for all subsequent operations. Valid pixel types are PBITS_Y8, PBITS_RGB15, PBITS_RGB16, PBITS_RGB24, and PBITS_RGB32. (For more information on pixel data types, see Allocating and Freeing Frames, on page 39.)

If the VGA display doesn’t support the requested mode, the function returns NULL, and the display mode is unchanged.

See Also: AllocateVGA()
DisplayMsg()

Syntax
void DisplayMsg(menu *m, FRAME __PX_FAR *f, char *msg);

Return Value
None.

Description
Displays the text string pointed to by msg at the bottom of the display.

See Also
DrawTextString()

DrawLine()

Syntax
short DrawLine(FRAME __PX_FAR *f, short x0, short y0, short x1, short y1);

Return Value
The length of the line if successful.
NULL if the specified location is outside the boundaries of the screen.

Description
Draws a line on the frame f from (x0, y0) to (x1, y1) in the current color.

See Also
SetColor()

DrawRectangle()

Syntax
short DrawRectangle(FRAME __PX_FAR *f, short x0, short y0, short dx, short dy);

Return Value
Non-zero if successful.
0 on failure.

Description
Draws an unfilled rectangle on the frame f with upper-left corner at
(x0, y0) in the current color. The rectangle is dx pixels wide and dy pixels
tall.

See Also
FillRectangle(), SetColor()
### Imagenation

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DrawTextString()</strong></td>
<td>Draws a string of text on the frame ( f ) starting at location ((x_0, y_0)) in the current color.</td>
</tr>
<tr>
<td><strong>FillRectangle()</strong></td>
<td>Draws a filled rectangle on the frame ( f ) with upper-left corner at ((x_0, y_0)) in the current color. The rectangle is (dx) pixels wide and (dy) pixels tall.</td>
</tr>
<tr>
<td><strong>FreeFrame()</strong></td>
<td>Resets the display to the mode it was in just before AllocateVGA() was called. Programs must call FreeFrame() after all other VGA Video Display functions have been called, but before calling VGALIB_CloseLibrary().</td>
</tr>
</tbody>
</table>

#### DrawTextString()

**Syntax**

```
short DrawTextString(FRAME __PX_FAR *f, short x0, short y0, char *string);
```

**Return Value**

- Non-zero if successful.
- NULL if the total length of the string is outside the boundaries of the screen.

**Description**

Draws a string of text on the frame \( f \) starting at location \((x_0, y_0)\) in the current color.

**See Also**

SetColor(), SetFontSize()

#### FillRectangle()

**Syntax**

```
short vgalib.FillRectangle(FRAME __PX_FAR *f, short x0, short y0, short dx, short dy);
```

**Return Value**

- Non-zero if successful.
- 0 on failure.

**Description**

Draws a filled rectangle on the frame \( f \) with upper-left corner at \((x_0, y_0)\) in the current color. The rectangle is \(dx\) pixels wide and \(dy\) pixels tall.

**See Also**

DrawRectangle(), SetColor()

#### FreeFrame()

**Syntax**

```
void FreeFrame(FRAME __PX_FAR *f);
```

**Return Value**

None.

**Description**

Resets the display to the mode it was in just before AllocateVGA() was called. Programs must call FreeFrame() after all other VGA Video Display functions have been called, but before calling VGALIB_CloseLibrary().

**See Also**

AllocateVGA(), VGALIB_CloseLibrary()
GetBkColor()
Syntax  short GetBkColor(FRAME __PX_FAR *f, short color);
Return Value  The current background color if successful.
               NULL if color is not supported.
Description  Returns the current value for color for the background, where color is one
               of RED, GREEN, BLUE, or ALPHA. Values can range from zero to 255.
See Also    SetBkColor()

GetColor()
Syntax  short GetColor(FRAME __PX_FAR *f, short color);
Return Value  The current foreground color if successful.
               NULL if color is not supported.
Description  Returns the current value for color for the foreground, where color is one
               of RED, GREEN, BLUE, or ALPHA. Values can range from zero to 255.
See Also    SetColor()

