P/N 4411-0084

PC Interface Library

Programming manual

Manual Version 2
Revision F
July 23, 1999

Table of Contents

Chapter 1 General Information 15

Princeton Instruments General Command Library 15
Functionality 15
Using PICM 16
- Basic Operation 16
- Windows 3.1 Example Programs 16
- Windows 95/NT Example Programs 17
- Using This Manual 17

Chapter 2 Installation 19

Installing PICM for Windows 3.1 19
PICM Files for Windows 3.1 19
Installing PICM for Windows/95 20
System Requirements 20
- Hardware requirements 20
- Operating System requirements 21

Installation Procedure 21
Aborting the EasyDLL95 Installation 29
Installing Some EasyDLL95 Files at a Later Time 29
Installing More than One Version of EasyDLL95 30
Uninstalling and Reinstalling 30
PICM Files for Windows 95 32

Chapter 3 Using PICM 35

PICM Overview 35
Compiler Settings 35
Writing Code Using PICM Functions 35
Memory Considerations and Requirements 36
Memory Allocation 36
Interface Limitations 36
- ISA (Industrial Standard Architecture) Interface 36
- PCI (Peripheral Components Interface) 37
- EISA (Enhanced Industrial Standard Architecture) Interface 37

Setting up the Registry and Device Drivers for Use with the Princeton Instruments EasyDlls for Windows 95 and Windows NT 37
- Location and name of device drivers 37
- How do they get loaded ? 37
- *.Reg files 39

Chapter 4 Controller Setup 41

Creating the Controller Object 41
Controller & Array Types 41
Application Type 44
Example 44
Destroying the Controller Object 44
Example 44
Important 45
Selecting A/D Rate 45
Auto-Stop 45

Chapter 5 Controller Parameters 47

Interface Card 47
Exposure Setting 48

Chapter 6 Imaging Option 49

Example 50

Chapter 7 Video Protocol 51

RS170 51
PICM_CMSetLongParam 51
Summary 51
Description 51
Example 51

Chapter 8 Collecting Data 53

Allocating Memory for Data 53
Initializing the Hardware 53
Performing Data Collection 53
Example 54
Direct Access to the DMA Buffers Linear Counterpart Per Frame 55
Synchronous vs. Asynchronous Acquisition 57
Synchronous 57
Asynchronous 57
Nframe vs. Focus Mode 57
Nframe Mode 57
Focus Mode 58
Working with Data Acquisition Events 58

Chapter 9 Utility Functions 61

Sensor Size 61
Pixel Dimensions 61
Reading and Writing TTL Signals 61

Chapter 10 Example Program 63

Simple Use of ST-138 With EEV 578x384 CCD..... 63

Chapter 11 Programming Reference 69

application_type..... 69
Prototype 69
Description 69
Example..... 69
Controller_type 69
Prototype 69
Description 70
Example..... 70
Data_Collection_Mode..... 70
Prototype 70
Description 70
Example..... 70
Detector_type..... 71
Prototype 71
Description 72
Example..... 73
Interface_card 73
Description 73
Example..... 73
PICM_ChkData 73
Prototype 73
Description 73
Example..... 74
PICM_CleanUp 74
Prototype 74
Description 74
Example..... 75
PICM_Clear_MultStrip 75
Prototype 75
Description 75
Example..... 75
PICM_Clear_user_rois 76
Prototype 76
Description 76
Example..... 76
PICM_CMGetDoubleParam..... 76
Prototype 76
Description 76
Example..... 76
PICM_CMGetLongParam 77
Prototype 77
Description 77
Example..... 77
PICM_CMSetDoubleParam 77

Prototype 77
Description 77
Example 77
PICM_CMSetLongParam 77
Prototype 77
Description 77
Example 77
PICM_CreateController 78
Prototype 78
Description 78
Error Codes 78
Example 79
PICM_CreateControllerNvram 79
Prototype 79
Description 79
Example 79
PICM_Download_MultROI 80
Prototype 80
Description 80
Example 80
PICM_Easy_Focusing 80
Prototype 80
Description 80
Example 81
PICM_FindController 81
Prototype 81
Description 81
Example 81
PICM_FindPCICards 82
Prototype 82
Description 82
Example 82
PICM_Generate_sideeffects 82
Prototype 82
Description 82
Example 82
PICM_Get_acqmode 83
Prototype 83
Description 83
Example 83
PICM__Get_Actual_Temperature 83
Prototype 83
Description 83
Example 83
PICM_Get_AutoStop 83
Prototype 83
Description 83
Example 83
PICM_Get_cleanscans 84
Prototype 84

Description 84
Example 84
PICM_Get_controller_version 84
Prototype 84
Description 84
Example 84
PICM_GetEnumParam 84
Prototype 84
Description 84
Example 85
PICM_GetEnumString 85
Prototype 85
Description 85
Example 85
PICM_Get_MinBlk 85
Prototype 85
Description 85
Example 85
PICM_Get_normal_rois 85
Prototype 85
Description 86
Example 86
PICM_Get_NumMinBlk 86
Prototype 86
Description 86
Example 86
PICM_Get_num_strips_per_clean 86
Prototype 86
Description 86
Example 86
PICM_Get_pixeldimension_x 86
PICM_Get_pixeldimension_y 86
Prototype 86
Description 87
Example 87
PICM_Get_RS170_enable 87
Prototype 87
Description 87
Example 87
PICM_Get_sensor_x 87
PICM_Get_sensor_y 87
Prototype 87
Description 88
Example 88
PICM_Get_shuttermode 88
Prototype 88
Description 88
Example 88
PICM_Get_shutter_type 88

Prototype 88
Description 88
Example 88
PICM_Get_Temperature 88
Prototype 88
Description 89
Example 89
PICM_Get_Temperature_Status 89
Prototype 89
Description 89
Example 89
PICM_Get_TTL_pattern 89
Prototype 89
Description 89
Example 89
PICM_Initialize_RS170 89
Prototype 89
Description 90
Example 90
PICM_Initialize_System 90
Prototype 90
Description 90
Error Codes 90
Example 90
PICM_IsAvail 91
Prototype 91
Description 91
Example 91
PICM_LoadNvramDefaults 95
Prototype 95
Description 95
Example 95
PICM_LockCurrentFrame 96
Prototype 96
Description 96
Example 96
PICM_ResetUserBuffer 96
Prototype 96
Description 96
Example 97
PICM_Set_acqmode 97
Prototype 97
Description 97
Example 97
PICM_Set_AutoStop 97
Prototype 97
Description 97
Example 97
PICM_Set_controller_version 97
Prototype 98

Description 98
Example 98
PICM_Set_cleanscans 98
Prototype 98
Description 98
Example 98
PICM_Set_EasyDLL_DC 98
Prototype 98
Description 98
Example 99
PICM_SetExposure 99
Prototype 99
Description 99
Example 99
PICM_Set_Fast_ADC 99
Prototype 99
Description 99
Example 99
PICM_SetInterfaceCard 100
Prototype 100
Description 100
Errors 100
Example 101
PICM_Set_MinBlk 101
Prototype 101
Description 101
Example 101
PICM_Set_MultiStrip_Flag 101
Prototype 101
Description 101
Example 101
PICM_SetNewUserBuffer 101
Prototype 101
Description 102
Example 102
PICM_Set_NumMinBlk 102
Prototype 102
Description 102
Example 102
PICM_Set_num_strips_per_clean 102
Prototype 102
Description 102
Example 102
PICM_SetROI 103
Prototype 103
Description 103
Example 103
Error Codes 103
PICM_SetROI_MultiStrip 104

Prototype 104
Description 104
Example 104
PICM_Set_RS170_enable 104
Prototype 104
Description 104
Example 104
PICM_Set_shutter 105
Prototype 105
Description 105
Example 105
PICM_Set_shuttermode 105
Prototype 105
Description 105
Example 105
PICM_Set_shutter_type 105
Prototype 105
Description 105
Example 105
PICM_Set_Slow_ADC 106
Prototype 106
Description 106
Example 106
PICM_Set_Temperature 106
Prototype 106
Description 106
Example 106
PICM_Set_TTL_pattern 106
Prototype 106
Description 106
Example 106
PICM_SetUserEvent 107
Description 107
Prototype 107
Example 107
PICM_SizeNeedToAllocate 107
Prototype 107
Description 108
Example 108
PICM_Start_Controller 108
Prototype 108
Description 108
Example 108
PICM_Stop_Controller 108
Prototype 108
Description 108
Example 109
PICM_UnlockCurrentFrame 110
Prototype 110
Description 110

Example 110

Chapter 12 Applications Strategies 111

Appendix A Headers and Calibration Structures 113

Saving Data To A WinView/WinSpec File 113
Version 1.43 Header 113
Version 1.6 Header 114
Calibration Structures 117
WINX Header Structure (with actual offsets) 118
Start of Data 121
Reading Strips and Frames of Data 122
Example File Write 123

Appendix B CCD Chips and Diode Arrays 127

CCDs 127
Diode Arrays 128

Appendix C Timing Modes vs. Controller Model 131

Introduction 131
PentaMAX 132
Introduction 132
Standard Timing Modes 132
Shutter Modes 132
Freerun timing 132
External Sync timing 132
MicroMAX 133
Standard Timing Modes 133
Shutter Modes 133
Freerun timing 134
External Sync timing 134
External Sync with Continuous Cleans Timing 134
ST138 135
Store Enable Option 135
Standard Triggering Modes 136
Shutter Modes 136
Freerun timing 136
External Sync timing 136
Continuous Cleans timing 137
ST130 138
Introduction 138
Shutter Options 138
Freerun 138
External Sync timing & Shutter Modes 138
ST12x 139

Introduction 139
Freerun 139
External Sync 139
Line Sync 140
External Trigger 140
External Sync and External Trigger 140
Line Sync and External Trigger 141
Event Counter 141
Store Enable Option 141
Acquisition Timing Mode Definitions 141
CTRL_FREERUN 141
CTRL_LINESYNC 142
CTRL_EXTSYNC_NORMAL 142
CTRL_EXTSYNC_PREOPEN 142
CRTL_EXTTRIG_NORMAL 143
CTRL_EXTTRIG_PREOPEN 143
CTRL_FR_STORE_TRIG 143
CTRL_SN_STORE_TRIG 144
CTRL_SP_STORE_TRIG 144
CTRL_TN_STORE_TRIG 144
CTRL_TP_STORE_TRIG 144
CTRL_EVENT_COUNTER 145
CTRL_EXT_SYNC_EXT_TRIG 145
CTRL_LINE_SYNC_STR_ENA 145

Index 147

Figures

Figure 1. Running the Setup program 22
Figure 2. Install Shield Setup Message 22
Figure 3. EasyDLL Setup Window 22
Figure 4. EasyDLL95 Welcome screen 23
Figure 5. Password Entry Screen 23
Figure 6. EasyDLL95 Unpacking Message 24
Figure 7. Install Shield Setup message 24
Figure 8. EasyDLL Setup Window and Welcome Screen 24
Figure 9. Software License Agreement Screen 24
Figure 10. User Information screen 25
Figure 11. Registration Confirmation screen 25
Figure 12. Choose Destination Location screen 26
Figure 13. Interface Type selection screen 26
Figure 14. Custom Installation Selection screen 27
Figure 15. Select Program Folder screen 27
Figure 16. Start Copying Files screen 28
Figure 17. Copying Files screen 28
Figure 18. Setup Complete screen 29
Figure 19. Quit dialog box 29

Figure 20. Your Computer dialog box 30
Figure 21. Control Panel dialog box 31
Figure 22. Add/Remove Programs Properties dialog box 31
Figure 23. Remove Confirmation dialog box 32
Figure 24. Regedit.exe (Windows 95 and NT) 38
Figure 25. PIVXDPCI.VXD Parameters for Windows 95 38
Figure 26. I_PCI.SYS Parameters for Windows NT 39
Figure 27. Store Enable timing diagram 135

This page intentionally left blank.

Chapter 1
General Information

Princeton Instruments General Command Library

Welcome to Princeton Instruments' General Command Library Version 1.6 (Windows 3.1) or Version 2.0 (Windows '95). This command library, known as PICM for Princeton Instruments Controller Module, is a set of C library and header files that have been designed for programmers using Microsoft C /C++ version 1.52 or greater for Windows 3.1 or Windows for Workgroups or using Microsoft Visual C/C++ ver 4.1 for Windows 95. There are some functions that will not work correctly if used in Borland C/C++. For example, in Windows 3.1, functions that return a double will not work. In Windows 95, if using Borland C/C++, DLLs must be loaded dynamically (not statically linked). Alternatively, you can use Borland's Implib on the DLL to create a LIB you can statically link to.

PICM provides users with the ability to collect single-frame data from Princeton Instruments CCD detectors through the PIXCM.DLL (Princeton Instruments External Controller Module Dynamic Link Library).

As a high-level function set, PICM allows you to create custom data collection applications quickly and easily.

Functionality

Although this library has been designed to be easy-to-use, it still provides a comprehensive function list. The following is a brief list of the features available to a programmer using the DLL.

- Open and close communications to Princeton Instruments controllers.
- Initiate data collection and retrieve a single frame of data using the "free run" timing mode..
- Write and read through the controller's TTL port.

Other, more advanced operations, such as region of interest and multiple frame collection, are possible through extensions to the library. These extensions are described in the appendices of this manual.

Using PICM

Basic Operation

In a Princeton Instruments system, the CCD controllers are designed to run continuously from when a software “start” command is issued to when the software commands the system to stop. To facilitate data collection on a rigorous, real time schedule, all operations during this time are handled by the hardware and low level software. This ensures that all time-critical aspects of an experiment are handled properly regardless of the real time performance of your software, making dark current reproducible and thus subtractable.

With this in mind, the sequence of operations which need to be done to acquire an image are as follows:

- Define the controller and CCD chip. Also set the interface that will be used. This defines how the computer will talk to the controller.
- Set experiment conditions such as Exposure and ROI.
- Initialize the controller. This involves both initializing the data structures that the library operates from and initializing the hardware of the controller itself. Both of these can be done without actually starting the controller, so the controller can be completely ready to begin operating the detector as soon as the separate “start” command is given, improving the response time in real time systems.
- Allocate memory for data to be collected into.
- Setup the Windows 3.1 system aspects of the software. (i.e. create a window for the application, set up user controls, etc.)
- Instruct the controller to start.
- Wait for data collection to be completed. This can be done asynchronously by polling PICM for the status of data collection. This can be done continuously or occasionally, allowing the software to perform processing while new data is being collected, improving overall system throughput.
- Instruct the controller to stop.
- Process the collected data. (perform calculations, save to disk, etc.)
- Release the software resources allocated at the beginning of execution. (i.e. memory for both the controller parameter structures and data memory, as well as MS-Windows resources used for the user interface)

Windows 3.1 Example Programs

PICM is shipped with a comprehensive example program that you can compile and run. It is recommended that you begin by working with the example program and then modify it to suit your specific system requirements. Complete examples are provided for the compiler that PICM has been tested on, Microsoft Visual C/C++ (ver 1.52). The library may be compatible with other compilers. However, only Microsoft Visual C/C++ (ver1.52) has been tested.

In addition, the following programs written in simple C are provided, together with one program (Imgx) written in C++. Each of these examples illustrates an action, but they should not be copied verbatim into the user's software.

General: Collects one frame (all controllers)

Imaging: Collects one ROI (sub-frame). Applies to all CCDs but not to diode arrays.

Multfram: There are three examples for collecting multiple frames. These examples are limited to the ST138 with auto-stop, the MicroMAX (ST133), and the PentaMAX (DC131) controllers. The three example programs are:

mltcomex: Shows how to reuse the user pointer/buffer.
mlt2comx: Shows how to use multiple user buffers.
mltaccum: Shows how to do software accumulation (adding frames).

Multstrp: Shows how to do multiple strips (ROIs).

Kinetics: Shows how to use the ST138 (only) in kinetics mode. (Note that not all of the ST138 hardware supports the kinetics mode).

Focus: Shows how to use the MicroMAX focus (RS170) mode.

Focuspen: Shows how to use the RS170 and hardware look-up table on the PentaMAX camera.

Imgx: C++ example showing how to display an image.

Windows 95/NT Example Programs

In addition to the above example programs, the following example programs for Windows 95 and Windows NT are included with the EasyDLL's.

Multcont: Shows how to setup and collect data form multiple controllers.

SplitROI: Shows how to collect multiple regions of interest.

Using This Manual

If you are about to use PICM for the first time, it would be a good idea to read through the first section of this manual which explains the basic concepts and techniques, including examples, used for implementing the DLL.

After you have become familiar with the DLLs function calls, you'll find the Programming Reference section extremely useful for looking up the exact descriptions and formats of the library's functions and variables.

Note: This manual assumes that you are already familiar with the programming and use of applications in Microsoft windows. Additionally, it is expected that you already understand the basic architecture, components and operation of the Princeton Instruments' products that you intend to operate with this command library.

This page intentionally left blank.

Chapter 2
Installation

Installing PICM for Windows 3.1

To install, insert the installation Disk and select “Run” from the Program Manager’s File menu. Type “A:SETUP” into the run dialog box. (Or, the appropriate letter of the floppy drive.) Follow the instructions to install the PICM files into your system.

The install program will ask you to select from several options:

Typical	These are the files needed to write programs using PICM. This will also install the example programs and makefiles.
Compact	This is the minimum set of files that must be installed in order to write programs using PICM.
Custom	This allows you to choose the files you want installed on your system.

PICM Files for Windows 3.1

platform.h	Platform specific code (i.e. SUN, Mac, Win95, Win NT).
pixcm.dll	Main DLL that user links into his system.
pixcmtxt.dll	Used by pixcm.dll
pi133.dll	Used by pixcm.dll
pixcm.lib	lib to link into user project
pi133.dat	chip information used by MicroMAX, must be in executables path
piadcdef.h piadcfcn.h	functions to override default ADC settings.
pifcsdef.h pifcsfcn.h	MicroMAX (ST133) focus example
pigendef.h pigenfcn.h	General defines, need for all examples, see general example
piimgdef.h piimgfcn.h	ROI function, see imaging example.
pikindef.h pikinfcn.h	Kinetics functions and defines, see kinetics example

pimandef.h pimanfcn.h	OEMs only, Not supported in easy dll manual override of system.
pimltfcn.h	Multiple frame collection, supported only for ST138s with autostop, MicroMAX, and PentaMAX.
pishtdef.h pishtfcn.h	Shutter override functions
piskpfcn.h	functions to override default cleans and skips.
pistpfcn.h	Multiple strip, V1.4.3 supports ST130 & ST138 V1.6 supports MicroMAX (ST133) & PentaMAX (DC131).
pitimdef.h pitimfcn.h	timing mode functions (i.e. ext. sync.)
piverdef.h piverfcn.h	DLL version functions.

Installing PICM for Windows/95

After confirming that your computer meets all hardware and software requirements, install PICM for Windows/95 according to the instructions that follow. Also, it is recommended that you install the Princeton Instruments interface board in your computer before running the software. Instructions for installing the interface board are provided in your Princeton Instruments hardware manuals.

Note: If your computer and system were purchased together from Princeton Instruments, the Interface card will have been installed at the factory.

System Requirements

The following information lists the system hardware and software requirements.

Hardware requirements

- Princeton Instruments ST series controller and Camera system.
- Princeton Instruments high speed PCI serial card (standard) or ISA serial card. *Computers purchased from Roper Scientific are shipped with the card installed.*
- Minimum of 32 Mbyte total RAM for CCDs up to 1.4 million pixels. Collecting larger images at full frame or at high speed (such as with the Penta*MAX* camera) may require 128 Mbytes or more of RAM.
- Hard disk with a minimum of 80 Mbytes available. A complete installation of the program files takes about 6 Mbytes, and the remainder is required for data storage. Collection of large images may require additional hard disk storage, depending on the number of images collected and their size. Disk level compression programs are not recommended.

- Minimum is AT compatible computer with 80486 (or higher) processor (50 MHz or faster); Pentium or better recommended.

  Note: Not all computers are able to satisfy the software and data-transfer performance requirements of Princeton Instruments systems. If you purchased a computer through Roper Scientific, it will have already been tested for proper operation with a Princeton Instruments system and will have the Interface card installed.

- Super VGA monitor and graphics card supporting at least 256 colors with at least 1 Mbyte of memory. Memory required will depend on desired display resolution.
- Two-button Microsoft compatible serial mouse or Logitech three-button serial/bus mouse.

Operating System requirements

Windows 95 (or higher) or Windows NT (ver 4.0 or higher). PICM for Windows/95 is ***not*** supported under OS/2. Nor will it run under Windows 3.1 or 3.11.