GetFontSize()
Syntax  short GetFontSize(void);
Return Value  The currently selected font number on success.
               NULL if the specified font is not supported.
Description  Returns the font number of the currently selected font. There are three
               fonts available: 8x8, 8x14, and 8x16, numbered 1, 2, and 3 respectively.
See Also    DrawTextString(), SetFontSize()

GetKey()
Syntax  short GetKey(void);
Return Value  The scan code of the key hit.
Imagenation

Description
Waits for a key to be depressed, and then returns the scan code for the key. This library has definitions for the following non-standard ASCII keys and key combinations: the arrow keys, page up, page down, insert, delete, home, end, the function keys, and CONTROL + the arrow keys. The definitions are in the file VIDEO.H. The MenuSelect() function uses some of these special keys, so it should take its input from GetKey().

See Also
MenuSelect()

MenuCalcDx()

Syntax
short MenuCalcDx(menu *m, FRAME __PX_FAR *f, short columns);

Return Value
The calculated menu width.

Description
Calculates the width in pixels that the menu m should be if its items are arranged in a number of columns equal to columns. This calculation is based on the width of each menu item and the width in pixels of the text (as defined by SetFontSize()).

For more information, see Menu Structure, on page 123, and MenuItem Structure, on page 124.

See Also
MenuCalcDy(), MenuGenerate, SetFontSize()

MenuCalcDy()

Syntax
short MenuCalcDy(menu *m, FRAME __PX_FAR *f, short columns);

Return Value
The calculated menu height.

Description
Calculates the height in pixels that the menu m should be if its items are arranged in a number of columns equal to columns. This calculation is based on the number of items and the height in pixels of the text (as defined by SetFontSize()).

For more information, see Menu Structure, on page 123, and MenuItem Structure, on page 124.

See Also
MenuCalcDx(), MenuGenerate, SetFontSize()
MenuDisplay()

Syntax 
short MenuDisplay(menu *m, FRAME __PX_FAR *f);

Return Value 
Non-zero if successful.
0 on failure.

Description 
Displays menu m on the VGA screen at the location specified by the x and y values in the menu structure. It erases the area where the menu is to be drawn, draws a rectangle to frame the menu, displays the menu options and title, displays (at the bottom of the screen) the help text for the currently-selected menu option, and highlights the currently selected menu option.

For more information, see Menu Structure, on page 123, and Menuitem Structure, on page 124.

See Also 
MenuErase()

MenuErase()

Syntax 
void MenuErase(menu *m, FRAME __PX_FAR *f);

Return Value 
None.

Description 
Erases the menu m from the VGA display by calling FillRectangle(menu->xmin, menu->ymin, menu->dx, menu->dy, colors.standardbk). It does not check, before erasing this area, to see whether the menu was actually displayed on the VGA monitor.

For more information, see Menu Structure, on page 123, and Menuitem Structure, on page 124.

See Also 
MenuDisplay()
Imagenation

MenuGenerate()

Syntax

```
short MenuGenerate(menu *m, FRAME __PX_FAR *f);
```

Return Value

Return values are:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Menu successfully initialized.</td>
</tr>
<tr>
<td>MENU_BOUNDS_ERR</td>
<td>Menu screen coordinates off screen or otherwise invalid.</td>
</tr>
<tr>
<td>MENU_WIDTH_ERR</td>
<td>Menu not wide enough to hold a menu item.</td>
</tr>
<tr>
<td>MENU_HEIGHT_ERR</td>
<td>Menu not tall enough for specified width and number of menu items.</td>
</tr>
</tbody>
</table>

Description

Sets up some internal data in menu \( m \) required by the menu functions. In order for MenuGenerate() to function properly, several items in the menu structure must be initialized before MenuGenerate() is called: xmin, ymin, dx, dy, numitems, *data, and *title. (*title may be initialized to NULL if you don't want your menu to have a title, but it can't be left uninitialized.)