Installation Procedure

EasyDLL95 must be installed under Windows. It cannot be installed on a computer only running DOS, or from the DOS shell.

EasyDLL95 is normally distributed either on floppy disks, or by executing the installation program provided on Roper Scientific's FTP site.

Note: When EasyDLL95 is installed, it modifies the Windows Registry . If for any reason you reinstall Windows, the Registry may be replaced, and EasyDLL95 may not run correctly. Reinstall EasyDLL95 to correct this problem.

Note: If installing under Windows NT, you must be logged on as administrator of the NT Workstation. Certain changes are made to the Registry during the installation. If you are not logged on as the administrator, the Registry changes cannot be made and the installation will fail. Note that the failure doesn't occur until the installation process is almost complete.

Following are the steps to install EasyDLL95 on your computer.

- Exit any software currently running. This will speed the installation.

Installation from disk

- Insert the EasyDLL95 floppy disk #1 into your computer's floppy drive.
- Click the desktop **Start** button, select **Run**, key **x:\Setup** (where "x" is the letter designating your installation drive as shown in Figure 1) and press the **Enter** key on your keyboard. The install sequence will begin, the Install Shield setup will run (Figure 2) and the EasyDLL95 Setup Welcome screen will appear (Figure 3).
- Proceed to **Steps Common to all Setups**.

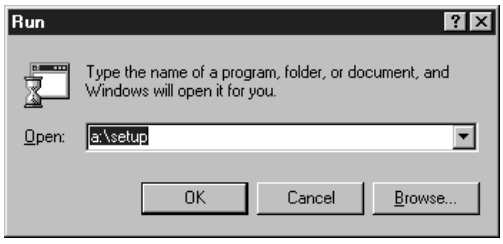


Figure 1. Running the Setup program

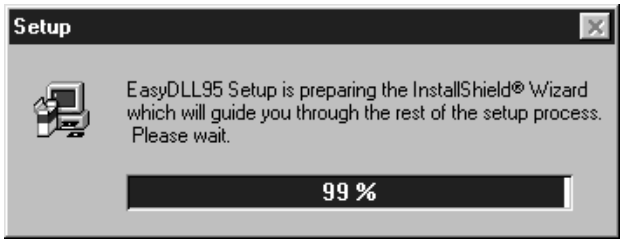


Figure 2. Install Shield Setup Message

Figure 3. EasyDLL Setup Window

Installation from FTP Site

- Log onto the FTP site.
- If installing the Windows 95 version of the software, execute the program EasyDLL95.exe. If installing the Windows NT version of the software, execute the program EasyDLLNT.exe. Executing either program will cause the EasyDLL Welcome screen to appear as shown in Figure 4.

Note: If you aren't sure how to access the FTP site, contact Roper Scientific's Technical Support Department for assistance. Contact information follows.

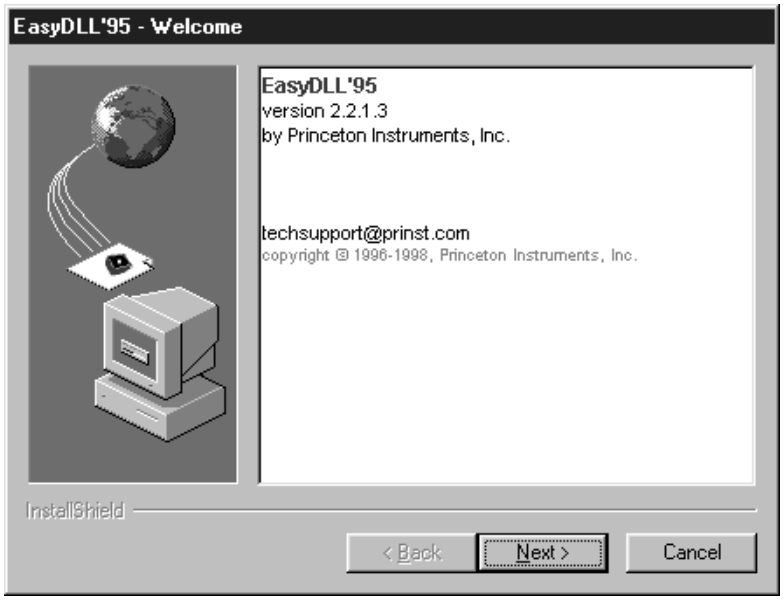
Roper Scientific
3660 Quakerbridge Road
Trenton, NJ 08619 (USA)

Tel: 609-587-9797
Fax: 609-587-1970

Tech Support E-mail: techsupport@roperscientific.com

For technical support and service outside the United States, see our web page at www.roperscientific.com. An up-to-date list of addresses, telephone numbers, and e-mail addresses of Roper Scientific's overseas offices and representatives is maintained on the web page.

Figure 4. EasyDLL95 Welcome screen



- Click **Next** to continue. You will then be asked to supply the required password, as shown in Figure 5. Initially the **Next** button will be grayed out. When the correct password has been entered, the button will become selectable and you will be able to continue.

Figure 5. Password Entry Screen

- Click on **Next**. EasyDLL95 will unpack (Figure 6), the Install Shield setup (Figure 7) will run and the EasyDLL95 Setup Welcome screen will appear (Figure 8).

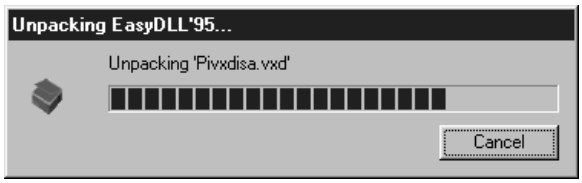


Figure 6. EasyDLL95 Unpacking Message

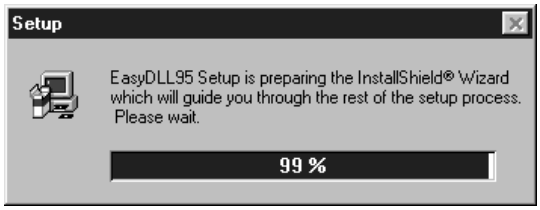


Figure 7. Install Shield Setup message

Figure 8. EasyDLL Setup Window and Welcome Screen

Steps Common to all Setups

- Click on **Next** to continue. The Software License Agreement will appear as shown in Figure 9.

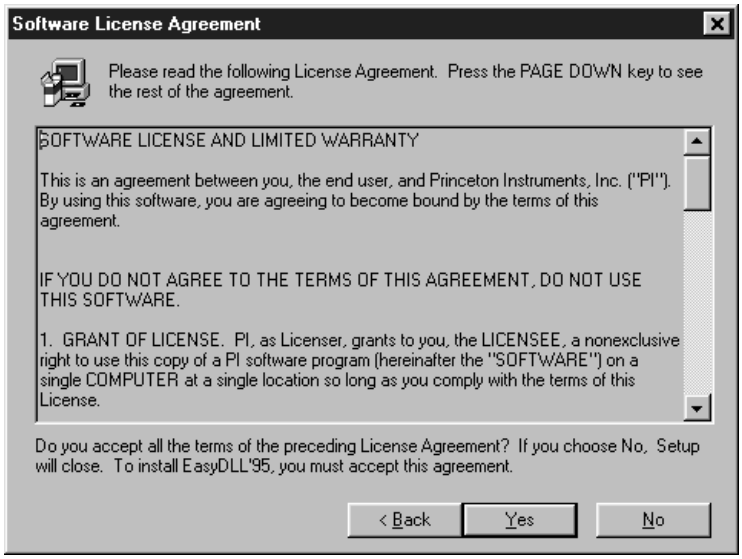
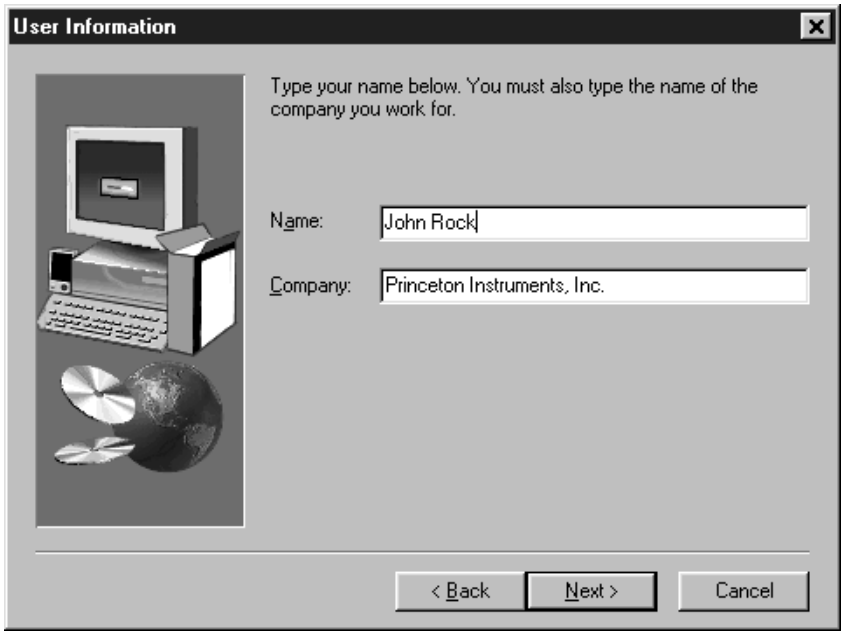


Figure 9. Software License Agreement Screen

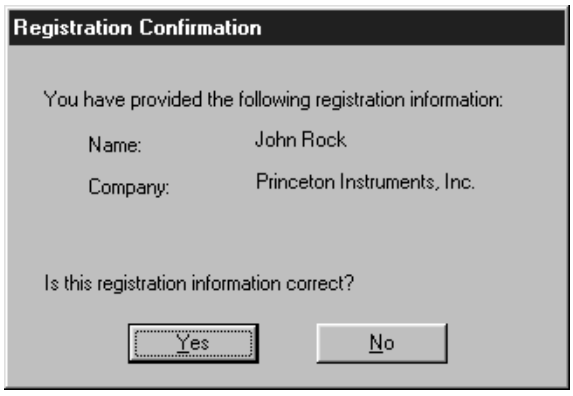
- If you accept the License Agreement (necessary to complete the installation), click on **Yes** to continue. The User Information screen will open as shown in Figure 10.

Figure 10. User Information screen



- Enter the required information and click **Next** to continue. The Registration Confirmation screen (Figure 11) will appear.

Figure 11. Registration Confirmation screen



- If the information is correct, click on **Yes** to continue. The Choose Destination Location screen will open.

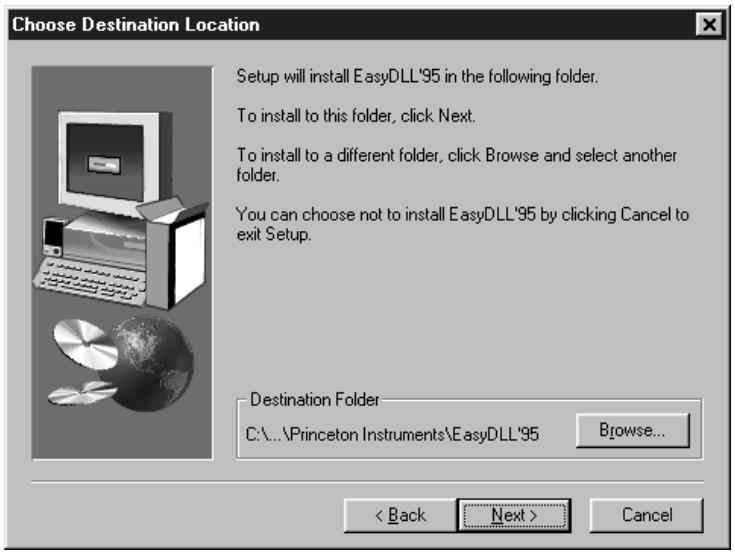


Figure 12. Choose Destination Location screen

- If the default destination is acceptable, click **Next** to continue. If some other destination is preferable, click on Browse to open a Win/95 browser and designate the desired destination. On returning to the Choose Destination Location screen, click on **Next** to continue. The Interface Type screen will open as shown in Figure 13.

Figure 13. Interface Type selection screen

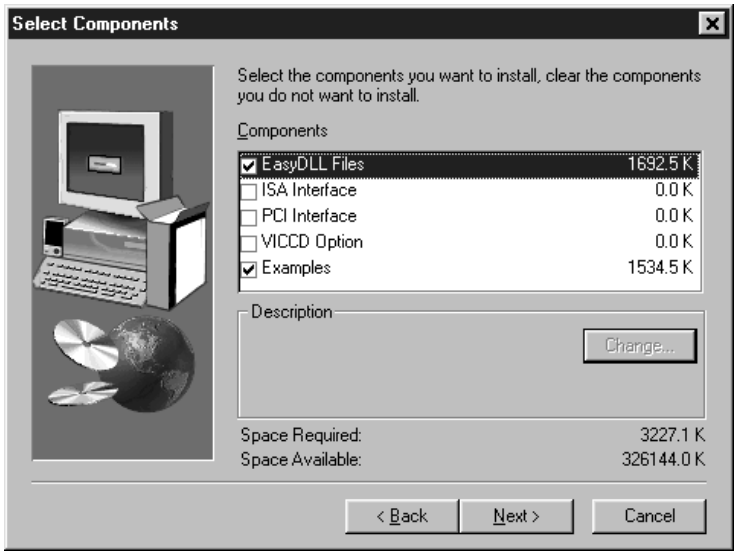
- Select the installation type you prefer. A brief description of each of the available types follows.

  Typical: This is the best choice for most users. It loads the executables, the PCI Interface Card driver, and the example files.

  Compact: Loads the executables only. Allows data to be taken but there is no provision for transferring the acquired data to the computer.

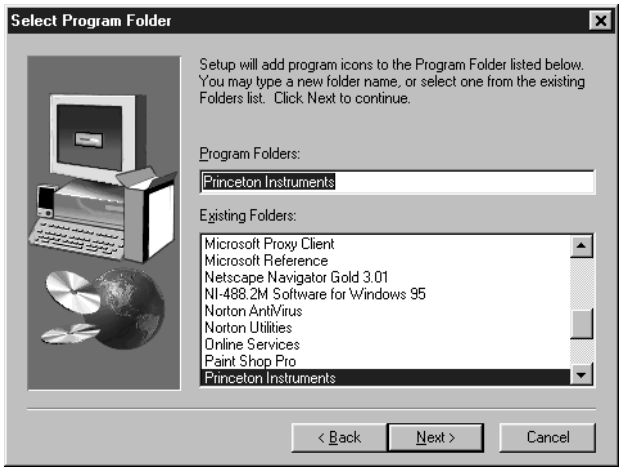
Custom: Allows the user to select the components to be installed. Figure 14 shows the available choices. Note that the Custom Installation ***must*** be used to select the ISA Interface or V/ICCD Option. Users are advised ***not*** to install support both ISA and PCI support, but rather only the one that is installed in the interfacing computer. The interface support files each reserve a large block of memory at bootup. By selecting only one interface, memory allocation will be optimized.

Figure 14. Custom Installation Selection screen



- After making your selections, click on **Next** to continue. The Select Program Folder screen will open as shown in Figure 15.

Figure 15. Select Program Folder screen



- Select the Program Folder you prefer or enter a new one. Then click on **Next** to continue. The Start Copying Files screen (Figure 16) will open.

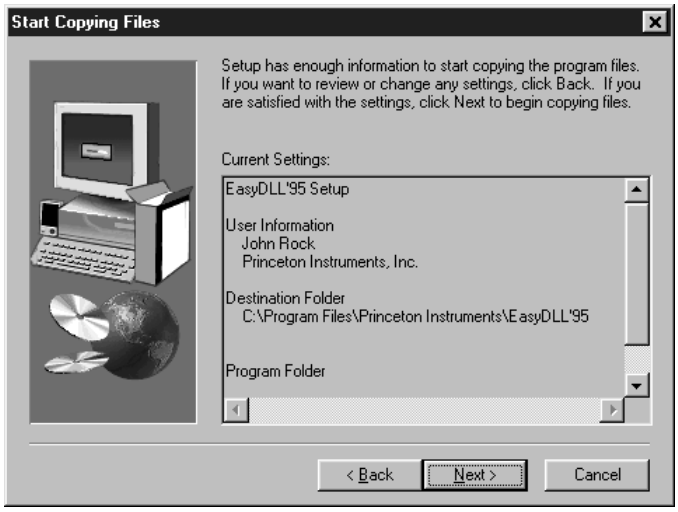


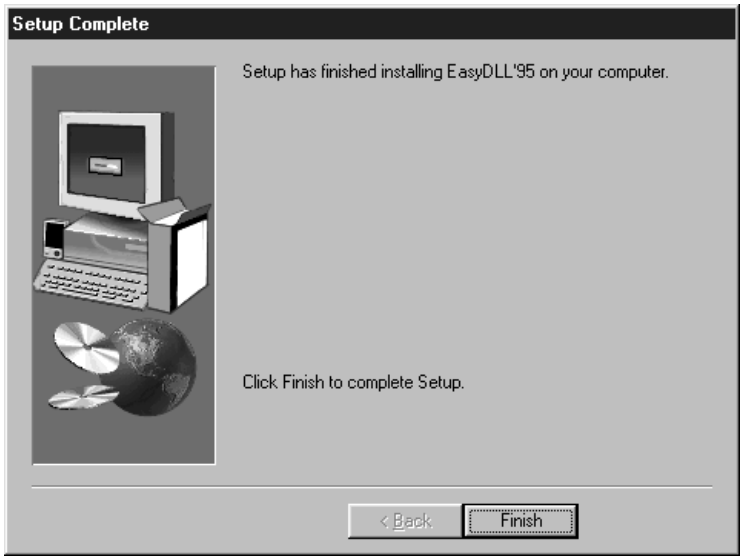
Figure 16. Start Copying Files screen

- If satisfied with the displayed settings, click on **Next** to continue. Actual installation of the EasyDLL95 files will commence and the Copying Files screen will be displayed as shown in Figure 17.

Figure 17. Copying Files screen

- Once all of the files have been installed, the Setup Complete screen will appear as shown in Figure 18.

Figure 18. Setup Complete screen

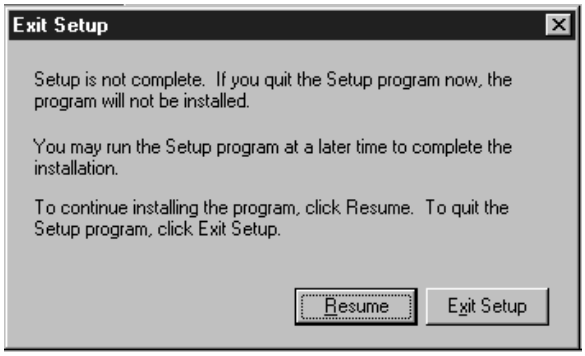


- Click on **Finish** to complete the setup and close the installation program.

Aborting the EasyDLL95 Installation

- Clicking on the Cancel button anytime during installation will cause the dialog box in Figure 19 to appear.

Figure 19. Quit dialog box



- Click on the **Exit Setup** button to cancel the installation of the software. No part of the software will be installed. Click on the **Resume** button to return to the installation at the point where Cancel was selected so that installation can be completed.

Installing Some EasyDLL95 Files at a Later Time

You can install some EasyDLL95 files at first, and other files at a later time. Simply repeat the installation procedure, taking care to select Custom as the installation type. Then select the file type(s) to be installed and proceed as previously described.

Installing More than One Version of EasyDLL95

You can install more than one version of EasyDLL95 on a single computer. In the Custom Installation dialog box simply change all the paths listed to a new directory such as C:\EasyDLL32B. The install program will automatically create the new directory, if necessary.

It is also possible to install both 16 bit and 32 bit versions of the software in the same computer. However, keep in mind that EasyDLL95 will not operate under Windows 3.1 or 3.11. Similarly, the device drivers for the 16 bit version of EasyDLL will not function properly under Win 95 or NT.

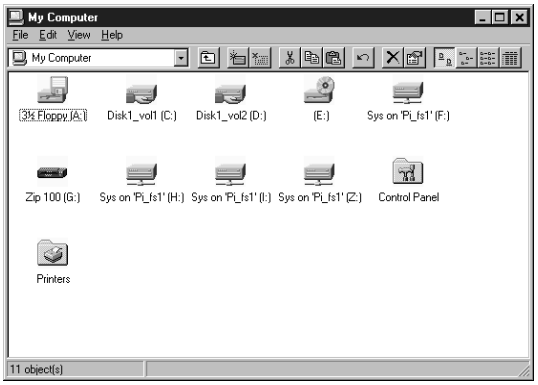
Uninstalling and Reinstalling

If you suspect any of the EasyDLL95 files have become corrupt, you should first delete all EasyDLL95 files, then reinstall the software from the FTP site or from the original floppy disks. Follow the steps below to remove all traces of the WinView/32 software. Then reinstall the software.