The MenuGenerate() function assumes that all menu item names have the same number of characters. The function calculates the number of rows for the displayed menu based on the height of the menu and of the individual characters, and then calculates the number of columns based on the number of rows and number of items. The MenuGenerate() function will fail under the following circumstances:

- The menu coordinates are off-screen.
- With the given origin, the menu is too wide to fit on the screen.
- The menu is not wide enough, based on the width of each menu item name and the number of columns.
- The menu is not tall enough, based on the width in pixels of the menu and the number of menu items.
The return value of MenuGenerate() should always be checked for errors before menu m is used with any other VGA Video Display function.

For more information, see Menu Structure, on page 123, and Menuitem Structure, on page 124.

**See Also**

MenuCalcDx(), MenuCalcDy(), MenuDisplay()

---

**MenuSelect()**

**Syntax**

```
short MenuSelect(menu *m, FRAME __PX_FAR *f, short key);
```

**Return Value**

Return values are:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>No selection made.</td>
</tr>
<tr>
<td>0 to m-&gt;numitems - 1</td>
<td>Index of selected menu item.</td>
</tr>
</tbody>
</table>

**Description**

Changes the highlighted menu option depending on the key that is input, or returns the index of the highlighted menu item if the key is RETURN or a defined hotkey for that menu item. The following keys have special meaning to MenuSelect():

- **Left and Right Arrows**—Move selection left or right by one column.
- **Up and Down Arrows**—Move selection up or down by one row.
- **PAGE UP and PAGE DOWN**—Move selection to top or bottom of current column.
- **HOME and END**—Move selection to first or last menu item.

For more information, see Menu Structure, on page 123, and Menuitem Structure, on page 124.
SetBkColor()

Syntax    short SetBkColor(FRAME __PX_FAR *f, PIX_RGB32 __PX_FAR *color);

Return Value   Non-zero if successful.
                NULL if color is not supported.

Description  Sets the current background color to the RGB values specified. For each color component, values can range from zero to 255.

See Also   GetBkColor()

SetColor()

Syntax    short SetColor(FRAME __PX_FAR *f, PIX_RGB32 __PX_FAR *color);

Return Value   Non-zero if successful.
                NULL if color is not supported.

Description  Sets the current foreground color to the RGB values specified. For each color component, values can range from zero to 255.

See Also   GetColor()

SetFontSize()

Syntax    short SetFontSize(short font_number);

Return Value   Non-zero if successful.
                NULL if the specified font is not supported.

Description  Sets the font used by DrawTextString() to font_number. There are three fonts available: 8x8, 8x14, and 8x16, with font_number 1, 2, and 3 respectively. The default (set by AllocateVGA()) is the 8x16 font.

See Also   AllocateVGA(), DrawTextString()
VGALIB_CloseLibrary()

Syntax
void VGALIB_CloseLibrary(VGALIB __PX_FAR *interface);

Return Value
None.

Description
Releases any resources allocated by VGALIB_OpenLibrary(). Programs
must call VGALIB_CloseLibrary() before exiting.

See Also
VGALIB_OpenLibrary()

VGALIB_OpenLibrary()

Syntax
short VGALIB_OpenLibrary(VGALIB __PX_FAR *interface,
short sizeof(interface));

Return Value
Non-zero if successful.
0 on failure.

Description
Initializes the library and fills in the interface structure, where interface is
the name you will use for calling other library functions (for more information on calling conventions, see Programming in C, on page 30).

See Also
VGALIB_CloseLibrary()

WhereX()

Syntax
short WhereX(void);

Return Value
The horizontal position of the cursor.
-1 on failure.

Description
Returns the horizontal position, in pixels, of the cursor following a
DrawLine(), DrawRectangle(), or DrawTextString() function call.

See Also
WhereY()
Imagenation

WhereY()

Syntax       short WhereY(void);

Return Value The vertical position of the cursor.
               -1 on failure.

Description  Returns the vertical position, in pixels, of the cursor following a
               DrawLine(), DrawRectangle(), or DrawTextString() function call.

See Also     WhereX()
This chapter includes information on making cables for the PXC200 frame grabber.

**Standard PCI Bus Cables**

You can make cables using the pinout information in the next section.

**26-pin D Connector**

Pinouts for the 26-pin D connector on the PXC200 is shown below:
Connecting the +12V Output

To activate the +12V output on pin 9, you must connect the board to the computer’s power supply. You make this connection using the same type of connectors used to power the disk drives.

PC/104-Plus Cables

Connector and pinout information for the PC/104-Plus configuration of the PXC200 was not available at printing time and will be listed in the release notes for the product.
# Hardware Specifications

This appendix lists specifications for the PXC200 hardware.

<table>
<thead>
<tr>
<th>Input video formats</th>
<th>NTSC, PAL, SECAM, S-Video.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input video signal</td>
<td>1 V peak-to-peak, 75 Ω.</td>
</tr>
<tr>
<td>Resolution</td>
<td>NTSC: 640 x 480 pixels</td>
</tr>
<tr>
<td></td>
<td>PAL/SECAM: 768 x 576 pixels.</td>
</tr>
<tr>
<td>Sampling jitter</td>
<td>Maximum of ±4 ns relative to horizontal synchronization (for a stable source).</td>
</tr>
<tr>
<td>Output formats</td>
<td>Color: YCrCb 4:2:2; RGB 32, 24, 16, and 15.</td>
</tr>
<tr>
<td></td>
<td>Monochrome: Y8</td>
</tr>
<tr>
<td>External trigger</td>
<td>Input pulled up by 10 KΩ to 5 V. Trigger requires a TTL pulse of 100 ns minimum. Software programmable edge or level sensitivity and polarity.</td>
</tr>
<tr>
<td>Over-voltage protection</td>
<td>All inputs and outputs are diode protected.</td>
</tr>
<tr>
<td>Specification</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Form factor           | PCI short card: 174.6 x 106.7 mm 6.875 x 4.2 in.  
                        | PC/104 Plus module: 91.4 x 96.5 mm 3.4 x 3.6 in.                           |
| Video noise           | ≤ 1 LSB (least significant bit) RMS.                                         |
| Power                 | +5 VDC.                                                                     |
| Camera power          | +12 VDC output.                                                             |
| Video multiplexer     | Four video inputs, only one of which can be S-Video; all four can be composite video. |
| Operating temperature | 0° C to 60° C.                                                              |
| Warranty              | One-year limited parts and labor.                                           |
A block diagram of the PXC200 board is shown on the following page.
Imagenation

I/O Control
Processor

I/O Buffers

Video 0
Video 1
Video 2
Video 3

Sync Detect

Digitize, Decode, & Scale

Color Space & Format Convert

PCI Engine

PCI BUS

Trigger 0
Index

Numerics
26-pin D connector 137
386MAX 13

A
accessing frame grabbers 37
addresses
   logical 64
   physical 41, 65
adjusting the video image 45
AGC 48
allocating frame grabbers 37
   multiple frame grabbers 38, 71
AUTOEXEC.BAT file 14
automatic gain control 48

B
BBS phone number 20
binary files 66
block diagram 141–142
BMP files 65
board diagram 141–142
board serial number 63
brightness 45
buffers, Visual Basic 32

C
cables 9, 137–138
CACHE flag 57
camera inputs 44
capture resolution 49–50
capturing images 42–43
comb filter 47, 48
compiling programs 26–30
CompuServe address 20
CONFIG.SYS file 13
connectors 9, 137–138
continuous acquire mode 81
contrast 45
core function 47
corrupt image data 43
cropping images 50
customer support 20

digital I/O 57
direct memory access 41, 42
directories 17
DLLs
  error loading 18
  FRAME_31.DLL 28
  FRAME_95.DLL 29
  PXC2_31.DLL 28
  PXC2_95.DLL 29
  Video Display 66
  VIDEO_16.DLL 67
  VIDEO_32.DLL 67
  Windows 3.1 28
  Windows 95 29
  Windows Video Display DLL 67
DMA 41, 42
DOS Install program 13