WinView/32 includes provision for automatically deinstalling the software. To deinstall WinView/32 from your computer, proceed as follows.

- From the Windows 95 desktop, click on My Computer. This will bring up the dialog box shown in Figure 20.

Figure 20. Your Computer dialog box



- Double click on Control Panel to bring up the Control Panel dialog box (Figure 21).

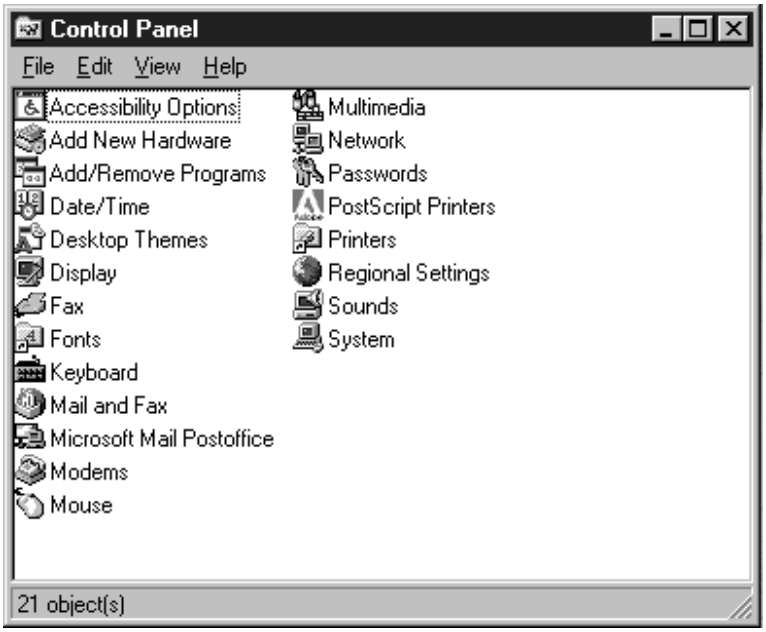


Figure 21. Control Panel dialog box

- Double click on Add/Remove Programs to bring up the Add/Remove Programs Properties dialog box (Figure 22).
- Select EasyDLL95 and click on the Add/Remove button. The Remove Confirmation dialog box (Figure 23) will appear.
- Click on Yes and EasyDLL95 will be removed. The Remove Confirmation dialog box will then disappear.
- On the Add/Remove Programs Properties dialog box (Figure 22), click on OK to complete the process.
- Close the open windows to return to the "clean" desktop.

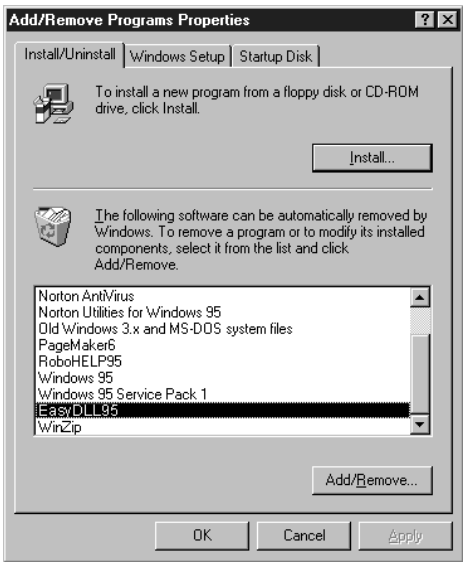


Figure 22. Add/Remove Programs Properties dialog box

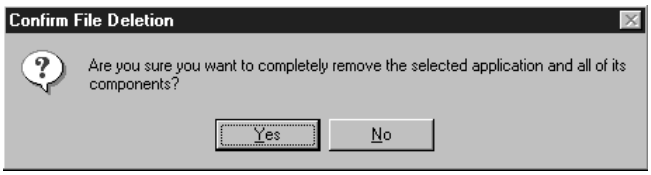


Figure 23. Remove Confirmation dialog box

To later reinstall the software, follow the Installing EasyDLL95 instructions provided earlier in this chapter. Use the original installation disks or install using the installation program on the FTP site.

PICM Files for Windows 95

platform.h	Platform specific code (i.e. SUN, Mac, Win95, Win NT).
pixcm32.dll	Main DLL that user links into his system.
pixcmtxt32.dll	Used by pixcm32.dll
pi13332.dll	Used by pixcm32.dll
pidc32.dll	Princeton Instruments data collection
pipp32.dll	Talks to hardware, Princeton Instruments physical port
piscc32.dll	Used by pixcm32.dll
pixcm32.lib	lib to link into user project
pi133.dat pi133b.dat pi1335b.dat	chip information used by MicroMAX, must be in executables path
piadcdef.h piadcfcn.h	functions to override default ADC settings.
pifcsdef.h pifcsfcn.h	RS170 functions and defines.
pigendef.h pigenfcn.h	General defines, need for all examples, see general example
piimgdef.h piimgfcn.h	ROI function, see imaging example.
pikindef.h pikinfcn.h	Kinetics functions and defines, see kinetics example
pimandef.h pimanfcn.h	OEMs only! Not supported in EASY DLL manual override of system.
pimltfcn.h	Multiple frame collection, supported only for ST138s with autostop, MicroMAX, and PentaMAX.
pishtdef.h pishtfcn.h	Shutter override functions
piskpfcn.h	functions to override default cleans and skips.

pistpfcn.h	Multiple strip, V1.4.3 supports ST130 & ST138 V1.6 supports MicroMAX (ST133) & PentaMAX (DC131).
pitimdef.h pitimfcn.h	timing mode functions (i.e. ext. sync.)
piverdef.h piverfcn.h	DLL version functions.

Chapter 3
Using PICM

PICM Overview

The PIXCM.DLL accepts simple function calls from your own C program and responds by transmitting detailed instructions to an Princeton Instruments CCD controller connected to your computer.

By handling all of the complex communications and setup functions, the PIXCM DLL allows you to begin writing and using your own controller applications quickly and easily.

Compiler Settings

In most ways, a program with PIXCM DLL calls may be compiled just like any normal, stand-alone Windows application. However, a program using PICM most be compiled using the large model. Makefiles for compiling the example programs with Microsoft Visual C/C++ v1.52 are included and may be used as models for your own programs.

Writing Code Using PICM Functions

To use PICM functions, a C file must contain `#define PIXCM 1`, `#include "Platform.h,"` `#include "PIGENFCN.H"` and `#include "PIGENDEF.H"` statements and be linked with the PIXCM.LIB. PICM functions are named with the standard naming convention: PICM_*OperationObject.* (Example: `PICM_SetExposure`).

If you are writing a program using C++, PICM header files must be declared as C files:

```
extern "c" {
#define PIXCM 1
#include "platform.h"
#include "PIGENDEF.H"
#include "PIGENFCN.H"
}
```

All PICM functions return an integer value which will be set to zero (FALSE), whenever an error occurs. The reason for the error message can be determined by checking the `error_code` variable that is passed as the last parameter for the PICM function.

Memory Considerations and Requirements

We recommend a minimum of 32 MegaBytes of RAM in the system that will run our controllers. This will enable our device driver to allocate a buffer of at least 8 - megabytes. Windows NT users can expect to be able to allocate somewhat less than Windows 95 users because the operating system takes up more RAM (about 4 Megs more). However you don't need an 8 Meg buffer for all applications. This amount depends on two things, ROI being acquired and mode of operation. At a minimum, one frame size must be allocated. However, this can not support synchronous collection because of its nature. If your application is going to acquire multiple frames via nframe mode (synchronously), you will not get a data overrun only if the buffer is large enough for the number of frames. The number of frames can be greater than the number of the frames the buffer can hold if you pull them out quickly, but this is a race condition depending on the data readout rate and the CPU speed. The controller most likely to have problems in this area is the PentaMax running at 5 MHz. If you are running asynchronously a single frame at a time, you only need to allocate one frame size in the buffer. The amount of memory to be allocated is lost to the operating system and can only be used by the Princeton Instruments Libraries and OEMs.

Memory Allocation

The PICM library was written using object oriented concepts. The primary interface between the PIXCM.DLL and your software is a software controller "object". which must be allocated with `PICM_CreateController` before any other PICM functions can be used. During the execution of your program, this object will be allocating its own blocks of memory and, therefore, must later be cleared from memory by using `PICM_CleanUp ()`before exiting your program. Using the clean up command will ensure that all memory resources allocated for controller communications are properly released.

It is important to note that configuration functions such as `PICM_SetROI` operate on this memory object. The parameters stored in the software controller are transmitted to the hardware controller when `PICM_Initialize_System` is called.

The memory allocated for the controller object itself is separate from memory to be used for storing data collected from the controller. Data memory must be allocated explicitly by your program. This is discussed in Chapter 6.

Interface Limitations

ISA (Industrial Standard Architecture) Interface

Due to the physical hardware limitations of the DMA controller in the computer, memory for the ISA interface card is limited to the first 16 megabytes. This means that the amount of memory allowable for the ISA driver is restricted to less then 16 megabytes. Also, because it is not a bus mastering device and uses the chip in the computer, operation with multiple controllers using the ISA interface is not supported. Currently we are encountering problems using ISA interface cards in EISA computers under Windows 95. We hope to attain a solution as soon as possible. There is no support

for ISA cards under Windows NT because development of the device driver is not yet complete.

PCI (Peripheral Components Interface)

There are few limitations to the PCI interface card. We support Windows 95 and Windows NT. The amount of memory the card can use for transfers is dependent solely on the amount of memory in the system. We have tested 4k by 4k chips (32 Megs per image) under Windows 95 and windows NT. At this time we support multiple controllers for PCI under Windows 95 only, and have tested this with four PentaMaxes synchronously at 5 MHz with 0 exposure (40 Megabytes a Second sustained). It was also tested with four ST-133s.

EISA (Enhanced Industrial Standard Architecture) Interface

There is currently no support for this interface under Windows 95 and Windows NT; the laws of supply and demand are in effect.

Setting up the Registry and Device Drivers for Use with the Princeton Instruments EasyDlls for Windows 95 and Windows NT

Location and name of device drivers

Windows 95

PIVXDPCI.VXD is the Princeton Instruments PCI device driver. It belongs in your Win95\System directory.

PIVXDISA.VXD is the Princeton Instruments ISA device driver. It belongs in your Win95\System directory.

Windows NT

PI_PCI.SYS is the Princeton Instruments PCI device driver its location should be in your WinNT\System32\drivers directory.

How do they get loaded ?

All the drivers for Windows 95 and NT will be loaded via the registry. In general, users will not and should probably not need to edit the registry. The registry for NT and 95 are basically the same with some slight differences. With Windows 95 and NT distributions comes a program in the windows directory entitled regedit.exe, which enables us to look at and alter some of our device drivers parameters. These parameters can also be altered by double clicking on a *.reg file. A *.reg file is a file that contains variable positions, names, and values that will be added to the registry when it is double clicked from Explorer or a similar application. The *.reg files will be discussed in greater detail later in this document.

A. The Windows 95 Registry (through regedit.exe)

When you run regedit.exe it should look similar to the screen capture shown in Figure 24. To get to the Princeton Instruments information you must go through the registry down the path by double clicking on folders:

HKEY_LOCAL_MACHINE
System
CurrentControlSet
Services
VxD
PIVXDPCI or PIVXDISA

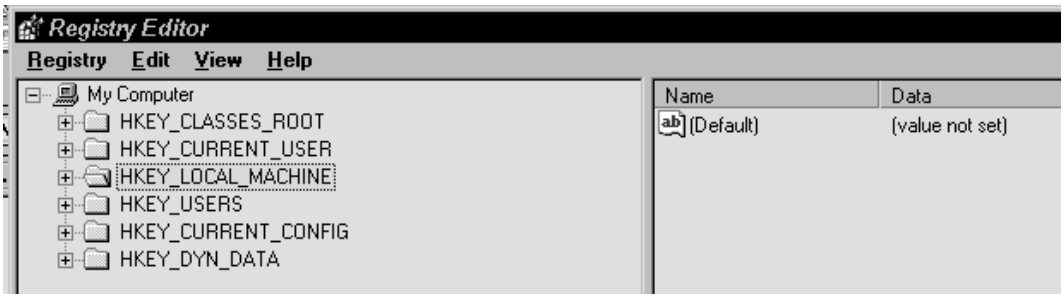


Figure 24. Regedit.exe (Windows 95 and NT)

Once you get this far, if you open the folder PIVXDPCI or PIVXDISA, you will see the parameters for our driver (Figure 25). These can be modified by clicking the right mouse button on the small icon in the right side of regedit. However, when entering values, make sure you note whether decimal or hex is selected because of the huge discrepancy from hex to decimal.

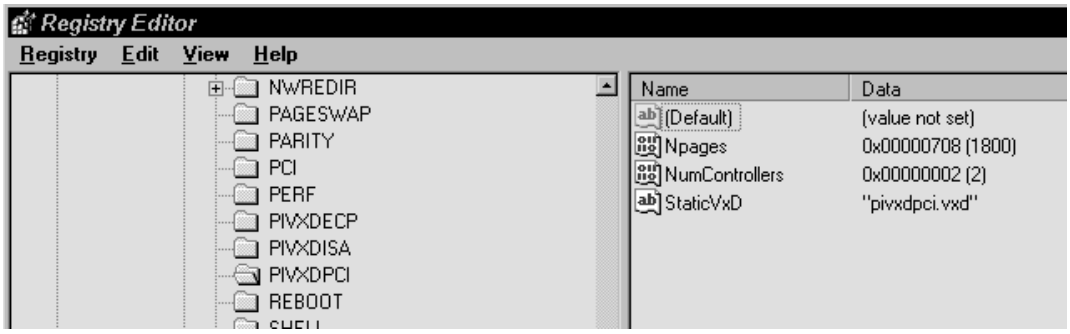


Figure 25. PIVXDPCI.VXD Parameters for Windows 95

Parameters

Default: Not Used

Npages: Number of 4096 Byte Pages for DMA Buffer.

NumControllers: Number of Controllers going to be running simultaneously (PCI only).

StaticVxD: Name of the driver to be loaded. This is what forces the system to load the driver.

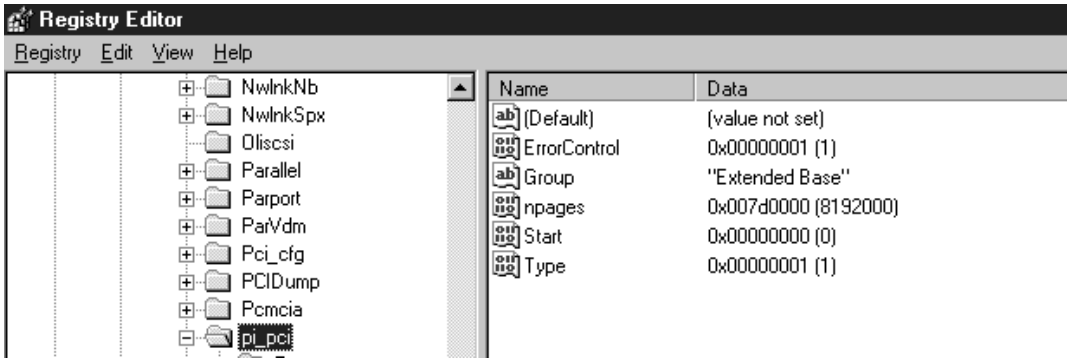
B. The Windows NT Registry (through regedit.exe)

When you run regedit.exe it should look similar to the screen capture shown in Figure 24. To get to the Princeton Instruments information you must go through the registry down the path by double clicking on folders:

```
HKEY_LOCAL_MACHINE
    System
        CurrentControlSet
            Services
                PI_PCI
```

Once you get this far, if you open the folder PI_PCI, you will see the parameters for our driver (Figure 26). These can be modified by clicking the right mouse button on the small icon in the right side of regedit. However, when entering values, make sure you note whether decimal or hex is selected because of the huge discrepancy from hex to decimal.

Figure 26. I_PCI.SYS Parameters for Windows NT



Parameters

Default:	Not Used
Npages:	Number of 4096 Byte Pages for DMA Buffer.
NumControllers:	Number of Controllers to be run simultaneously (Not Implemented Yet for Windows NT).
Start:	Should be zero. Signifies start driver at boot. This forces driver to load.
Group:	Should be “Extended Base”
Type:	Should be 1
ErrorControl:	Should be 1

*.Reg files

A reg file is one that will allow the user to double click on it from just about anywhere. When this is done, parameters from the file will be loaded into the registry. An example of a typical reg file follows.

```
Start File
REGEDIT4

[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\PIVXDPCI]
"Npages"=dword:000007d0
"NumControllers"=dword:00000001
"StaticVxD"="pivxdpci.vxd"
End File
```

This is the reg file for our Windows 95 PCI driver. First is the version, REGEDIT4. Then the path will add the key PIVXDPCI if it isn't there, and finally the parameters. If you went into the registry and entered all of the parameters you see in the Figure 25 by hand it would have the same effect as double clicking on one of these files. There should be reg files for our device drivers installed by the Easydll 95 or NT installation procedure.

Chapter 4
Controller Setup

Creating the Controller Object

`PICM_CreateController ()` must be the first PICM function called by your software. This is used to allocate a software controller object in your computer's memory to be used by all other PICM functions. The following is a function prototype for `PICM_CreateController`:

```
int _export FAR PASCAL PICM_CreateController
(
 int Controller_type,       /* controller to create, see ctrl_type          */
 int Detector_type,         /* CCD/PDA to run, see ctrl_CCD_sensor          */
 int Data_Collection_Mode,  /* Data collection mode, see sensorReadoutMode  */
 int application_type,      /* type of application, see Application_Type    */
 unsigned int *error_code   /* Error code; used if function returns false.  */
 );
```

Controller & Array Types

The `Controller_type` integer describes the controller you are using and may be any one of the following: `ST130, ST138, DC131, ST121, ST133, ST133.5MHZ.`

Note: Not all features, such as video output on the DC131 (PentaMAX) and ST-133 controllers, are supported by the PICM library.

`Detector_type` should correspond to the detector array to be used.

Note: Always refer to pigendef.h to determine the latest chips supported.

The array setting is independent of which detector housing is being used, such as liquid nitrogen, thermal electric or air cooled styles. The following values are legal:

```
                    /*             DLL/WinView        Y     X      */
CUSTOM_CCD = -1,    /*  Custom User defined CCD chip               */
      /*** CUSTOM_CCD AS A CHIP TYPE IS NO LONGER USED !!!! ***/
NO_CCD_SENSOR,      /*        PN#  CCD                X     Y      */
EEV_256x1024_3ph,   /* EEV 256x1024 3-phase          256  1024    */
EEV_576x384_3ph,    /* EEV 576x384 3-phase           576   384    */
EEV_1152x298_3ph,   /* EEV 1152x298 3-phase         1152   298    */
EEV_1152x1242_3ph,  /* EEV 1152x1242                1152  1242    */
KDK_512x768,        /* KODAK 512x768                 512   768    */
KDK_1035x1317,      /* KODAK 1035x1317              1035  1317    */
KDK_1024x1280,      /* KODAK 1024x1280              1024  1280    */
KDK_2044x2033,      /* KODAK 2044x2033              2044  2033    */
KDK_2048x3072,      /* KODAK 2048x3072              2048  3072    */
                    /* Fast Kodak special, will be removed in future */
KODAKFAST1400,      /* Use custom chip              1035  1317    */
                    /* Overscan special, will be removed in future */
OSTK1024B,          /* Use custom chip              1050  1050    */
```

```
PID_330x1100_8phH,     /* PI 330x1100 8 phase (horz)     330  1100    */
PID_532x1752,          /* PI 532x1752                    532  1752    */
RET_400x1200,          /* RET 400x1200                   400  1200    */
RET_512x512,           /* RET 512x512                    512   512    */
NOT_USED2,             /* old EEV 576x384  (not used)    576   384    */
RET_1024x1024,         /* RET 1K x 1K                   1024  1024    */
RET_2048x2048,         /* RET 2K x 2K                   2048  2048    */
TEK_512x512_B_100ns,   /* TEK512x512B Back  [100ns]      512   512    */
TEK_512x512_F_100ns,   /* TEK512x512F Front [100ns]      512   512    */
TEK_1024x1024_B_100ns, /* TEK1024x1024B Back [100ns]    1024  1024    */
TEK_1024x1024_F_100ns, /* TEK1024x1024F Front[100ns]    1024  1024    */
TEK_2048x2048,         /* TEK 2K x 2K                   2048  2048    */
THM_576x384,           /* TH576x384                      576   384    */
EEV_256x1024_6ph,      /* EEV 256x1024  6 PHASE          256  1024    */
EEV_1024x512_FT,       /* EEV frame transfer            1024   512    */
NOT_USED3,             /* Use custom chip                512  1024    */
NOT_USED4,             /* Use custom chip                512  1024    */
EEV_576x384_6ph,       /* EEV576x384   6 PHASE           576   384    */
EEV_1152x298_6ph,      /* EEV1152x298  6 PHASE          1152   298    */
EEV_1152x1242_6ph,     /* EEV1152x1242 6 PHASE          1152  1242    */
NOT_USED5,             /* Use custom chip               1024  2048    */
NOT_USED6,             /* Use custom chip               1024  2048    */
TEK_1024x1024_B_200ns, /* TEK1024x1024B Back Illm       1024  1024    */
TEK_1024x1024_F_200ns, /* TEK1024x1024F Front Illm      1024  1024    */
KDK_1024x1536,         /* KODAK 1024x1536               1024  1536    */
TEK_512x512_B_200ns,   /* TEK512x512B [200ns]            512   512    */
TEK_512x512_F_200ns,   /* TEK512x512F [200ns]            512   512    */
NOT_USED1,             /* NOT USED                         ?     ?    */
TEK_512x512D_B,        /* TEK512x512D Back Illm          512   512    */
TEK_512x512D_F,        /* TEK512x512D Front Illm         512   512    */
HAM_64x1024,           /* HAMMAMATSU   64 x 1024          64  1024    */
HAM_128x1024,          /* HAMMAMATSU  128 x 1024         128  1024    */
HAM_256x1024,          /* HAMMAMATSU  256 x 1024         256  1024    */
EEV_256x1024_8ph,      /* EEV 256 x 1024 8 PHASE         256  1024    */
EEV_1152x770_3ph,      /* EEV1152x770  3 PHASE          1152   770    */
EEV_1152x770_6ph,      /* EEV 1152x770 6 PHASE          1152   770    */
TEK_1024x1024_B_42usV, /* TEK1024x1024B Back Illm       1024  1024    */
PID_330x1100_6phH,     /* PI 330x1100 6 PHASE (horz)     330  1100    */
EEV_256x1024_6ph_CCD30,/* EEV 256x1024  6 PHASE CCD30    256  1024    */
TEK_1024x1024D_B,      /* TEK1024x1024D Back Illm       1024  1024    */
TEK_1024x1024D_F,      /* TEK1024x1024D Front Illm      1024  1024    */
TEK_1024x1024D_B_T3,   /* TEK1024x1024D Back Illm       1024  1024    */
THM_512x512,           /* Thomson 512X512 Front Illum    512   512    */
THM_256x1024,          /* Thomson 256X1024 FI MPP        256  1024    */
THM_2048x1024_FT,      /* Thomson 2048X1024 FT          1024  1024    */
SIT_800x2000_B,        /* SIT 800x2000 Back Illm         800  2000    */
SIT_800x2000_F,        /* SIT 800x2000 Front Illm        800  2000    */
PID_240x330_MCT,       /* TEST CHIP # 1                               */
OEEV_1203x1336_3ph,    /* EEV 1203x1336                 1203  1336    */
OEEV_1203x1336_6ph,    /* EEV 1203x1336                 1203  1336    */
PI_800x1000_B,         /* PI 800x1000 back               800  1000    */
PI_64x1024,            /* PI special   64 x 1024          64  1024    */
PI_128x1024,           /* PI special  128 x 1024         128  1024    */
PI_256x1024,           /* PI special  256 x 1024         256  1024    */
KDK_4096x4096,         /* Kodak 4096x4096               4096  4096    */
EEV_100x1340_6ph_CCD36,/* EEV 100x1340  6 PHASE CCD36    100  1340    */
PID_1030x1300,         /* PI Special 1030x1300          1030  1300    */
VICCD_NTSC_480x640,    /* Video Chip - N. American Std   480   640    */
VICCD_CCIR_576x768,    /* Video Chip - European Std      576   768    */
PID_582x782,           /* PI Special 582x782             582   782    */
PID_2500x600_B,        /* PI 2500x600 Back              2500   600    */
```

```
    PID_2500x600_F,         /* PI 2500x600 Front             2500   600    */
    TEST_CHIP_2,            /* TEST CHIP # 2                               */
    EEV_400x1340,           /* EEV 400x1340                   400  1340    */
    EEV_700x1340,           /* TEST CHIP # 3                               */
    EEV_1024x1024,          /* TEST CHIP # 4                               */
    EEV_1024x1024_FT,       /* EEV frame transfer            1024  1024    */
    EEV_1300x1340,          /* EEV 1300x1340                 1300  1340    */
    SIT_2048x2048_B,        /* SITE 2048x2048 Back           2048  2048    */
    SIT_2048x2048_F,        /* SITE 2028x2048 Front          2048  2048    */
    TEST_CHIP_6,            /* TEST CHIP # 6                               */
    TEST_CCD36_00,          /* Special ccd36 for EEV.         110  1356    */
    TEST_CCD36_10,          /* Special ccd36 for EEV.         410  1356    */
    TEST_CCD36_20,          /* Special ccd36 for EEV.         710  1356    */
    TEST_CCD36_40,          /* Special ccd36 for EEV.        1330  1356    */
    THM_2048x2048,          /* Thomson 2048x2048             2048  2048    */
    OEEV_1300x1340,         /* EEV 1300x1340 [OverScan]      1300  1340    */
   EPIX_1300x1024,         /* Epix Controller */
    END_OF_CCD_LIST         /* END OF ENUMERATED CCD TYPE,                 */

/* sensor type definition */
/* #### used by PICM_CREATECONTROLLER param 2 (if diode array) #### */
enum ctrl_PDA_sensor

    NO_PDA_SENSOR =1000,
    DA0128S,                /* single  128 */
    DA0256S,                /* single  256 */
    DA0512S,                /* single  512 */
    DA1024S,                /* single 1024 */
    DA2048S,                /* single 2048 */
    DA0128D,                /* dual    128 */
    DA0256D,                /* dual    256 */
    DA0512D,                /* dual    512 */
    DA1024D,                /* dual   1024 */
    DA2048D,                /* dual   2048 ! */
    DA256S_INGAS,           /* Single 256 INGAS */
    DA512S_INGAS,           /* Single 512 INGAS */
    DA128S_GE,              /* Single 128 GE    */
    DA256S_GE,              /* Single 256 GE    */
    PDA_DUMMY               /* indicates end of list  */
```

To determine which array to select for your detector, use any text editor to look at the PIHWDEF.INI file included with newer systems that were shipped with WinView or WinSpec. The beginning of the INI file should look similar to the following:

```
;PI CONTROLLER DEFAULT PARAMETERS

 OriginalCustomerName            = CUSTOMER NAME HERE
 NumberOfControllers             = 1
;     CONTROLLER 1 SETTINGS

    Controller                   = 1
    ExposureTime                 = 0.100000
    ControllerType               = ST 130
    DetectorType                 = TEK 512x512D Back
```

This last line indicates that the software should be set to `TEK_512x512D_B`. If you cannot determine which array is the appropriate array or do not have the PIHWDEF.INI file, have your detector and controller model type and serial number information handy and contact Customer Support at the Roper Scientific factory:

Roper Scientific, 3660 Quakerbridge Rd., Trenton, NJ 08619 USA
Phone: (609) 587-9797 Fax: (609) 587-1970 e-mail:
TECHSUPPORT@ROPERSCIENTIFIC.COM.

Read out mode is selected by using the `Data_Collection_Mode` parameter. In most cases, this should be set to `ROM_FULL` for normal data collection. The collection mode can be set to `ROM_FRAME_TRANSFER, INTERLINE,` or `ROM_KINETICS` to take advantage of the ST-138's frame transfer or kinetics capabilities, respectively.

Note: These additional modes can only be used by detectors capable of performing these functions.

Application Type

Finally, `Application_type` is used to specify the type of data collection that the system is being used for. For now, you may only select `APP_NORMAL_IMAGING`. This parameter will allow the controller to take the correct shutter into account when calculating proper timing.

The last parameter sent to this function should be a pointer to an unassigned integer where `PICM_CreateController` can store an error value if needed. This value should be checked if `PICM_CreateController` returns zero(FALSE).

Example

```
/* Create controller, assign detector, set defaults for */
/* application. */
controller_ok =  PICM_CreateController(ST138,
                                       EEV_576x384_3ph,
                                       ROM_FULL,
                                       APP_NORMAL_IMAGING,
                                       &error_code);
```

Destroying the Controller Object

After you have performed all of your PICM functions, `PICM_CleanUp` should be called to free the memory being used by the controller object. Remember that the controller will have to be re-initialized with `PICM_CreateController` to use any PICM functions again.

Example

```
if( (error_code != 0) || !controller_ok )
  {
    PICM_CleanUp();
    controller_ok = FALSE;
    MessageBox ( hWnd,
             "ERROR in Downloaded",
             "DownLoad Button",
             MB_OK | MB_ICONEXCLAMATION );
    /* clean up on error        */
    /* release big buffer       */
    GlobalUnlock(hglb);
```

```
GlobalFree(hglb);
```

Important

`PICM_CleanUp` does not clear memory that you allocate for data handling. It is your program's responsibility to unlock and free any memory that is implicitly allocated by your own code.

Selecting A/D Rate

Some controllers are equipped with two separate analog-to-digital converters. To select between a fast or slow ADC, use the `PICM_Set_Fast_ADC` or `PICM_Set_Slow_ADC`. These functions do not require parameters.

Auto-Stop

(Applies to ST138, MicroMAX (ST133) and PentaMAX (DC131) only.

Auto-Stop is a hardware feature that allows capture of only one frame per Start Controller. Auto-Stop allows easy data collection from the user's software. The three multiple-frame programs provided, **`mltcomex`**, **`mlt2comx`** and **`mltaccum`**, make use of the Auto-Stop capability. In the case of the PentaMAX (beginning with ver 3) and MicroMAX (beginning with ver 1) controllers, the number of frames to be collected can be specified. This allows the system to collect many frames and have the hardware automatically stop data acquisition when the specified number of frames have been collected. This capability will be available in the easy dlls beginning with version 1.6.

Note: Auto-Stop was implemented on the Model 138 in early 1994, and has since been implemented on the PentaMAX (DC131) and MicroMAX (ST133).

This page intentionally left blank.

Chapter 5
Controller Parameters

With a software controller object allocated and initialized, specific data acquisition settings can be specified. PICM provides functions for setting the computer interface card and exposure. These settings can then be downloaded to the hardware controller before beginning data collection.

Note: The settings for the computer interface card and exposure are required by PICM for controller operation.

Interface Card

`PICM_SetInterfaceCard` specifies which computer interface card PICM will be communicating through. This function is declared as:

```
int _export FAR PASCAL PICM_SetInterfaceCard

(
 int Interface_Card,     /* Interface card, see interfaceType enum          */
 unsigned int Base_Address,/* base address of interface card, not used eisa */
 int Card_interrrupt,    /* Interrupt to use, see interrupt_channel enum    */
 unsigned int *error_code/* Error code, used if function returns false.     */
 );
```

The legal values for `Interface_card` are as follows:

```
DMA_Interface        = 1 /* DMA  : fast! (ISA board)                      */
TAXI_Interface       = 2 /* TAXI : fast serial  (ISA board)               */
EISA_Interface       = 3 /* EISA : EISA computer board                    */
TAXI_TypeB_Interface = 4 /* TAXI : ISA board in EISA computer, Type B DMA */
PCI_COMPLEX_PC_Interface = 11 /* PCI standard                             */
PCI_TIMER_Interface = 20/* PCI With timer no interrupts                  */
```

TAXI is Princeton Instrument's internal name for the fast serial interface controller card. When using this board in an ISA computer `TAXI_Interface` would be the appropriate card selection. If you are using this card in an EISA computer, make sure that you specify the `TAXI_TypeB_Interface` for proper communications. Note that this interface card is not at all the same as the `SERIAL_Interface` which specifies the RS170 interface for the Video ICCD camera.

`Base_Address` should be set to the base address of your ISA interface card, if needed. Default value: `0xA00`. This address is used only on ISA cards and is set by the address switch (SW1) on the card. On the switch 0=On and the default address is 0xA00.

EISA and PCI don't use the base address. The operating system sets the base address for these interfaces.

Use on of the following values to select the `Card_interrupt`:

```
CHANNEL_10=0,
CHANNEL_11=1,
CHANNEL_12=2,
CHANNEL_15=3,
CHANNEL_TIMER=88,
```

For the TAXI serial card and for the EISA card, the interrupt channel is selected on jumper JP4 and may be set to 10, 11, 12, or 15. The parallel DMA card may only be set to either channel 10 or 11.

PCI doesn't need to have the interrupt set because the system (plug and play) does it for you (even for Windows 3.1).

Exposure Setting

The second required parameter is exposure, specified by `PICM_SetExposure`:

```
/* Set the camera exposure in seconds.                                        */
/* int PICM_SetExposure                                                       */
/*   double exposure          :: exposure time in seconds                     */
/*   unsigned int *error_code :: Error code, used if function returns false. */
```

Then, set exposure to the desired exposure time in seconds. If any problems are encountered, `PICM_SetExposure`will set `error_code` to `EXPOSURE_SETTING_ERROR`.

Chapter 6

Imaging Option

PICM's Imaging Option adds a single function to the library that permits a specific Region of Interest to be selected and allows binning parameters to used. This function can only be used when the Imaging Option is selected during the installation of PICM and your .C file includes `PIIMGDEF.H` and `PIIMGFCN.H`. The imaging function is

```
int _export FAR PASCAL PICM_SetROI
(
 int startx,            /* 1st pixel of ROI in x dir. (note starts at 1) */
 int starty,            /* 1st pixel of ROI in y dir. (note starts at 1) */
 int endx,              /* last pixel of ROI in x dir.                   */
 int endy,              /* last pixel of ROI in y dir.                   */
 int groupx,            /* amount to bin/group x data.                   */
 int groupy,            /* amount to bin/group y data.                   */
 unsigned int *error_code/* Error code, used if function returns false.  */
 );
```

If an error is encountered by `PICM_SetROI`, it will return FALSE and set `error_code` as follows:

```
#define CONTROLLER_SETUP_WRONG 0x0001 /* Got error in getting info from     */
                                      /* controller, check to see if        */
                                      /* previous initialize/create commands*/
                                      /* executed without error. Note : ROI */
                                      /* not set.                           */

/* The following 4 messages are for illegal values for ROI, the function    */
/* PICM_SetROI will try to correct, but user should check code.             */
#define STARTX_ILLEGAL         0x0004
#define STARTY_ILLEGAL         0x0002
#define ENDY_ILLEGAL           0x0008
#define ENDX_ILLEGAL           0x0010

/* the following 2 warnings are start and end values where reversed. The  */
/* function swapped start end but code should be checked.                 */
#define X_VALUES_SWAPPED       0x0020
#define Y_VALUES_SWAPPED       0x0040

/* The following 2 errors are for illegal group/bin sizes.                */
#define GROUPSIZE_X_ERROR      0x0080
#define GROUPSIZE_Y_ERROR      0x0100



#define ROI_ERROR              0x0200 /* The ROI defined was not accepted   */
                                      /* by controller object. Check ROI    */
                                      /* values make sure valued for CCD    */
                                      /* and controller.                    */
```

Example

```
/* Set an ROI that is 300x300 and binned by 2 in the x */
/* direction and 3 in the y direction.                 */
if (controller_ok)
{
    controller_ok = PICM_SetROI( 1, 1, 300, 300,
                                 2, 3, &error_code);
}
```

Chapter 7
Video Protocol

RS170

(Applies to MicroMAX (ST133) and PentaMAX (DC131) only.)

The MicroMAX and PentaMAX both support RS170 (NTSC & PAL) video. By using the RS170 capability, focusing can be accomplished swiftly and easily. During data collection, RS170 should be OFF to minimize noise. User programs can activate RS170 by passing the `PICM_Set_RS170_enable (pifcsfcn.h)` a TRUE, and calling `initialize_system`. The center region of the selected Region of Interest (ROI) that fits on the RS170 screen (NTSC 756x484, PAL 741x574) will be displayed. *Data for the full ROI selected will continue to be sent to the computer.* Note that RS170 doesn't work for multiple ROIs.

Version 2 of the PentaMAX has a hardware lookup table that can be easily programmed for RS170. PentaMAX version 5 and MicroMAX version 1 cameras have double looping. As a result they can be programmed to include RS170 zoom and pan functions (`PICM_Easy_Focusing in pifcsfcn.h and pifcsdef.h`).

To change the video output to either PAL or NTSC (default), use the `PICM _CMSetLongParam`.

PICM_CMSetLongParam

Summary

```
/* enum PICM_CMSetLongParam                                    */
/*   enum CM_CMD param     ::     CMP_VIDEO_TYPE must be used   */
/*   long param2           ::     1 if NTSC video, 2 if PAL video  */
```

Description

`PICM_CMSetLongParam` sets the variables inside the controller routine. Currently the only supported variable is `CMP_VIDEO_TYPE`. This function should be called before `PICM_Initialize_System`.

Example

```
/* Set up PAL video */
PICM_CMSetLongParam (CMP_VIDEO_TYPE, 2);
```

This page intentionally left blank.

Chapter 8
Collecting Data

Allocating Memory for Data

Before using PICM's data collection procedures, you will need to allocate memory for storing data as it is received. the `PICM_SizeNeedToAllocate` function will return a value for the amount of memory that you will need for one frame of data.

```
long PICM_SizeNeedToAllocate (void):
```

You can then allocate a block of memory with `GlobalAlloc` The handle returned by `GlobalAlloc` should then be locked with `GlobalLock` to a pointer of `void *huge` type to be used by `PICM_Initialize_System`.

When the data is written to memory, it is stored in a two dimensional array, starting with row and pixel closest to the read-out amplifier and then continuing row by row until reaching the pixel that is furthest from the readout amp. Each pixel uses two bytes of memory. 8bit and 18bit data are not currently supported by the PICM library.

Initializing the Hardware

The final step before initiating data collection is to initialize the hardware controller using `PICM_Initialize_System`. This function will download all of the parameters that were defined in previous functions to the controller and sets up the DMA buffers, ring buffer, and the users buffer. It Also hooks in the interrupt routine used by data collection. All special settings (exposure, ROIs, etc.) should be done before calling this function.

```
int _export FAR PASCAL PICM_Initialize_System

(
 void huge* big_buffer,   /* Users data buffer, Data collected stored here */
 unsigned int *error_code/* Error code, used if function returns false.    */
 );
```

Performing Data Collection

Data collection is triggered by the `PICM_Start_controller` procedure. Data collection will begin very shortly after the command is called since the controller initialization and setup should have already occurred with `PICM_Initialize_System`. `PICM_Start_controller` has no parameters or return values.

With data collection under way, you may continue your application and poll the controller `PICM_ChkData` to determine when data collection has been completed. `PICM_ChkData` is defined as

```
/* int PICM_Chk_Data                                                    */
/*   unsigned int *status_code   ::  status code                        */
```

PICM_ChkData returns False until data collection is completed or if an error has occurred, at which point it will return TRUE. By examining the setting of each bit in status_code you can determine if data collection was successful or if an error occurred:

```
#define RUNNING         1   /* bit for "experiment running"              */
#define WAITING         2   /* bit for "waiting for start event"         */
#define CONERROR        4   /* bit for "controller error"                */
#define COMERROR        8   /* bit for "command error"                   */
#define NEWDATARDY     16   /* bit for "new data block received"         */
#define INITERROR      32   /* bit for "no initialization done"          */
#define NEWDATAFIXED   64   /* bit for "DMA_copy_buffer -> to new data"  */
#define DATAOVERRUN   128   /* bit for "DMA data overrun"                */
#define VIOLATION     256   /* bit for "TAXI violation"                  */
#define MAILERROR     512   /* bit for EISA mail box communication error */
#define XFERERRORCH0 1024   /* bit for Channel zero xfer not enabled.    */
#define XFERERRORCH1 2048   /* bit for Channel one  xfer not enabled.    */
#define DONEDCOK     4096   /* bit for done data collection.             */
```

When PICM_ChkData returns TRUE, the data collected from your detector will have been stored into the memory block that had been passed to the PIXCM.DLL by PICM_Initialize_System.