E
EITHER flag 57
EMM386 13
environment variables 14, 19, 21
errors
  error loading DLL 18
  error loading VxD 18
execution timing 51–56
exiting libraries 34, 120
external triggers 6

F
FIELD0 flag 57
FIELD1 flag 57
files
  AUTOEXEC.BAT 14
  BIN format 66
  binary 66
  BMP format 65
  CONFIG.SYS 13
  PXCVU.HLP 21
  PXCVU.INI 21
reading and writing 65
  SYSTEM.INI 15
  VIDEO_16.BAS 67
  VIDEO_32.BAS 67
  WPXC2_31.BAS 31, 32
  WPXC2_95.BAS 31
  flags 52, 54, 55, 57
frame buffers
  error trying to allocate 41
  memory allocation 15
frame grabber handles 37
FRAME.H file 27, 28, 29
FRAME_31.DLL 28
FRAME_95.DLL 29
FRAME_FW.LIB library 27
FRAME_LB.LIB library 27
FRAME_LM.LIB library 27
FRAME_LW.LIB library 27
freeing frame grabbers 37
  PXCLEAR program 38
freeing memory 39
function flags 57
function reference 69–100, 101–117, 125–136
function timing 51–56
functions
  AliasFrame() 102
  AllocateAddress() 41, 103
  AllocateBuffer() 39, 70
  AllocateFG() 37, 71
  AllocateFlatFrame() 65, 103
  AllocateMemoryFrame() 65, 104
  AllocateVGA() 120, 125
  ChangeResolution() 126
  CheckError() 37, 43, 62, 64, 72
  CloseLibrary() 34, 36, 72, 105
  CopyFrame() 64, 105
  DisplayMsg() 64, 105
  DrawLine() 121, 127
  DrawRectangle() 121, 127
  DrawTextString() 121, 128
  ExtractPlane() 64, 106
  FillRectangle() 121, 128
Index

FRAME_CloseLibrary() 35
FRAME_OpenLibrary() 35
FrameAddress() 65, 106
FrameBuffer() 64, 107
FrameHeight() 65, 107
FrameType() 65, 108
FrameWidth() 65, 107
FreeFG() 37, 73
FreeFrame() 41, 73, 120
GetBkColor() 129
GetBrightness() 45, 73
GetCamera() 44, 74
GetChromaControl() 49, 74
GetColor() 121, 129
GetColumn() 64, 109
GetContrast() 45, 74
GetFontSize() 121, 129
GetHeight() 50, 75
GetHue() 46, 75
GetInterface() 76
GetIOType() 58, 76
GetKey() 129
GetLeft() 50, 77
GetLumaControl() 48, 77
GetPixel() 64, 109
GetRectangle() 64, 110
GetRow() 64, 110
GetSaturation() 46, 77
GetSwitch() 60, 78
GetSyncThreshold() 78
GetTop() 50, 78
GetVideoDetect() 44, 79
GetVideoLevel() 47, 79
GetWidth() 50, 79
GetXResolution() 80
GetYResolution() 80
Grab() 42, 80
GrabContinuous() 42, 81
imagination_CloseLibrary() 34, 72, 105
imagination_OpenLibrary() 34, 83, 111
immediate 54
IsFinished() 82
KillQueue() 54, 82
MenuCalcDx() 130
MenuCalcDy() 130
MenuDisplay() 122, 131
MenuErase() 131
MenuGenerate() 122, 132
MenuSelect() 122, 133
OpenLibrary() 34, 36, 83, 111
PutColumn() 64, 111
PutPixel() 64, 112
PutRectangle() 64, 112
PutRow() 64, 113
PX200_CloseLibrary() 35
pxc200_CloseLibrary() 72, 105
PX200_OpenLibrary() 35
pxc200_OpenLibrary() 83, 111
pxPaintDisplay() 66
pxSetWindowSize() 66
queued 52, 54
ReadBin() 66, 114
ReadBMP() 65, 115
ReadIO() 58, 61, 83
ReadProtection() 63, 84
ReadRevision() 62, 84
ReadSerial() 63, 84
Reset() 62, 85
SetBkColor() 134
SetBrightness() 45, 85
SetCamera() 44, 85
SetChromaControl() 49, 86
SetColor() 121, 134
SetContrast() 45, 87
SetFontSize() 121, 134
SetHeight() 50, 87
SetHue() 46, 88
SetIOType() 58, 88
SetLeft() 50, 89
SetLumaControl() 48, 90
SetPixelFormat() 42, 91
SetSaturation() 46, 91
Imagenation