After data collection has been completed, the detector controller must be properly stopped with PICM_Stop_controller. There are no parameters for this function. If you are using a ST138, ST133 (SpectroMAX or MicroMAX), or DC131 (PentaMAX) in the AutoStop mode (default), then you should not call PICM_Stop_controller since the hardware automatically stops.

Note: Do not forget to unlock and free your data memory block after calling PICM_CleanUp before exiting your application.

Example

```
/*  Startup Controller & acquire data  */
PICM_Start_controller();

done_flag = FALSE;
/* Wait for data to be collected. */
while (!done_flag)
    done_flag = PICM_ChkData(&status);

if (status & DONEDCOK)
{
    /*  Write Test File  */
    Test_WriteTestFile( "test.spe", lpvBuffer, 2 );
    MessageBox ( hWnd,
                 "Data has been Collected!",
                 "Collect Button",
                 MB_OK | MB_ICONEXCLAMATION );
```

Direct Access to the DMA Buffers Linear Counterpart Per Frame

The function `PICM_LockCurrentFrame` provides an address into the DMA buffer and a size. When you call this function you want to be acquiring data because what it returns in its parameters is the address and size of a frame of data within the buffer. The prototype for this function is defined in `pigenfcn.h` and looks as follows.

```
PREHEAD int PISTDAPI PICM_LockCurrentFrame (void **LinearAddress, unsigned
                                            long *BufferSize, unsigned int
                                            *ErrorCode);
```

The return value is of type integer and will be either success (1) or failure (0). Parameter 1 is of type void ** and returns the address of the either the current frame (focus mode) or the next frame (nframe mode) in the buffer to be pulled out. Parameter 2 is of type unsigned long * and returns the size of the frame pulled out. The third parameter is of type unsigned int * and returns the error status of the acquisition.

```
PREHEAD int PISTDAPI PICM_UnlockCurrentFrame( void );
```

This function should be called when you are done using the address from **`PICM_LockCurrentFrame`** and before you attempt to lock another frame. It returns success (1) or failure (0).

Example: The following example is of a thread created after creating and initializing a controller, whose sole purpose is to check for data in the background of the application and to update a display when new data is acquired. You can do the same thing with a `while` loop without a thread, but the thread is more elegant, and will not hang your application like a `while` loop can. This example loops until a variable to exit has been set and simply checks for data. If new data has been acquired, it updates a display with the data. It also checks to see whether or not the DMA buffer has been filled (DATAOVERRUN) and if the serial cable has been accidentally pulled (VIOLATION). One thing to note when using threads if you call **`PICM_CleanUp`** or have not created a controller successfully, when you call **`PICM_LockCurrentFrame`**, you may have problems because no controller exists. It is imperative that this thread exit to completion before calling **`PICM_CleanUp`.**

```
DWORD WINAPI DataCollectionThread(PVOID x)
{
   void               *caddress;
   unsigned  long     size;
   unsigned int       error_code;
   DWORD              dwCode;

   /* While loop until variable exit thread is set from application     */
   while ( (TRUE) && (!exitthread))
   {
      /* Get the address of the current frame                          */
      /* If there is a frame ready                                      */
      if ( PICM_LockCurrentFrame( &caddress, &size, &error_code ))
      {
         /* Update the display or whatever you want to do with the frame */
```

```
            if ( caddress != NULL )
               UpdateDisplay( (unsigned short*)caddress, size );
            /* Check for data overrun                                    */
            /* DATAOVERRUN defined in pigendef.h                         `*/
            if ( error_code & DATAOVERRUN )
            {
               PICM_Stop_controller();
               MessageBox( NULL, "Data Overrun", "Error",MB_OK);
            }
            /* Check for serial violations                               */
            /* VIOLATION defined in pigendef.h                           */
            if ( error_code & VIOLATION )
            {
               PICM_Stop_controller();
               MessageBox( NULL, "Serial Failure", " Error", MB_OK );
            }

            /* Unlock the frame                                          */
            PICM_UnlockCurrentFrame();
         }  /* End of If Statement                                      */
      }   /* End of While Loop                                          */

      /* Once exit thread has been set get out                          */
      exitthread = 0;
      /* Note: application waits for 0 before PICM_CleanUp              */
      /* thandle is thread handle stored in application                 */
      GetExitCodeThread( thandle, &dwCode );
      ExitThread( dwCode );
   }  /* End of DataCollectionThread                                    */
```

If in the past you where using **`PICM_ChkData( &error )`**, this function will work as it did in Windows 3.xx under Windows 95 and Windows NT. What this function does is call **`PICM_LockCurrentFrame`** and then copy the data from the DMA buffer into the buffer from **`PICM_Initialize_System`** and then calls **`PICM_UnlockCurrentFrame`**. So if you where using the **`PICM_ChkData`** function the down side is an extra copy(which is pretty fast anyway) and the allocation of an extra buffer. Here are two code fragments illustrating the difference in the initial setup between using **`PICM_ChkData`** and **`PICM_LockCurrentFrame.`**

Old Way using `PICM_ChkData`

```
   {
      /* Get the Size                                                 */
      buf_size = PICM_SizeNeedToAllocate();

         /* Allocate And Lock Memory Free When Done                   */
         bhandle  = GlobalAlloc( GPTR, buf_size );
         lpvBuff  = (void huge*) GlobalLock( bhandle );

         /* Initialize Buffer for Collection                          */
         status = PICM_Initialize_System( lpvBuff, &error );
   }
```

New Way using `PICM_LockCurrentFrame / PICM_UnlockCurrentFrame`

```
   {
      /* Notice passing a null pointer, this is fine as long as we   */
      /* Don't call PICM_ChkData which will try and copy from        */
      /* the DMA buffer to the null pointer                          */
```

```
        status = PICM_Initialize_System( NULL, &error );
    }
```

Synchronous vs. Asynchronous Acquisition

Synchronous

Continuous flow of data; a single call to start the controller returns data until either a stop is issued or until a data overrun occurs.

Note: Data overrun can only occur in `N-Frame` mode.

Asynchronous

Single shot; each call to start the controller returns only a single frame or a set number of frames. This allows data collection to go at a pace the computer can handle.

To set the controller up for the various modes of acquisition we use the function **`PICM_Set_AutoStop.`** The prototype can be found in **`pimltfcn.h`** and looks as follows.

```
PREHEAD int PISTDAPI PICM_Set_AutoStop( int number )
```

Synchronous Mode Acquisition

```
/* Call this function with a zero set before calling          */
/* InitializeSystem will force synchronous acquisition        */
PICM_Set_AutoStop( 0 );
```

Asynchronous Mode Acquisition

```
/* Call this function with n - number of frames to acquire    */
/* on a single start. Note: Some controllers can not count    */
/* frames and do not support this feature with values other   */
/* than 1.                                                    */
PICM_Set_AutoStop( n );
/* Valid n values                                             */
/* PENTAMAX(DC131) or MICROMAX (ST133) -> 0 to 254            */
/* ST138 -> 0 or 1                                            */
```

The default for the PC Library DLLs is acquisition in Asynchronous mode with AutoStop set to 1.

Nframe vs. Focus Mode

Nframe Mode

Calls to **`PICM_ChkData`** or **`PICM_LockCurrentFrame`** pull off frames in the order that they are transferred into the system without skipping any frames. Applications for this mode of operation include experiments where it is important to get each frame for a burst of frames. Any extended running in synchronous nframe mode will probably cause a data overrun (depends on CPU speed, software speed and ADC rate). This occurs when the DMA buffer is filled to capacity with the number of frames it can hold.

Focus Mode

Calls to **PICM_ChkData** or **PICM_LockCurrentFrame** pull off only the most recent frame and discard the rest previous to it, because frames are discarded it is impossible to get an overrun in this mode, unless a hardware conflict occurs. Applications include those where only updating the screen with current data is important, like focusing the camera. To maximize the update rate put the camera in synchronous mode.

To set the modes up through the EASYDLLs, you need to call the function **PICM_Set_EasyDLL_DC.** The prototype for this is in **pigenfcn.h** and looks as follows.

```
PREHEAD int PISTDAPI PICM_Set_EasyDLL_DC(int mode);
```

The possible modes are defined in **pigendef.h.**

Setting Nframe Mode

```
PICM_Set_EasyDLL_DC (EASYDLL_NFRAME_MODE);
```

Setting Focus Mode

```
PICM_Set_EasyDLL_DC (EASYDLL_FOCUS_MODE);
```

In order for these functions to take effect, they must be called prior to an initialize system.

Note: An application can switch back and forth between running nframe or focus mode as well as synchronous vs. asynchronous acquisition by calling the right picm function and then calling **PICM_Initialize_System**.

Working with Data Acquisition Events

Due to the multithreaded, multitasking environments of Windows NT and Windows 95, we decided to take advantages of the use of Events for data collection purposes. What you can do is set up an event through the Princeton Instruments DLLs and have a thread fall asleep waiting for the event. When a frame comes in, it will wake up the waiting thread and do any type of display or processing you want to do. This is done with the function **PICM_SetUserEvent**, the prototype for this can be found in **pievtfcn.h** and looks as follows.

```
PREHEAD HANDLE PISTDAPI PICM_SetUserEvent(
       int sec,         /* Security attributes                    */
       BOOL bManual,    /* Manual Reset Flag                      */
       BOOL bInitial,   /* Initial State                          */
       LPCSTR name,     /* Event name                             */
       int number);     /* Number of Frames                       */
```

The first four parameters are the same as those used in the standard API for `CreateEvent`. The last parameter is the number of frames to wait on for the event to be

signaled (currently set to 1). The return value is the handle on which you can wait. If this fails the handle will be NULL. Using a named eventonly works under Windows 95.

Example: The following example illustrates the use of data acquisition events.

Note: PICM_SetUserEvent will only work with interrupts and not with timer mode.

```
      /* Global Scope                                                    */
      DWORD WINAPI  ImageThread(PVOID x);
      DWORD  Thread_ID;

      /* Local Function                                                  */
      HANDLE userevent = 0;

      /* Create Controller                                               */
      /* Set Interface                                                   */
      /* Initialize System                                               */

      userevent PICM_SetUserEvent(0,     /* Security attributes          */
                      0,                /* Manual Reset Flag            */
                      0,                /* Initial State                */
                      0,                /* Event name                   */
                      1);               /* Number of Frames             */

      /* Create A thread to Wait on Event                                */
      CreateThread(0, 0x1000, ImageThread, userevent, 0, &Thread_ID);

      /* End Local Function                                              */
      /* Now the thread shown below is waiting for a frame of data.      */
      /* At any time you can start and stop a controller and this Thread */
      /* Will update the display */


/*********************************************************************
*
*   ImageThread
*   Waits for data event to happen; After Creating, Initializing, Setting
*   up event, and creating this thread once PICM_Start_controller has been hit,
*   the WaitForSingleObject should get triggered after a frame has been
*   transferred into ram.
*
*********************************************************************/
DWORD WINAPI ImageThread(PVOID hEventR3)
{
   long i;
   unsigned long size;
   unsigned int error;
   void *caddress;

   /* Infinite Loop */
   /* Note that this thread does not exit (it should). See example          */
   /* On page 55 for cleanly exiting a thread                              */
   while (TRUE)
   {
      /* Wait for event. Note: hEventR3 is passed in at creation as userevent*/
      /* Unlike the thread in the page 55 example, this uses no cpu time     */
      i = WaitForSingleObject((HANDLE)hEventR3, INFINITE );

      /* Switch based on WaitForSingleObject Return value                  */
```

```
            switch ( i )
            {
            case WAIT_ABANDONED:
               break;
            case WAIT_OBJECT_0:

               if ( PICM_LockCurrentFrame( &caddress, &size, &error ))
               {
                  if ( caddress != NULL )
                     UpdateDisplay((unsigned short *)caddress , size);
                  /* Check for data overrun                                    */
                  /* DATAOVERRUN defined in pigendef.h */
                  if ( error_code & DATAOVERRUN )
                  {
                     PICM_Stop_controller();
                     MessageBox( NULL, "Overrun", "Error",MB_OK);
                  }
                  /* Check for serial violations */
                  /* VIOLATION defined in pigendef.h */
                  if ( error_code & VIOLATION )
                  {
                     PICM_Stop_controller();
                     MessageBox( NULL, "Serial", " Error", MB_OK );
                  }


                  PICM_UnlockCurrentFrame();
               }
               break;
            case WAIT_TIMEOUT:
               break;
            case WAIT_FAILED:
               error = GetLastError();
               break;
             }  /* End of switch statement                                    */
         }   /* End of While Loop                                             */

         /* Never Returns                                                     */
         return 0;

      }   /* End of ImageThread                                              */
```

Chapter 9
Utility Functions

PICM also provides several utility functions which are provided as an additional resource.

Sensor Size

`PICM_Get_sensor_x` and `PICM_Get_sensor_y` are used to determine the physical size of a detector's sensor. These functions have no parameters and return an integer value based on the number of elements the sensor has in the specified direction.

Pixel Dimensions

Some controller settings, such as Region of Interest and binning settings, affect the pixel dimensions of data being returned from the controller. `PICM_Get_pixeldimension_x` and `PICM_Get_pixeldimension_y` will return correct dimensions by taking ROI and binning into account.

Reading and Writing TTL Signals

ST-12x, ST-130, ST-133 (SpectroMAX or MicroMAX) and ST-138 controllers have a TTL port which can be used to connect to other laboratory equipment.

Note: The reading and writing of TTL signals is not supported by the DC-131 (PentaMAX).

To write to this port, use `PICM_Set_TTL_pattern`. This function accepts a single integer value and sets the TTL output signals accordingly. `PICM_Get_TTL_pattern` returns an integer value corresponding to the current state of the TTL input on the controller.

This page intentionally left blank.

Chapter 10
Example Program

The following is an example program that collects data from a system using an ST-138 controller with a detector containing a 578×384 EEV CCD.

Note: The last function in this example will save the image data in a format readable by WinView.

Simple Use of ST-138 With EEV 578x384 CCD

```
//Filename: Gencomex.C   General Example program.

#include <WINDOWS.H>
#include "malloc.h"
#include "stdlib.h"
#include "math.h"
#define PIXCM 1
#include "platform.h"
#include "GENCOMEX.H"
#include "GENCOMEX.WMC"
#include "newtypes.h"    /* common data types. */
#include "pigendef.H"    /* PI General Definition file for easy dlls (enums) */
#include "pigenfcn.h"    /* PI General functions for easy dlls.              */
#include "pimltfcn.h"

/* Example function that writes data in format that is readable by WinView  */
#include "cheader.h"    /* Used by WriteTestFile                             */
#include <stdio.h>     /* compiler dependend include files                   */
void Test_WriteTestFile
(
 char     *runfile,               /* data file to write to.                  */
 void huge *big_buffer,           /* buffer holding data.                    */
 int byte_size_of_pixel           /* size of a pixel in bytes                */
 );

BOOL    CreateButtonControls(HWND);


//*************************************************************
//               WinMain FUNCTION
//*************************************************************

int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance, LPSTR lpCmdLine, int
nCmdShow)
    {
    MSG msg;                    // message

    hInst = hInstance;          // Saves the current instance

    if (!BLDInitApplication(hInstance,hPrevInstance,&nCmdShow,lpCmdLine))
```

```
        return FALSE;

    if (!hPrevInstance)         // Is there an other instance of the task
        {
        if (!BLDRegisterClass(hInstance))
            return FALSE;       // Exits if unable to initialize
        }

    MainhWnd = BLDCreateWindow(hInstance);
    if (!MainhWnd)              // Check if the window is created
        return FALSE;

    ShowWindow(MainhWnd, nCmdShow);  // Show the window
    UpdateWindow(MainhWnd);          // Send WM_PAINT message to window

    BLDInitMainMenu(MainhWnd);  // Initialize main menu if necessary

    /*  Create CONTROLLER MANAGER  */
//  Create_controllermanager();

    SetTimer( MainhWnd, 1000, 100, NULL );

    while (GetMessage(&msg,     // message structure
                      0,        // handle of window receiving the message
                      0,        // lowest message to examine
                      0))       // highest message to examine
    {
        if (BLDKeyTranslation(&msg)) // WindowsMAKER code for key translation
            continue;
        TranslateMessage(&msg); // Translates character keys
        DispatchMessage(&msg);  // Dispatches message to window


    }

    KillTimer( MainhWnd, 1000 );

    BLDExitApplication();       // Clean up if necessary

    return(msg.wParam);         // Returns the value from PostQuitMessage
}



//************************************************************
//           WINDOW PROCEDURE FOR MAIN WINDOW
//************************************************************

LONG FAR PASCAL BLDMainWndProc( HWND hWnd,
                                WORD message,
                                WORD wParam,
                                LONG lParam )
{
    /*  used for allocation of test buffer */
    long buffer_size;
    static HANDLE hglb;
    static void huge *lpvBuffer;
    static int initialize_flag = FALSE;     /* Set to true if init done  */
    int done_flag;                          /* flag for data collection  */
    int controller_ok;        /* Status of function called (t/f)         */
    unsigned int error_code;  /* if status is false, this holds error    */
```

```
 unsigned int status;      /* used for status of PICM_ChkData        */

switch (message)
 {

 case WM_CREATE:             // window creation
     CreateButtonControls ( hWnd );  // Create child text buttons
     // Send to BLDDefWindowProc in (.WMC) for controls in main window
     return BLDDefWindowProc(hWnd, message, wParam, lParam);
 break;

 case WM_SETFOCUS:           // window is notified of focus change
     // Send to BLDDefWindowProc in (.WMC) for controls in main window
     return BLDDefWindowProc(hWnd, message, wParam, lParam);
 break;

 case WM_DESTROY:       // window being destroyed
     PostQuitMessage(0);
     return BLDDefWindowProc(hWnd, message, wParam, lParam);
 break;


 case WM_COMMAND:            // command from the main window
     switch ( wParam )
     {
         case ID_DOWNLOAD_BTN:

             /* Create controller, assign detector, set defaults for */
             /* application. */
             controller_ok =  PICM_CreateController(ST138,
                                                    EEV_576x384_3ph,
                                                    ROM_FULL,
                                                    APP_NORMAL_IMAGING,
                                                    &error_code);
             if (controller_ok)
             {
                 /* Setup interface card (card in computer). */
                 /* See pigenfcn.h for parameter definition.  */
                 controller_ok = PICM_SetInterfaceCard(
                                 /* PC interface card    */
                                 TAXI_Interface,
                                 0xA00,           /* Base address of card */
                                 CHANNEL_10,      /* Interrupt used       */
                                 &error_code);    /* error codes.         */

                 /* ######################################################## */
                 /* Set special ADC here, Make sure setting comes after. */
                 /* PICM_SetInterfaceCard since it sets defaults for ADC */
                 /* ######################################################## */
                 PICM_Set_Fast_ADC();
                 if (controller_ok)
                 {
                     controller_ok = PICM_SetExposure(0.1, &error_code);
                     if (controller_ok)
                     {


                         /* **** temp until new DLLs available ***** */
                         /*  allocate a big buffer for this test     */
                         buffer_size = PICM_SizeNeedToAllocate();
```

```
                    hglb = GlobalAlloc(GPTR, buffer_size );
                    lpvBuffer = (void huge*) GlobalLock(hglb);

                    /* Initialize system (hardware and software). */
                    if (controller_ok)
                        controller_ok = PICM_Initialize_System(
                                                        lpvBuffer,
                                                        &error_code);
                }
            }
            if( (error_code != 0) || !controller_ok )
            {
                PICM_CleanUp();
                controller_ok = FALSE;
                MessageBox ( hWnd,
                         "ERROR in Downloaded",
                         "DownLoad Button",
                         MB_OK | MB_ICONEXCLAMATION );
                /* clean up on error        */
                /*  release big buffer */
                GlobalUnlock(hglb);
                GlobalFree(hglb);
            }
            else
            {

                MessageBox ( hWnd,
                             "Controller has been Downloaded",
                             "DownLoad Button",
                             MB_OK | MB_ICONEXCLAMATION );
                initialize_flag = TRUE;
            }
        }

        return FALSE;

    case ID_COLLECT_BTN:
        if (initialize_flag)
        {
            /*  Startup Controller & acquire data  */
            PICM_Start_controller();


            done_flag = FALSE;
            /* Wait for data to be collected.      */
            while (!done_flag)
                done_flag = PICM_ChkData(&status);

            if (status & DONEDCOK)
            {
                /*  Write Test File                */
                Test_WriteTestFile( "test.spe", lpvBuffer, 2 );


                MessageBox ( hWnd,
                             "Data has been Collected!",
                             "Collect Button",
                             MB_OK | MB_ICONEXCLAMATION );
            }
            else /* error in data collection       */
            {
```

```
                            MessageBox ( hWnd,
                                         "ERROR in data collection",
                                         "Collect Button",
                                         MB_OK | MB_ICONEXCLAMATION );
                        }
                        /* Clean up when done.                */
                        /*  release big buffer                */
                        /*  stop controller                   */
                        PICM_Stop_controller();
                        PICM_CleanUp();
                        GlobalUnlock(hglb);
                        GlobalFree(hglb);
                        initialize_flag = FALSE;
                    }
                    else    /* Down Load Not successful */
                    {
                        MessageBox ( hWnd,
                                     "You Must DownLoad 1st Successfully",
                                     "Collect Button",
                                     MB_OK | MB_ICONEXCLAMATION );
                    }
                    return FALSE;

                case ID_EXIT_BTN:

                    MessageBox ( hWnd,
                                 "Exit Not Implemented Yet",
                                 "EXIT Button",
                                 MB_OK | MB_ICONEXCLAMATION );

                    return FALSE;

                default:
                    break;
            }

            if (BLDMenuCommand(hWnd, message, wParam, lParam))
                break;              // Processed by BLDMenuCommand.
            // else default processing by BLDDefWindowProc.
        default:
            // Pass on message for default processing
            return BLDDefWindowProc(hWnd, message, wParam, lParam);
        }

        return FALSE;               // Returns FALSE if processed
    }


    /*****************************************************************************
    *
    *       Test_WriteTestFile
    *
    *       writes data to a PI SPE data format
    *
    *
    *----------------------------------------------------------------------------
    */
    void Test_WriteTestFile
    (
     char     *runfile,              /* data file to write to.   */
     void huge *big_buffer,          /* buffer holding data.     */
```

```
 int byte_size_of_pixel           /* size of a pixel in bytes */
 )
{


    /*  Generate a TEST WinView DATA file       */
    FILE * fileptr;    /*  Output File Pointer  */
    struct WINXHEAD * header;
    char huge *address;
    int jndx;

    header = (struct WINXHEAD *) malloc((size_t)sizeof(struct WINXHEAD));

    /*  fill in some of the header  */
    header->noscan = -1;

    header->datatype = 3;  //unsigned integer

    header->stripe   = PICM_Get_pixeldimension_y();
    /* if multiple frame, * frameNum */
    header->lnoscan  = header->stripe;
    header->faccount = PICM_Get_pixeldimension_x();
    header->dioden   = header->faccount;
    header->scramble = 1;

    fileptr = fopen( runfile, "w+b" );

    /*  write WinView HEADER to Output File  */
    fwrite( header,
            (size_t)4100,
            (size_t)1,
            fileptr );

    /*  release Header Memory               */
    free( header );

    /*  write data to Output File           */
    address = (char huge*)big_buffer;

    for( jndx = 0; jndx < PICM_Get_pixeldimension_y(); jndx++ )
    {
        fwrite( address,
                byte_size_of_pixel,
                PICM_Get_pixeldimension_x(),
                fileptr );

        address = address +
                  PICM_Get_pixeldimension_x() *
                  byte_size_of_pixel;
    }

    /*  close Output File                   */
    fclose( fileptr );



}   /*  end of Test_WriteTestFile()         */
```

Chapter 11
Programming Reference

application_type

Parameter for PICM_CreateController, defining the type of application that the Princeton Instruments system is being used for.

Prototype

```
/* Definition of application */
/* #### used by PICM_CREATECONTROLLER param 4  #### */
enum Application_Type
{
    APP_NORMAL_IMAGING,         /* General case imaging               */
    APP_SPECTROSCOPY,           /* for single (few) strip spectroscopy */
    APP_MICROSCOPY              /* for cameras on microscopes.        */
};
```

Description

application_type notifies the PIXCM.DLL how the Princeton Instruments system is being used.

For now, always use the variable APP_NORMAL_IMAGING. The others are not supported yet.

Example

```
/* Create controller, assign detector, set defaults for */
/* application. */
controller_ok =  PICM_CreateController(ST138,
                                       EEV_576x384_3ph,
                                       ROM_FULL,
                                       APP_NORMAL_IMAGING,
                                       &error_code);
```

Controller_type

Parameter for PICM_CreateController, specifying which Princeton Instruments hardware controller is being used.

Prototype

```
/* controller type definition */
/* #### used by PICM_CREATECONTROLLER param 1 #### */
enum ctrl_type
{
    ST130,              /* ST130             =  2 */
    ST138,              /* ST138             =  3 */
```

```
    DC131,              /* DC131(PentaMAX)   =  5 */
    ST121,              /* ST121             =  8 */
    ST135,              /* ST135             =  9 */
    ST133,              /* ST133(MicroMAX)   = 10 */
    ST133_5MHZ,         /* ST133 5Mhz system = 18 */
};
```

Description

`Controller_type` notifies the PIXCM.DLL of the model of Princeton Instruments controller being used.

Example

```
/* Create controller, assign detector, set defaults for */
/* application. */
controller_ok =  PICM_CreateController(ST138,
                                       EEV_576x384_3ph,
                                       ROM_FULL,
                                       APP_NORMAL_IMAGING,
                                       &error_code);
```

Data_Collection_Mode

Parameter for `PICM_CreateController`, specifying which data collection mode is to be used.

Prototype

```
/* CCD chip frame readout mode  */
/* #### used by PICM_CREATECONTROLLER param 3  #### */
/* for normal data collection USE ReadFullFrame */
enum sensorReadoutMode
{
  READOUT_NOTUSED = 0,   /* Readout Mode not used       */
  ROM_FULL,              /* normal data collection mode. */
  ROM_FRAME_TRANSFER,    /* frame transfer mode          */
  ROM_KINETICS,          /* kinetics (st138)             */
  ROM_INTERLINE,         /* interlined CCD mode          */
  ODD_FRAME,             /* viccd only                   */
  EVEN_FRAME,            /* viccd only                   */
  ANY_FRAME              /* viccd only                   */
};
```

Description

`Data_Collection_Mode` notifies the PICM.DLL which data collection mode is to be used.

Example

```
/* Create controller, assign detector, set defaults for */
/* application. */
controller_ok =  PICM_CreateController(ST138,
                                       EEV_576x384_3ph,
                                       ROM_FULL,
                                       APP_NORMAL_IMAGING,
```

```
                                        &error_code);
```

Detector_type

Parameter for `PICM_CreateController`, specifying which Princeton Instruments detector is being used.

Note: Users should always refer to pigendef.h to determine the latest chips supported.

Prototype

```
Diode Arrays
/* sensor type definition */
/* #### used by PICM_CREATECONTROLLER param 2 (if diode array) #### */
    DA0128S = 1001             /* single  128   */
    DA0256S = 1002             /* single  256   */
    DA0512S = 1003             /* single  512   */
    DA1024S = 1004             /* single 1024   */
    DA2048S = 1005             /* single 2048   */
    DA0128D = 1006             /* dual    128   */
    DA0256D = 1007             /* dual    256   */
    DA0512D = 1008             /* dual    512   */
    DA1024D = 1009             /* dual   1024   */
    DA2048D = 1010             /* dual   2048 ! */

CCDs
                               /*             CCD                X     Y      */
    EEV_256x1024_3ph  =  1     /* EEV 256x1024 3-phase           256  1024    */
    EEV_576x384_3ph   =  2     /* EEV 576x384 3-phase            576   384    */
    EEV_1152x298_3ph  =  3     /* EEV 1152x298 3-phase          1152   298    */
    EEV_1152x1242_3ph =  4     /* EEV 1152x1242                 1152  1242    */
    KDK_512x768       =  5     /* KODAK 512x768                  512   768    */
    KDK_1035x1317     =  6     /* KODAK 1035x1317               1035  1317    */
    KDK_1024x1280     =  7     /* KODAK 1024x1280               1024  1280    */
    KDK_2044x2033         =  8 /* KODAK 2044x2033               2044  2033    */
    KDK_2048x3072         =  9 /* KODAK 2048x3072               2048  3072    */
    PID_330x1100_8phH     = 12 /* PI 330x1100 8 phase (horz)     330  1100    */
    PID_532x1752          = 13 /* PI 532x1752                    532  1752    */
    RET_400x1200          = 14 /* RET 400x1200                   400  1200    */
    RET_512x512           = 15 /* RET 512x512                    512   512    */
    RET_1024x1024         = 17 /* RET 1K x 1K                   1024  1024    */
    RET_2048x2048         = 18 /* RET 2K x 2K                   2048  2048    */
    TEK_512x512_B_100ns   = 19 /* TEK512x512B Back  [100ns]      512   512    */
    TEK_512x512_F_100ns   = 20 /* TEK512x512F Front [100ns]      512   512    */
    TEK_1024x1024_B_100ns = 21 /* TEK1024x1024B Back [100ns]    1024  1024    */
    TEK_1024x1024_F_100ns = 22 /* TEK1024x1024F Front[100ns]    1024  1024    */
    TEK_2048x2048         = 23 /* TEK 2K x 2K                   2048  2048    */
    THM_576x384           = 24 /* TH576x384                      576   384    */
    EEV_256x1024_6ph      = 25 /* EEV 256x1024  6 PHASE          256  1024    */
    EEV_1024x512_FT       = 26 /* EEV frame transfer            1024   512    */
    EEV_576x384_6ph       = 29 /* EEV576x384   6 PHASE           576   384    */
    EEV_1152x298_6ph      = 30 /* EEV1152x298  6 PHASE          1152   298    */
    EEV_1152x1242_6ph     = 31 /* EEV1152x1242 6 PHASE          1152  1242    */
    TEK_1024x1024_B_200ns = 34 /* TEK1024x1024B Back Illm       1024  1024    */
    TEK_1024x1024_F_200ns = 35 /* TEK1024x1024F Front Illm      1024  1024    */
    KDK_1024x1536         = 36 /* KODAK 1024x1536               1024  1536    */
    TEK_512x512_B_200ns   = 37 /* TEK512x512B [200ns]            512   512    */
    TEK_512x512_F_200ns   = 38 /* TEK512x512F [200ns]            512   512    */
    TEK_512x512D_B        = 40 /* TEK512x512D Back Illm          512   512    */
    TEK_512x512D_F        = 41 /* TEK512x512D Front Illm         512   512    */
```

```
HAM_64x1024           = 42 /* HAMMAMATSU   64 x 1024          64  1024    */
HAM_128x1024          = 43 /* HAMMAMATSU  128 x 1024         128  1024    */
HAM_256x1024          = 44 /* HAMMAMATSU  256 x 1024         256  1024    */
EEV_256x1024_8ph      = 45 /* EEV 256 x 1024 8 PHASE         256  1024    */
EEV_1152x770_3ph      = 46 /* EEV1152x770  3 PHASE          1152   770    */
EEV_1152x770_6ph      = 47 /* EEV 1152x770 6 PHASE          1152   770    */
TEK_1024x1024_B_42usV = 48 /* TEK1024x1024B Back Illm       1024  1024    */
PID_330x1100_6phH     = 49 /* PI 330x1100 6 PHASE (horz)     330  1100    */
EEV_256x1024_6ph_CCD30,    /* EEV 256x1024  6 PHASE CCD30    256  1024    */
TEK_1024x1024D_B,          /* TEK1024x1024D Back Illm       1024  1024    */
TEK_1024x1024D_F,          /* TEK1024x1024D Front Illm      1024  1024    */
TEK_1024x1024D_B_T3,       /* TEK1024x1024D Back Illm       1024  1024    */
THM_512x512,               /* Thomson 512X512 Front Illum    512   512    */
THM_256x1024,              /* Thomson 256X1024 FI MPP        256  1024    */
THM_2048x1024_FT,          /* Thomson 2048X1024 FT          1024  1024    */
SIT_800x2000_B,            /* SIT 800x2000 Back Illm         800  2000    */
SIT_800x2000_F,            /* SIT 800x2000 Front Illm        800  2000    */
PID_240x330_MCT,           /* TEST CHIP # 1                               */
OEEV_1203x1336_3ph,        /* EEV 1203x1336                 1203  1336    */
OEEV_1203x1336_6ph,        /* EEV 1203x1336                 1203  1336    */
PI_800x1000_B,             /* PI 800x1000 back               800  1000    */
PI_64x1024,                /* PI special   64 x 1024          64  1024    */
PI_128x1024,               /* PI special  128 x 1024         128  1024    */
PI_256x1024,               /* PI special  256 x 1024         256  1024    */
KDK_4096x4096,             /* Kodak 4096x4096               4096  4096    */
EEV_100x1340_6ph_CCD36,    /* EEV 100x1340  6 PHASE CCD36    100  1340    */
PID_1030x1300,             /* PI Special 1030x1300          1030  1300    */
VICCD_NTSC_480x640,        /* Video Chip - N. American Std   480   640    */
VICCD_CCIR_576x768,        /* Video Chip - European Std      576   768    */
PID_582x782,               /* PI Special 582x782             582   782    */
PID_2500x600_B,            /* PI 2500x600 Back              2500   600    */
PID_2500x600_F,            /* PI 2500x600 Front             2500   600    */
TEST_CHIP_2,               /* PI 512x512                     512   512    */
EEV_400x1340,              /* EEV 400x1340                   400  1340    */
EEV_700x1340,              /* TEST CHIP # 3                               */
EEV_1024x1024,             /* TEST CHIP # 4                               */
EEV_1024x1024_FT,          /* EEV frame transfer            1024  1024    */
EEV_1300x1340,             /* EEV 1300x1340                 1300  1340    */
SIT_2048x2048_B,           /* SITE 2048x2048 Back           2048  2048    */
SIT_2048x2048_F,           /* SITE 2028x2048 Front          2048  2048    */
TEST_CHIP_6,               /* TEST CHIP # 6                               */
TEST_CCD36_00,             /* Special ccd36 for EEV.         110  1356    */
TEST_CCD36_10,             /* Special ccd36 for EEV.         410  1356    */
TEST_CCD36_20,             /* Special ccd36 for EEV.         710  1356    */
TEST_CCD36_40,             /* Special ccd36 for EEV.        1330  1356    */
THM_2048x2048,             /* Thomson 2048x2048             2048  2048    */
OEEV_1300x1340,            /* EEV 1300x1340 [OverScan]      1300  1340    */
EPIX_1300x1024,            /* Epix Controller                             */
PIB_512x512_FT,            /* PI Back 512x512 Frame Xfer(16-Tap)          */
HAM_60x1024,               /* HAMMAMATSU   60 x 1024          60  1024    */
HAM_124x1024,              /* HAMMAMATSU  124 x 1024         124  1024    */
HAM_252x1024,              /* HAMMAMATSU  252 x 1024         252  1024    */
EEV_1024x512_FT_CCD57,     /* EEV frame transfer CCD57      1024   512    */
END_OF_CCD_LIST            /* END OF ENUMERATED CCD TYPE,                 */
```

Description

Detector_type notifies the PICM.DLL which Princeton Instruments detector is being used.

Example

```
/* Create controller, assign detector, set defaults for */
/* application. */
controller_ok =  PICM_CreateController(ST138,
                                       EEV_576x384_3ph,
                                       ROM_FULL,
                                       APP_NORMAL_IMAGING,
                                       &error_code);
```

Interface_card

Parameter for `PICM_SetInterfaceCard`, specifying which computer-to-controller interface is being used.

```
/* Controller interface type */
/* #### used by PICM_SetInterface param 1  #### */
    DMA_Interface        = 1 /* DMA  : fast! (ISA board)                    */
    TAXI_Interface       = 2 /* TAXI : fast serial  (ISA board)             */
    EISA_Interface       = 3 /* EISA : EISA computer board                  */
    TAXI_TypeB_Interface = 4 /* TAXI : ISA board in EISA computer, Type B DMA */
    PCI_COMPLEX_PC_Interface = 11 /* PCI standard                           */
    DEMO_Interface       = 15     /* for demo software and test software.  */
    PCI_TIMER_Interface  = 20     /* PCI With timer no interrupts          */
```

Description

`Interface_card` sets the computer interface through which PICM.DLL will be communicating with a Princeton Instruments controller.

Example

```
/* Setup interface card (card in computer). */
/* See pigenfcn.h for parameter definition.  */
controller_ok = PICM_SetInterfaceCard(
                TAXI_Interface,  /* PC interface card    */
                0xA00,           /* Base address of card */
                CHANNEL_10,      /* Interrupt used       */
                &error_code);    /* error codes.         */
```

PICM_ChkData

Polls the hardware controller for data collection status.

Prototype

```
int _export FAR PASCAL PICM_ChkData  (unsigned int *status_code);
```

Description

With data collection underway, you may continue your application and poll the controller with `PICM_ChkData` to determine when data collection has been completed.

`PICM_ChkData` returns FALSE until data collection is complete, at which point it will return TRUE. By examining the status of each bit in `status_code` you can also check other information about the controller:

```
#define RUNNING         1   /* bit for "experiment running"             */
```

```
#define WAITING         2   /* bit for "waiting for start event"         */
#define CONERROR        4   /* bit for "controller error"                */
#define COMERROR        8   /* bit for "command error"                   */
#define NEWDATARDY     16   /* bit for "new data block received"         */
#define INITERROR      32   /* bit for "no initialization done"          */
#define NEWDATAFIXED   64   /* bit for "dma_copy_buffer -> to new data"  */
#define DATAOVERRUN   128   /* bit for "DMA data overrun"                */
#define VIOLATION     256   /* bit for "TAXI violation"                  */
#define MAILERROR     512   /* bit for EISA mail box communication error */
#define XFERERRORCH0 1024   /* bit for Channel zero xfer not enabled.    */
#define XFERERRORCH1 2048   /* bit for Channel one  xfer not enabled.    */
#define DONEDCOK     4096   /* bit for done data collection.             */
#define INITIALIZED  8192   /* bit to say controller has been initialized*/
```

When `PICM_ChkData` returns TRUE, the data collected from your detector will have been stored into the memory block that had been passed to the PICM.DLL by `PICM_Initialize_System`.

Example

```
/*  Initialize the controller passing a pointer to the user buffer lpvBuffer  */
controller_ok = PICM_Initialize_System(lpvBuffer, &error_code);

/*  Startup Controller & acquire data  */
PICM_Start_controller();

done_flag = FALSE;
/* Wait for data to be collected. */
while (!done_flag)
    done_flag = PICM_ChkData(&status);

if (status & DONEDCOK)
{
    /*  Write Test File  */
    Test_WriteTestFile( "test.spe", lpvBuffer, 2 );

    MessageBox ( hWnd,
                 "Data has been Collected!",
                 "Collect Button",
                 MB_OK | MB_ICONEXCLAMATION );
```

PICM_CleanUp

Frees memory used by PICM to store controller parameters.

Prototype

```
int _export FAR PASCAL PICM_CleanUp (void);
```

Description

`PICM_CleanUp` clears all memory allocated by the PICM.DLL. This will not unlock, or free, memory allocated by a program using PICM, even if a pointer had been passed to the DLL with the `PICM_Initialize_System`. Remember that no further calls to PICM functions may be made after this procedure has been initialized.

Example

```
if( (error_code != 0) || !controller_ok )
   {
      PICM_CleanUp();
      controller_ok = FALSE;
      MessageBox ( hWnd,
               "ERROR in Downloaded",
               "DownLoad Button",
               MB_OK | MB_ICONEXCLAMATION );
      /* clean up on error        */
      /*  release big buffer      */
      GlobalUnlock(hglb);
      GlobalFree(hglb);
```

PICM_Clear_MultStrip

When using multiple regions of interest, clears out all regions that have been defined.

Prototype

```
INT32 PICM_Clear_MultStrip( void );
```

Description

`PICM_Clear_MultStrip` clears all the pre-existing rois downloaded with the function

`PICM_SetROI_MultiStrip`. Returns true or false corresponding to success and failure.