SetTop() 50, 92
SetVideoDetect() 44, 92
SetVideoLevel() 47, 93
SetWidth() 50, 94
SetXResolution() 49, 94
SetYResolution() 49, 95
SwitchCamera() 61, 95
SwitchGrab() 61, 95
VGALIB_CloseLibrary() 120, 135
VGALIB_OpenLibrary() 120, 135
VideoType() 44, 96
Wait() 54, 96
WaitAllEvents() 59, 97
WaitAnyEvent() 59, 98
WaitFinished() 29, 53, 98
WaitVB() 29, 53, 99
WhereX() 122, 135
WhereY() 122, 136
WriteBin() 66, 116
WriteBMP() 65, 117
WriteImmediateIO() 61, 99

G

gamma correction 47
grabbing images 42–43
  incomplete image captures 43
  invalid data in buffer 43
grayscale noise 2

H

handles 37
hardware installation 10–12
hardware protection key 63
hardware serial number 63
hardware specifications 139–140
header files 17
  DOS 27
  FRAME.H 27, 28, 29
  PXC200.H 27, 28, 29
  VIDEO.H 27
  VIDEO_16.H 67
  VIDEO_32.H 67
  Watcom DOS/4GW 27, 28, 29
  Windows Video Display DLL 67
high-frequency gain filter 48
hue 45

I

ILIB_31.LIB library 28
ILIB_95.LIB library 29
image adjustments 45
image cropping 50
image resolution 49–50
image scaling 49
IMAGENATION variable 14, 19, 21
IMMEDIATE flag 54, 55, 57
immediate functions 54
initializing libraries 34, 120
input/output 57
inputs, video 44
INSTALL program 13
installation 9–20
installing the PX board 10–12
installing the PX software 12–17
Internet address 20
interrupt handlers 35
interrupts 37
IRQ conflicts 18, 19, 37

L

languages, programming 30–33
libraries
  Borland, DOS 27
  compiling and linking 26–30
  error when initializing 36
  exiting 34, 120
  FRAME_FW.LIB 27
  FRAME_LB.LIB 27
  FRAME_LM.LIB 27
  FRAME_LW.LIB 27
  Watcom DOS/4GW 27, 28, 29
  Windows Video Display DLL 67
Index

function reference 69–100, 101–117, 125–136
ILIB_31.LIB 28
ILIB_95.LIB 29
initializing 34, 120
Microsoft, DOS 27
PXC2_FW.LIB 27
PXC2_LB.LIB 27
PXC2_LM.LIB 27
PXC2_LW.LIB 27
troubleshooting 36
VGA Video Display 119–136
VIDEO_16.LIB 67
VIDEO_32.LIB 67
VIDEO_FW.LIB 27
VIDEO_LB.LIB 27
VIDEO_LM.LIB 27
VIDEO_LW.LIB 27
Watcom DOS/4GW 27
Windows 3.1 28
Windows 95 29
Windows Video Display DLL 67
linking programs 26–30
logical addresses 64
low filter 47
low-color removal 48
luma controls 47

M
memory
allocation variable 15
freeing 39
managers 13
requirements 12, 37
menus 119–136
monochrome detect 48
monochrome video controls 47
MSD program 13
multitasking and multithreaded operating systems 29