Example

```
#include "pistpfcn.h"
#include "pigendef.h"
#include "pigenfcn.h"

void main()
{
   unsigned short *data;
   unsigned int error;
   int xbin, ybin;
   IRCT normal_region;

   /* First Create the controller object */
   PICM_CreateController( ST133,
                          KDK_1035x1317,
                          ROM_FULL,
                          APP_NORMAL_IMAGING,
                          &error );
   /* Select the interface type */
    PICM_SetInterfaceCard(   PCI_COMPLEX_PC_Interface, 0, 0, &error );

   /* Set the multiple region flag to true */
   PICM_Set_MultStrip_Flag( TRUE );

   /* Clear out any regions of interest */
   PICM_Clear_MultStrip( );
   PICM_Clear_user_rois( );

   /* set up some regions */
```

```
    PICM_SetROI_MultiStrip( 100, 100, 200, 200, 1, 1, &error );
    PICM_SetROI_MultiStrip( 300, 300, 400, 400, 1, 1, &error);

    /* Download the regions to the controller */
    PICM_Download_MultROI( &error );

    /* Generate the side effects that may occur */
    PICM_Generate_sideeffects( );

    /* Initialize the system and acquire some data */
    data = acquired data…

    /* Get the first normalized region, this region will be
        left = 1, top = 1, bottom = 100; right = 100; */
    normal_region = PICM_Get_normal_rois( normal_region, &xbin, &ybin, 1 );

    /* Get the first normalized region, this region will be
        left = 101, top = 101, bottom = 200; right = 200;*/
    normal_region = PICM_Get_normal_rois( normal_region, &xbin, &ybin, 2 );
}
```

PICM_Clear_user_rois

Clears out the regions entered by the user, note actual regions in hardware may vary.

Prototype

```
UINT32  PICM_Clear_user_rois( void );
```

Description

Clears the regions actually entered by the user, note the actual regions programmed to the hardware may vary slightly due to side effects that may be generated.

Example

See `PICM_Clear_MultStrip`

PICM_CMGetDoubleParam

Used for getting parameter values

Prototype

```
enum PICM_CMGetDoubleParam(enum CM_CMD param, double *value);
(see pivarfcn.h, pivardef.h)
```

Description

`PICM_CMGetDoubleParam` is used for getting values of double parameters.

Example

```
PICM_CMGetDoubleParam(CMP_ACTUAL_TEMP, &temp);
```

PICM_CMGetLongParam

Used for getting parameter values

Prototype

`enum PICM_CMGetLongParam(enum CM_CMD param, INT32 *value); (see pivarfcn.h, pivardef.h)`

Description

`PICM_CMGetLongParam` returns current setting for a given `CM_CMD`

Example

```
PICM_CMGetLongParam(  param,  *value);
```

PICM_CMSetDoubleParam

Used for getting parameter values

Prototype

`enum PICM_CMGetDoubleParam(enum CM_CMD param, double *value); (see pivarfcn.h, pivardef.h)`

Description

`PICM_CMGetDoubleParam` is used for getting values of double parameters.

Example

```
PICM_CMGetDoubleParam(CMP_ACTUAL_TEMP, &temp);
```

PICM_CMSetLongParam

Allow parameters to be set

Prototype

`enum CM_ERR PICM_CMSetDoubleParam(enum CM_CMD param, double value); (see pivarfcn.h, pivardef.h)`

Description

`PICM_CMSetDoubleParam` is used for setting parameters using a given `CM_CMD` and value.

Example

```
PICM_CMSetLongParam( CMP_ADC_OFFSET, 45 );
```

PICM_CreateController

Allocates memory for the Controller Object which is used to pass data and parameters between your program and the PICM.DLL.

Prototype

```
/* Create and Initialize controller object. Set some defaults. MUST be 1ST   */
/* function called.                                                           */
/* int PICM_CreateController                                                  */
/*  int Controller_type      :: controller to create, see ctrl_type           */
/*  int Detector_type        :: CCD/PDA to run, see ctrl_CCD_sensor           */
/*  int Data_Collection_Mode :: Data collection mode, see sensorReadoutMode   */
/*  int application_type     :: type of application, see Application_Type      */
/*  unsigned int *error_code :: Error code if, used if function returns false */
```

Description

PICM_CreateController allocates a memory block for storing controller parameters for communicating with your Princeton Instruments detector system. This must be the first PICM function called. All other PICM functions use this memory block to transfer information to and from the hardware controller.

Error Codes

```
/* ##### ERROR CODES for PICM_CreateController ##### */

#define CREATE_CONTROLLER_ERROR    0x0001 /* Controller object could not be   */
                                          /* created, no further operation is */
                                          /* allowed.                         */

#define DETECTOR_TYPE_ERROR        0x0002 /* Error occurred in setting given  */
                                          /* detector type (i.e. CCD or PDA   */
                                          /* illegal).                        */
                                          /* Check to see if detector and     */
                                          /* controller are legal combination/

#define DATA_COLLECTION_MODE_ERROR 0x0004 /* Error occurred in setting data   */
                                          /* collection mode, check to see    */
                                          /* if controller and/or detector    */
                                          /* can perform this data collection */
                                          /* mode.                            */

#define DATA_GEOMETRY_ERROR        0x0008 /* error in default setting of      */
                                          /* flip/rotate/reverse              */

#define DATA_ACCESS_ERROR          0x0010 /* error in setting default data    */
                                          /* access type                      */

#define CLEANSCANS_ERROR           0x0020 /* error occurred trying to set     */
                                          /* default clean scans.             */

#define X_SKIPPING_ERROR           0x0040 /* error occurred while trying to   */
                                          /* assign default x skip size       */

#define SHUTTER_ERROR              0x0080 /* error occurred in either         */
                                          /* assigning a default shutter or   */
                                          /* shutter mode                     */
```

```
#define ACCESS_PATTERN_ERROR       0x0100 /* error occurred in setting default*/
                                          /* access pattern.                  */

#define TIMING_MODE_ERROR          0x0200 /* error occurred in setting default*/
                                          /* timing mode.                     */

#define EXPOSURE_ERROR             0x0400 /* error occurred in setting up     */
                                          /* default exposure.                */

#define GENERAL_ERROR              0x8000 /* Other errors involving setting   */
                                          /* up controller object failed.     */
```

Example

```
/* Create controller, assign detector, set defaults for */
/* application. */
controller_ok =  PICM_CreateController(ST138,
                                       EEV_576x384_3ph,
                                       ROM_FULL,
                                       APP_NORMAL_IMAGING,
                                       &error_code);
```

PICM_CreateControllerNvram

This function checks to see if a Princeton Instruments PCI interface card is in the system, it then looks for a controller and tries to create the controller and load default values retrieved from the controller.

Prototype

The external description can be found in `pii2cfcn.h`.

```
UINT32 PICM_CreateControllerNvram (UINT32 pci_card, UINT32 error_code);
```

Description

This function tries to read the non-volatile RAM stored in the detector head and or the controller. This will only works with PCI cards and the following controllers:

ST133 (MicroMax) Version 2 or greater;
PentaMax version 5 or greater

This routine will first check to see if a PCI card is present (PICM_FindPCICards). If a Princeton Instruments PCI card is found this routine will then find out what controller is hooked to the PCI card (`PICM_FindController`). If a controller is found, it then loads the default values stored in nvram in the controller/detector head(`PICM_LoadNvramDefaults`). This is equivalent to calling the functions `PICM_CreateController`, `PICM_Set_controller_version`, and `PICM_SetInterfaceCard`.

Example

```
UINT32 num_cards, status, error_code;
/* look for Princeton Instruments PCI interface cards */
Num_cards = PICM_FindPCICards (void);
/* if we found any cards create a controller */
if (Num_cards > 0)
{
   /* create controller for 1st card found */
```

```
    status =  PICM_CreateControllerNvram (1,  &error_code);
    if (!status)
       printf(" error = %d\n", error_code);
    else /* we are ready to initialize system and collect data. */
    {
          go_initialize_system_and_collect_data();
    }
}
```

PICM_Download_MultROI

Causes regions of interest to be downloaded.

Prototype

```
UINT32 PICM_DownLoad_MultROI(UINT32 *error_code );
```

Description

Applies the regions of interest set with the function `PICM_SetROI_MultStrip` and downloads the regions to the hardware.

Example

See `PICM_Clear_MultStrip`

PICM_Easy_Focusing

Sets up the fast focusing mode parameters, i.e. zoom factor, gains, offsets and the like.

Prototype

```
#include "pifcsdef.h"
#include "pifcsfcn.h"
UINT32  PICM_Easy_Focusing(   int focus_type,
                              double nominal_exposure,
                              int zoom,
                              int pan,
                              double *actual_exposure,
                              int *zoompattern,
                              unsigned int *error_code );
```

Description

This function sets up all of the parameters used in the fast focusing to a video monitor. It can be called before or after `PICM_Initialize_RS170`, and even after the controller is started, meaning you can change these parameters on the fly.

```
focus_type - BINNING_FOCUS - Brighter contrast as pixels are added together
           - DECIMATION_FOCUS – Darker image as pixels are dropped out
nominal_exposure  - desired exposure by the user.
zoom    - ZX1, ZX2, ZX4 zooming by 1x 2x or 4x
pan     - when zoom is not x1, position looking at
PAN_TL = 1,     /*  Top Left       */
          PAN_TC,         /*  Top Center    */
          PAN_TR,         /*  Top Right      */
```

```
            PAN_CL,          /*  Center Left    */
            PAN_CC,          /*  Center Center  */
            PAN_CR,          /*  Center Right   */
            PAN_BL,          /*  Bottom Left    */
            PAN_BC,          /*  Bottom Center  */
            PAN_BR,          /*  Bottom Right   */
actual_exposure – returned exposure that the controller is capable of.
zoompattern  - returns the zoom that the controller and detector are capable of.
error_code -  returns the error codes, see file pifcsdef.h for list of error
codes.
```

Example

```
void main()
{
   UINT32 error;
   UINT32 zoom;
   double exposure;

   ..// Create the controller
   PICM_Initialize_RS170( &error )

   PICM_Start_controller();

   PICM_Easy_Focusing( BINNING_FOCUS, 0.25, ZX1, PAN_CC, &exposure, &zoom,
&error );

}
```

PICM_FindController

This routine tries to read the NVram of the controller and then returns the enumerated type of the controller found.

Prototype

The external description can be found in `pii2cfcn.h`.

```
UINT32 PICM_FindController (UINT32 pci_number, UINT32 *error_code);
```

Description

This routine tries to read the NVram of the controller and then returns the enumerated type of the controller found. If a controller was not found the enumerated type returned is `NO_CONTROLLER (0)`. The enumerated types can be found in `pigendef.h`.

Example

```
/* check to see if the controller is a PentaMax which is the enumerated type
DC131 */
If (PICM_FindController(1, &error_code) == DC131)
{
   printf("we found a PentaMax controller\n");
}
```

PICM_FindPCICards

Search computer for all Princeton Instruments PCI interface cards. Return the number found.

Prototype

The external description can be found in `pii2cfcn.h`.

```
UINT32  PICM_FindPCICards (void);
```

Description

This function checks all the PCI slots in the computer system and counts the number of Princeton Instruments cards found. It then returns the number of cards found. The value returned is a 32 bit unsigned integer. If 0 is returned then now PCI cards where found in the system.

Example

```
UINT32 num_cards, status, error_code;
/* look for Princeton Instruments PCI interface cards */
Num_cards = PICM_FindPCICards (void);
/* if we found any cards create a controller */
if (Num_cards > 0)
{
   /* create controller for 1st card found */
   status =  PICM_CreateControllerNvram (1,  &error_code);
   if (!status)
      printf(" error = %d\n", error_code);
   else /* we are ready to initialize system and collect data. */
   {
      go_initialize_system_and_collect_data();
   }
}
```

PICM_Generate_sideeffects

Forces some restrictions around multiple regions of interest.

Prototype

```
UINT32 PICM_Generate_sideeffects( void );
```

Description

The hardware is limited to certain regions of interest, for reasons such as binning over different regions with different binning parameters. So side effects are need in some cases to make sure that the data comes into the computer in a format which can easily be handled.

Example

See `PICM_Clear_MultStrip`

PICM_Get_acqmode

Returns the current data acquisition mode

Prototype

```
INT32 PICM_Get_acqmode(void);
```

Description

`PICM_Get_acqmode` will return the current data acquisition mode.

Example

```
mode = PICM_Get_acqmode();
```

PICM__Get_Actual_Temperature

Returns the actual temperature of the detector.

Prototype

`double PICM_Get_Actual_Temperature(void);` **Note:** For St133 and MicroMax only.

Description

`PICM_Get_Actual_Temperature` is used to read the actual temperature that the detector is at. This function is only for use with ST133 and MicroMax).

Example

```
temperature = PICM_Get_Actual_Temperature();
```

PICM_Get_AutoStop

Gets the current Autostop setting

Prototype

int PICM_GetAutoStop(void);

Description

PICM_GetAutoStop returns the current setting AutoStop

Example

numframes = PICM_GetAutoStop();

PICM_Get_cleanscans

Returns the number of cleanscans

Prototype

```
INT32 PICM_Get_cleanscans(void );  (see piclnfcn.h)
```

Description

`PICM_Set_cleanscans` returns the current number of cleanscans to be done after `PICM_Start_controller` is called.

Example

```
cleans = PICM_Set_cleanscans( );
```

PICM_Get_controller_version

Returns the version number from the DLL.

Prototype

```
int PICM_Get_controller_version (void);
```

Description

`PICM_Get_controller_version` returns the version number. The related function, `PICM_Set_controller_version` sets the version number. Note that not all controllers store the version number (i.e. PentaMAX), so the user must set it to take advantage of new hardware features (i.e. hardware look-up table). See `piverfcn.h` for external definitions.

Example

```
ControlVersion = PICM_Get_controller_version();
```

PICM_GetEnumParam

Returns the enumerated value for a given CM_CMD

Prototype

```
UINT32  PICM_GetEnumParam(enum CM_CMD cmd, UINT32 index, INT32 *enumvalue); (see
pivarfcn.h, pivardef.h)
```

Description

`PICM_GetEnumParam` is given a `CM_CMD` and an index and returns the enumerated value for that `CM_CMD` and index as a parameter, it also returns TRUE from the function if there was a value associated with the enumerated value, if no value found or illegal cmd/index was sent then FALSE returned. For example if there are 3 valid items for the controller and you pass the index 1-3 the value returned will be TRUE and the numerated

value that is supported will be returned as a parameter, if you pass 4 or greater then FALSE (0) will be returned by the function.

Example

See PICM_IsAvail

PICM_GetEnumString

Returns a string associated with a given enumerated value and ID

Prototype

```
UINT16 PISTDAPI PICM_GetEnumString( enum CM_CMD cmd, UINT32 enumvalue, char
*RetString, UINT32 StringSize ); (see pivarfcn.h, pivardef.h)
```

Description

PICM_GetEnumString is given a enumerated value (from PICM_GetEnumParam) and an ID this function returns the string associated with the two as a param, for example the ID might be ADC_TYPE and the Enum might be ADC500, then the string would be "500 kHz". If the return value of the function is False (0) if no string could be returned.

Example

See PICM_IsAvail

PICM_Get_MinBlk

Returns the current MinBlk setting

Prototype

```
INT32 PICM_Get_MinBlk(void);
```

Description

PICM_Get_MinBlk returns the current setting of MinBlks.

Example

```
minblk = PICM_Get_MinBlk();
```

PICM_Get_normal_rois

When using multiple regions with sideffects this gives you regions coordinates into the data buffer.

Prototype

```
IRCT* PICM_Get_normal_rois(   IRCT *irct,
                              INT32 *xbin,
                              INT32 *ybin,
                              INT32 index);
```

Description

Returns the IRCT * which corresponds to the user defined region based on the index parameter, of the data into the buffer returned.

Example

```
See PICM_Clear_MultStrip
```

PICM_Get_NumMinBlk

Returns the current NumMinBlk setting

Prototype

```
INT32 PICM_Get_NumMinBlk(void );
```

Description

`PICM_Get_NumMinBlk` returns the current setting for the number of minblocks.

Example

```
num_minblks =  PICM_Get_NumMinBlk();
```

PICM_Get_num_strips_per_clean

Gets the current setting number of strips per clean

Prototype

```
INT32  PICM_Get_num_strips_per_clean (void);
```

Description

`PICM_Get_num_strips_per_clean` returns the current setting for the number of strips used for each cleanscan.

Example

```
strips = PICM_Get_num_strips_per_clean ();
```

PICM_Get_pixeldimension_x

PICM_Get_pixeldimension_y

Returns the X or Y dimension of data returned in a single frame, taking region of interest and binning into account.

Prototype

```
int PICM_Get_pixeldimension_x(void);
int PICM_Get_pixeldimension_y(void);
```

Description

`PICM_Get_pixeldimension_X` and `PICM_Get_pixeldimension_y` simply return the number of pixels in the X or Y direction that will be collected in the next frame of data. These values reflect the effects of setting region of interest and binning.

Example

```
/* Size of data coming back from the controller */
sizeofdata = PICM_Get_pixeldimension_x() * PICM_Get_pixeldimension_y() *
             bytes_per_pixel;
```

PICM_Get_RS170_enable

Returns whether or not the RS170 has been turned on or not.

Prototype

```
#include "pifcsfcn.h"
UINT32 PICM_Get_RS170_enable( void );
```

Description

This function simply returns whether or not the RS170 enable bit has been set.

Example

```
void main()
{
    ..// Create the controller, Setup the interface card etc….

    /* Turn of the Video output of the controller */
    if ( PICM_Get_RS170_enable() )
        printf( "RS170 has been turned on" );
    else
        printf( "RS170 has been turned off" );
    ..// Initialize the system start the controller
}
```

PICM_Get_sensor_x

PICM_Get_sensor_y

Returns the X or Y dimension of the selected CCD.

Prototype

```
int _export FAR PASCAL PICM_Get_sensor_x(void);
int _export FAR PASCAL PICM_Get_sensor_y(void);
```

Description

`PICM_Get_sensor_x` and `PICM_Get_sensor_y` simply return the number of pixels in either the X or Y direction on the CCD selected by the `Detector_type` parameter of `PICM_CreateController`.

Example

```
/* Total dimension of chip in bytes */
dimension = PICM_Get_sensor_x()*PICM_Get_sensor_y()*bytes_per_pixel;
```

PICM_Get_shuttermode

Gets the current shuttermode.

Prototype

`INT32 PICM_Get_shuttermode(void); (see pishtfcn.h, pishtdef.h)`

Description

`PICM_Get_shuttermode` returns the current shutter mode. The default mode is `SHUTTER_NORMAL.`

Example

```
mode = PICM_Get_shuttermode();
```

PICM_Get_shutter_type

Gets the current shutter type for shutter compensation time

Prototype

`INT32 PICM_Get_shutter_type(void); (see pishtfcn.h, pishtdef.h)`

Description

`PICM_Get_shutter_type` returns the current setting of shutter type.

Example

```
type = PICM_Get_shutter_type();
```

PICM_Get_Temperature

Returns the current temperature setting.

Prototype

`double PICM_Get_Temperature( void );` **Note:** For St133 and MicroMax only.

Description

`PICM_GetTemperature` only returns the software variable set by PICM_Set temperature. It does not return a temperature measurement. Use PICM_Get_Actual_Temperature for getting the actual temperature from the detector.

Example

```
temperature =  PICM_Get_Temperature();
```

PICM_Get_Temperature_Status

Gets the current temperature lock status of the CCD.

Prototype

`int PICM_Get_Temperature_Status( void );` **Note:** For ST-133 and MicroMax only.

Description

`PICM_Get_Temperature_Status` returns FALSE if detector temperature is locked and TRUE if the detector temperature is not locked to its programmed value.

Example

```
Set_Temperature(temperature );
while(!PICM_Get_Temperature_Status( ));
```

PICM_Get_TTL_pattern

Returns the bit pattern from the controller's TTL input port.

Prototype

```
int PICM_Get_TTL_pattern(void);
```

Description

Get TTL pattern reads the controller's TTL input port and returns it in integer form.

Example

```
Value_in = PICM_Get_TTL_pattern();
```

PICM_Initialize_RS170

Initializes the controller for high speed RS170 output.

Prototype

```
#include "pifcsfcn.h"
```

```
UINT32 PICM_Initialize_RS170( UINT32 *error );
```

Description

This function enables the high speed video focusing mode, at this time no data is coming back into the computer it is only going to the RS170 monitor. *Do **not** call* `PICM_Initialize_System` when using this function.

Example

```
void main()
{
   UINT32 error;

   // Create the controller
   PICM_Initialize_RS170( &error );
   if ( !error )
      // Start the controller
}
```

PICM_Initialize_System

Transmits all of the parameters that were defined with previous functions in the software controller object to the hardware controller.

Prototype

```
/* int PICM_Initialize_System                                              */
/*   void huge* big_buffer    :: Users data buf, Data collected stored here  */
/*   unsigned int *error_code :: Error code, used if function returns false. */
```

Description

`PICM_Initialize_System` sends the parameters stored in the software controller object to the hardware controller to setup the camera system.

Note: This function also allocates a ring buffer in the computer's memory to store the data as it is collected.