N
notch filter 48
NTSC 44, 50

O
operating systems 26–30
multitasking and multithreaded 29
Windows 95 28, 29

P
PAL/SECAM 44, 50
PATH variable 14
PC/104-Plus bus 2
cables 138
PCI BIOS 36
PCI bus 5, 63
cables 137
peak filter 48
performance 7, 63
physical addresses 41, 65
pixel jitter 2
pointers 31, 64
programming 25–63
programming languages 30–33
programs
compiling and linking 26–30
directory location 17
INSTALL 13
MSD 13
PXCDRAW1 7
PXCDRAW2 7
PXCLEAR 8, 38
PXCREV 7, 18
PXCVU 18, 21–24
SETUP 14
VGACOPY 7
protection key, hardware 63
purging the function queue 54
PX2 directory 17
Imagenation

PXC2.VXD virtual device driver 15, 28, 29
PXC2_31.DLL 28
PXC2_95.DLL 29
PXC2_FW.LIB library 27
PXC2_LB.LIB library 27
PXC2_LM.LIB library 27
PXC2_LW.LIB library 27
PXC200.H file 27, 28, 29
PXCDRAW1 program 7
PXCDRAW2 program 7
PXCIIVU.HLP file 21
PXCLEAR program 8, 38
PXCREV program 7
PXCVU.HLP file 21
PXCVU program 21–24
PXCVU.INI file 21

Q
QEMM 13
QUEUED flag 52, 55, 57
queued functions 52, 54

R
registry, Windows 95 16
requesting access to frame grabbers 37
resolution 49–50
revision numbers 7, 62

S
sample programs, see programs
saturation 45
scaling images 49
security 63
serial number, hardware 63
SETUP program 14
SINGLE_FLD flag 57
software
directories 17
installation 12–17
security 63
updates 20
source code directory location 17
specifications 139–140
StaticVxD registry key 16
structures
menu 122, 123
menuitem 122, 124
support 20
S-Video color signal 48
synchronizing program execution to video 53
system files 14
SYSTEM.INI file 15

T
technical support 20
timing, function execution 51–56
triggers 6
troubleshooting
AllocateBuffer() 41
AllocateFG() 38
can’t allocate a frame grabber 38
can’t allocate frames 41
corrupt image data 43
error loading DLL 18
error loading VxD 18
freeing frame grabbers 38
GetColumn(), GetRectangle(), GetRow() 64
grab functions fail 43
grabbing images 43
image is all black 43
incomplete image 43
invalid data in buffer 43
IRQ conflicts 18, 19, 37
library fails to initialize 36
OpenLibrary() 36
Index

partial image 19
PutColumn(), PutRectangle(), PutRow() 64
PXCREV program 18
PXCVU program 18
slow video display performance 19
Windows 19

U
updates, software 20
user interface 119–136
utility programs, see programs

V
VESA display drivers 19
VGA Video Display library 119–136
VGACOPY program 7
video
  automatic gain control 48
  brightness adjustment 45
  comb filter 47, 48
  contrast adjustment 45
  core function 47
  formats 44
  gamma correction 47
  high-frequency gain filter 48
  hue adjustment 45
  inputs 44
  level adjustment 46
  low filter 47
  monochrome detect 48
  notch filter 48
  peak filter 48
  processing adjustments 45
  saturation adjustment 45
  S-Video format 48
Video Display DLL 66
VIDEO.H file 27
VIDEO_16.BAS file 67
VIDEO_16.DLL 67
VIDEO_16.H file 67
VIDEO_16.LIB library 67
VIDEO_32.BAS file 67
VIDEO_32.DLL 67
VIDEO_32.H file 67
VIDEO_32.LIB library 67
VIDEO_FW.LIB library 27
VIDEO_LB.LIB library 27
VIDEO_L.M.LIB library 27
VIDEO_L.W.LIB library 27
virtual device drivers 15, 28, 29, 35
Visual Basic
  buffers 32
  declarations 31, 32
  End button 33
  programming tips 31
  Video Display DLL 66
VxD 15, 28, 29, 35
  error loading 18

W
Windows
  troubleshooting 19
Windows 95 28, 29
  programming tips 28
  registry changes 16
Windows Setup program 14
WPXC2_31.BAS file 31, 32
WPXC2_95.BAS file 31
Imagenation