Error Codes

```
ERROR CODES for PICM_Initialzie_System ##### */

#define INITIALIZE_ERROR        0x0001 /* Error occurred while trying to    */
                                       /* initialize the controller.        */

#define MEMORY_ALLOCATION_ERROR 0x0002 /* Error occurred in trying to       */
                                       /* allocate ring buffer              */

#define INTERFACE_INIT_ERROR    0x0004 /* Error occurred in initialize data */
                                       /* collection interface (check dma)  */
```

Example

```
controller_ok = PICM_Initialize_System(1pvBuffer,&error_code);
```

PICM_IsAvail

Tells what is available for the hardware selected.

Prototype

`UINT32 PICM_IsAvail(enum CM_CMD cmd, struct VALID_RANGE *MinMaxDefault); (see pivarfcn.h, pivardef.h)`

Description

PICM_IsAvail returns 0 if feature is unavailable for a given controller object, otherwise it returns non-zero. The Isavail functions tell what is possible in the current software and hardware settings. As parameters change, so does what is returned by the IsAvail function. Note the datatype definitions are in pitypes.h(X_INT). Some parameters that change or have the potential to change most of the IsAvail functions are :

1. changing the controller type.
2. changing the chip type.
3. changing the version.
4. changing the ADC (fast/slow).

Example

This function will decide (via IsAvail calls) the state and contents of a combo box. If the item can be "Set", the combo box is enabled and loaded with the appropriate strings and values. If the item can only be "Get", the combo box is visible but disabled. The calling routine must load the current value. If neither of the above is valid, the control will be hidden.

```
BOOL DoIsAvailComboBox(
    CComboBox *pComboCtrl,
    enum CM_CMD cm_id
    )
{
    BOOL RetVal = FALSE;

        struct VALID_RANGE VRange;
        UINT32 IsValue;

            // If Item is available, enable the control and fill it
        if( PICM_IsAvail(cm_id, &VRange ) )
        {

            IsValue = VRange.RdWrType;

            if( IsValue == CM_READ_N_WRITE )
            {
                pComboCtrl->ShowWindow( SW_SHOW );
       RetVal = PutIsAvailIntoComboBox( pComboCtrl, VRange,
                                                    cm_id, Controller );

                    // Disable window if 1 or less items in it
                if( VRange.NumberOfItems > 1 )
                    pComboCtrl->EnableWindow( TRUE );
                else
```

```
                        pComboCtrl->EnableWindow( FALSE );
                    }
                    else if( IsValue == CM_READ_ONLY )
                    {
                         // Can only read the item but still want to load the list
                         // of values available so the current one can be shown.
                         // Disable the control but keep it visible.
                        pComboCtrl->ShowWindow( SW_SHOW );
                        pComboCtrl->EnableWindow( FALSE );
                        RetVal = PutIsAvailIntoComboBox( pComboCtrl, VRange,
                                                         cm_id, Controller );
                        RetVal = TRUE;
                    }
                    else
                    {
                            // Item is not available at all so hide the control
                        pComboCtrl->ShowWindow( SW_HIDE );
                    }
                }
                else
                {
                        // Item is not available at all so hide the control
                    pComboCtrl->ShowWindow( SW_HIDE );
                }

    return( RetVal );
}

////////////////////////////////////////////////////////////////////////////
//
//  DoIsAvailControllers
//
//  This function will load the IsAvail values and strings for
//  Controllers Available into the Combo Box.
//
//  Returns: TRUE  = Successful Loading of Combo Box
//           FALSE = Data Type is X_NODATATYPE so nothing to load
//
////////////////////////////////////////////////////////////////////////////

BOOL DoIsAvailControllers(
    CComboBox *pComboCtrl
    )
{
    BOOL RetVal = FALSE;
    struct VALID_RANGE VRange;

         // If Item is available, enable the control and fill it
    if( PICM_IsAvail( CMP_CONTROLLERS_SUPPORTED, &VRange ) )
    {
        char enum_string[MAX_LISTBOX_STRING_SIZE];
        UINT32 idx;
        INT32 enum_value;

        if( VRange.DataType == X_ENUM )
        {
            pComboCtrl->ResetContent();     // First, clear out everything

            for( idx = 0; idx < VRange.NumberOfItems; idx++ )
            {
                if( PICM_GetEnumParam( CMP_CONTROLLERS_SUPPORTED,
```

```
                                                    idx, &enum_value ) )
                    {
                        PICM_GetEnumString( CMP_CONTROLLERS_SUPPORTED,
                                            enum_value, enum_string,
                                            MAX_LISTBOX_STRING_SIZE );

                       // Add the string and get back an index (position)
                       // Set the data item for this string
                       pComboCtrl->SetItemData(
                       pComboCtrl->AddString( enum_string ), enum_value );
                    }
                }

                RetVal = TRUE;
            }
        }
        return( RetVal );
    }

    ////////////////////////////////////////////////////////////////////////
    //
    //  PutIsAvailIntoComboBox
    //
    //  This function will load the IsAvail values and strings into
    //  the Combo Box.
    //
    //  Returns: TRUE  = Successful Loading of Combo Box
    //           FALSE = Data Type is X_NODATATYPE so nothing to load
    //
    ////////////////////////////////////////////////////////////////////////

    BOOL PutIsAvailIntoComboBox(
        CComboBox *pComboCtrl,      // ComboBox to display values into
        struct VALID_RANGE VRange,  // IsAvail Range structure
        enum CM_CMD id,             // Valid "IsAvail" id
        CONTROLLER *Controller      // Controller object to get values from
        )
    {
        char enum_string[MAX_LISTBOX_STRING_SIZE];
        UINT32 idx;
        INT32 enum_value;
        BOOL RetVal = TRUE;

        if( ( VRange.DataType == X_ENUM ) && ( Controller != NULL ) )
        {
            pComboCtrl->ResetContent();     // First, clear out everything

            for( idx = 0; idx < VRange.NumberOfItems; idx++ )
            {
                if( PICM_GetEnumParam( id, idx, &enum_value ) )
                {
                    PICM_GetEnumString( id, enum_value, enum_string,
                                        MAX_LISTBOX_STRING_SIZE );

                          // Add the string and get back an index (position)
                          // Set the data item for this string
                    pComboCtrl->SetItemData(
                           pComboCtrl->AddString( enum_string ), enum_value );
                }
            }
        }
```

```
    else if( VRange.DataType != X_NODATATYPE )
    {
        double value;
        CString strValue, strFormat;
        INT FormatId;

        pComboCtrl->ResetContent();     // First, clear out everything

        switch( VRange.DataType )
        {
            case X_INT:
            case X_BYTE:
                FormatId = ESU_DSTRING;
                break;

            case X_LONG:
                FormatId = ESU_LSTRING;
                break;

            case X_UNSIGNED_INT:
                FormatId = ESU_UISTRING;
                break;

            case X_UNSIGNED_LONG:
                FormatId = ESU_ULSTRING;
                break;

            case X_FLOAT:
            case X_DOUBLE:
                FormatId = ESU_GSTRING;
                break;
        }

        strFormat.LoadString( FormatId );

        for( idx = 0, value = VRange.MinValue;
                        idx < VRange.NumberOfItems;
                                    idx++, value += VRange.Increment )
        {
            strValue.Format( strFormat, value );
            pComboCtrl->SetItemData( pComboCtrl->AddString( strValue ),
                                      (int)value );
        }
    }
    else
        RetVal = FALSE;

    return( RetVal );
}

////////////////////////////////////////////////////////////////////////
//
//  SetUpSlider
//
////////////////////////////////////////////////////////////////////////

int SetUpSlider( VALID_RANGE *mmd, CSliderCtrl *slider)
{
   int status = FALSE;
   long max, position;
```

```
    /* Check For Valid Objects */
    if ( (mmd!=NULL) && (slider!=NULL))
    {
        max = (long)( (mmd->MaxValue - mmd->MinValue)/ mmd->Increment );

        /* Positive Range */
        if ( mmd->MinValue >= 0 )
            position = (long)( mmd->CurrentValue / mmd->Increment );
        /* Negative To Positive Range exple: temperature -25 to 26 */
        else
            position = (long)( ( mmd->CurrentValue - mmd->MinValue) / mmd-
>Increment );

        slider->SetRangeMin( 0, TRUE );
        slider->SetRangeMax( max, TRUE );
        slider->SetPos(  position );

        status = TRUE;
    }
    return (status);
}
```

PICM_LoadNvramDefaults

Loads default values from the controller nvram (non-volatile ram) and/or the detector head.

Prototype

The external description can be found in `pii2cfcn.h`.

```
UINT32 PICM_LoadNvramDefaults (UINT32 *error_code);
```

Description

Loads default values from the nvram (non-volatile ram) of the controller and/or the detector head. This will only works with PCI cards and the following controllers:

ST133 (MicroMax) Version 2 or greater;
PentaMax version 5 or greater.

The following defaults are loaded : CCD, controller version, readout mode, video type (NTSC or PAL), shutter type, ADC offset, and hardware specials.

Example

```
UINT32 status, error_code;
/* Load NvRam defaults from hardware, check if error (false) */
if (PICM_LoadNvramDefaults(&error_code))
{
    /* we successfully loaded the defaults */
}
else /* we got an error, print out error code */
{
    printf(" error code = %d\n", error_code);
}
```

PICM_LockCurrentFrame

Returns the address and size of the frame currently locked in the dma buffer.

Note: See PICM_ChkData function description for status definitions.

Prototype

```
int  PICM_LockCurrentFrame(void **directaddress, unsigned long *size, unsigned
int *status);
```

Description

`PICM_LockCurrentFrame` returns true then address and size are valid. Only one frame at a time can be locked in the dma buffer. `PICM_UnlockCurrentFrame` must be called before subsequent frames can be locked.

`PICM_LockCurrentFrame` returns FALSE until data collection is complete, at which point it will return TRUE. By examining the status of each bit in `status_code` you can also check other information about the controller:

```
#define RUNNING         1   /* bit for "experiment running"              */
#define WAITING         2   /* bit for "waiting for start event"         */
#define CONERROR        4   /* bit for "controller error"                */
#define COMERROR        8   /* bit for "command error"                   */
#define NEWDATARDY     16   /* bit for "new data block received"         */
#define INITERROR      32   /* bit for "no initialization done"          */
#define NEWDATAFIXED   64   /* bit for "dma_copy_buffer -> to new data"  */
#define DATAOVERRUN   128   /* bit for "DMA data overrun"                */
#define VIOLATION     256   /* bit for "TAXI violation"                  */
#define MAILERROR     512   /* bit for EISA mail box communication error */
#define XFERERRORCH0 1024   /* bit for Channel zero xfer not enabled.    */
#define XFERERRORCH1 2048   /* bit for Channel one  xfer not enabled.    */
#define DONEDCOK     4096   /* bit for done data collection.             */
#define INITIALIZED  8192   /* bit to say controller has been initialized*/
```

Example

```
ok = PICM_Start_controller();
while( !PICM_LockCurrentFrame( &lpvBuffer, &size, &status) );
```

PICM_ResetUserBuffer

Resets the user data buffer.

Prototype

```
void  PICM_ResetUserBuffer (void);
```

Description

`PICM_ResetUserBuffer` will reset the internal data buffer pointer to the beginning of the user buffer that a pointer was passed to in the `PICM_InitialeSystem` call. When collecting multiple frames either this function or `PICM_SetNewUserBuffer` must be called between frames.

Example

```
for ( LoopPre = 0; LoopPre < 10; ++LoopPre )
{
   done_flag = 0;
   PICM_Start_controller();
   /*Wait for data to be collected. */
   while ( !done_flag )
      data = PICM_ChkData( &status );
   Update_Bitmap( gwnd, (unsigned short *)lpvBuff , TRUE );
   PICM_ResetUserBuffer();
   }
```

PICM_Set_acqmode

Sets the acquisition mode for data collection.

Prototype

```
INT32  PICM_Set_acqmode(INT32 acqmode)
```

Description

`PICM_Set_acqmode` set the data acquisition mode. The default mode is `CTRL_FREERUN`. See Appendix C for specific timing mode information.

Example

```
PICM_Set_acqmode(CTRL_FREERUN);
```

PICM_Set_AutoStop

Sets the controller to stop after a number of frames.

Prototype

int PICM_set_AutoStop(int frames);

Description

PICM_set_AutoStop sets the controller to stop after a number of frames. For the ST133, MicroMax, and PentaMAX the range is 0 to 255. For the ST138, only 0 or 1 is allowed. Setting AutoStop to 0 will put the controller into synchronous mode and will run continuously until PICM_StopController is called.

Example

```
frames = 10;
PICM_set_AutoStop(frames);
```

PICM_Set_controller_version

Sets the controller version number.

Prototype

```
int PICM_Set_controller_version (int cversion)
```

Description

This function sets the version number. Not all controllers store the version (i.e. PentaMAX), so the user must set this parameter to take advantage of new hardware features, such as the hardware look-up table. The related function, `PICM_Get_controller_version` returns the version number from the DLLs. See `piverfcn.h` for external definitions.

Example

```
/* Should be done for ST-133 (SpectroMAX or MicroMAX) and DC-131 (PentaMAX) */
PICM_Set_controller_version (5);
```

PICM_Set_cleanscans

Sets the number of cleans to done

Prototype

```
INT32  PICM_Set_cleanscans(INT32 cleans);
```

Description

`PICM_Set_cleanscans` sets the number of cleans to be done when a start controller is called.

Example

```
cleans = 1;
PICM_Set_cleanscans(cleans);
```

PICM_Set_EasyDLL_DC

Sets the data collection mode

Prototype

```
INT32 PICM_Set_EasyDLL_DC (INT32 mode);
```

Description

`PICM_Set_EasyDLL_DC` sets the data collection mode to run in `EASYDLL_FOCUS_MODE` or `EASYDLL_NFRAME-MODE`. In `EASYDLL_FOCUS_MODE` mode the most recent frame will be retrieved from the dma buffer when `PICM_ChkData` or `PICM_LockCurrentFrame` returns a true. This means frames will be skipped if frames are coming into the buffer faster than they can be retrieved. It is impossible to get a data overrun. In `EASYDLL_NFRAME-MODE` mode all frames will be retrieved from the buffer. If there is no more room in the dma buffer a data overrun occurs and no more data will be stored in the buffer. All frames that were stored in the buffer up to the point when the data overrun can still be retrieved.

Example

```
PICM_Set_EasyDLL_DC (EASYDLL_NFRAME-MODE);
```

PICM_SetExposure

Sets the exposure time for the detector.

Prototype

int PICM_SetExposure (double exposure, unsigned int *error_code);

Description

PICM_SetExposure sets the exposure time in seconds. This function must be called before calling `PICM_Initialize_System`. Any change to the exposure time after initializing the system will require calling `PICM_Initialize_System` for the change to take effect.

Example

```
exposure = 0.1;
PICM_SetExposure (exposure);
```

PICM_Set_Fast_ADC

Selects the faster of a controller's two analog to digital converters.

Prototype

```
int PICM_Set_Fast_ADC(void);
```

Description

Set fast ADC activates the faster ADC on controllers with two ADC's. This function has no effect on controllers with only one ADC.

Example

```
PICM_Set_Fast_ADC ();
```

PICM_SetInterfaceCard

Defines computer-to-controller interface card.

Note: See Chapter 5 for list of valid interfaces types.

Prototype

```
/* int PICM_SetInterfaceCard */
/*   int Interface_Card       :: Interface card, see interfaceType enum       */
/*   unsigned int Base_Address:: base add. of interface card, not used eisa   */
/*   int Card_interrrupt      :: Interrupt to use, see interrupt_channel enum */
/*   unsigned int *error_code :: Error code, used if function returns false.  */
```

Description

`PICM_SetInterfaceCard` specifies which computer interface card PICM will be communicating through. Pass the correct identifier for the interface that you are using and set the correct address and, if needed, interrupt.

Errors

```
#define INTERFACE_TYPE_ERROR   0x0001 /* illegal interface                  */

#define BASE_ADDRESS_ERROR     0x0002 /* illegal base address               */

#define QUERY_CONTROLLER_ERROR 0x0004 /* error occurred talking to controller*/

#define ATOD_CONVERTER_ERROR   0x0008 /* an illegal A to D converter was    */
                                      /* entered. */

#define GAIN_MULTIPLIER_ERROR  0x0010 /* an illegal gain multiplier was set */

#define DATA_CLIP_ERROR        0x0020 /* an illegal data clip value was set */

#define CONTROLLER_SPEED_ERROR 0x0040 /* A illegal controller speed was     */
                                      /* entered (check valid speeds for    */
                                      /* ADCs).                             */

#define NEEDS_EISA_COMPUTER    0x0080 /* The fastest speeds this controller */
                                      /* can run need a EISA computer. If   */
                                      /* controller supports slower speeds  */
                                      /* you can try to set these manually  */
                                      /* this function will try defaulting  */
                                      /* to the slowest.                    */
                                      /* IF you have an EISA computer and a */
                                      /* PI ISA interface card and a 1 MHz  */
                                      /* controller try interface type ::   */
                                      /* TAXI_TypeB_Interface (Note this is */
                                      /* only good on an EISA computer and  */
                                      /* only up to 1 MHz).                 */

#define NEEDS_HIGH_SPEED_CARD  0x0100 /* Fastest speeds of controller need  */
                                      /* an EISA computer and a EISA PI     */
                                      /* interface card.                    */

#define IRQ_ERROR 0x0200              /* Error occurred in IRQ setting.     */
```

Example

```
controller_ok = PICM_SetInterfaceCard(
                                        TAXI_Interface,  /* PC interface card    */
                                        0xA00,           /* Base address of card */
                                        CHANNEL_10,      /* Interrupt used       */
                                        &error_code);    /* error codes.         */
```

PICM_Set_MinBlk

Sets the number of invalid strips to group right before the valid data

Prototype

```
INT32  PICM_Set_MinBlk(INT32 minblk);
```

Description

`PICM_Set_MinBlk` sets the number of strips to group together of the invalid data that is before the valid data.

Example

```
minblk = 2;
PICM_Set_MinBlk(minblk);
```

PICM_Set_MultiStrip_Flag

Tells the dlls to use multiple regions of interest for acquisitions.

Prototype

```
INT32 PICM_Set_MultStrip_Flag(INT32 flag );
```

Description

If the flag is set to TRUE then the Dlls process multiple regions set in `PICM_SetROI_MultiStrip`, else Process single region.

Example

See `PICM_Clear_MultStrip`

PICM_SetNewUserBuffer

Allows user to use multiple buffers for data collection

Prototype

```
PICM_SetNewUserBuffer( void huge* big_buffer,  unsigned INT32 *error_code );
```

Description

PICM_SetNewUserBuffer sets the internal data buffer pointer to a new pointer supplied by this routine. When collecting multiple frames either this function or PICM_ResetUserBuffer must be called between frames.

Example

```
buffer_size = PICM_SizeNeedToAllocate() * numframes;  // allocate a user buffer
for numframes
pixels = PICM_Get_pixeldimension_x () * PICM_Get_pixeldimension_y();
tempptr = (unsigned short *)lpvBuffer;
controller_ok = PICM_Initialize_System(lpvBuffer, &error_code);

for (i = 0; i < numframes;  i++)
{
   while (!done_flag)
      done_flag = PICM_ChkData(&status);

   tempptr += pixels;  // increment the pointer  for the next frame
   PICM_SetNewUserBuffer(tempptr, &error_code);
   done_flag = FALSE;
}
```

PICM_Set_NumMinBlk

Sets the number of Min Blocks to do before going to a geometric grouping algorithm.

Prototype

```
INT32 PICM_Set_NumMinBlk( INT32 minblk );
```

Description

PICM_Set_NumMinBlk set the number of Min Blocks to do before going to a geometric grouping algorithm when doing regions of interest.

Example

```
numminblk = 5
PICM_Set_NumMinBlk( minblk );
```

PICM_Set_num_strips_per_clean

Set the number of strips to be associated with each PICM_Set_cleanscan

Prototype

```
INT32  PICM_Set_num_strips_per_clean (INT32 number_strips_per_clean);
```

Description

PICM_Set_num_strips_per_clean set the number of strips used for each cleanscan.

Example

```
strips = PICM_Get_sensor_y();
```

```
PICM_Set_num_strips_per_clean (strips);
```

PICM_SetROI

Sets a region of interest for data acquisition

Prototype

```
int PICM_SetROI(
   int startx,          //  first pixel of ROI in x dir.(note: starts at 1)
   int starty,          //  first pixel of ROI in y dir.(note: starts at 1)
   int endx,            //  last pixel of ROI in x dir.
   int endy,            //  last pixel of ROI in y dir.
   int groupx,          //  amount to bin/group x data.
   int groupy,          //  amount to bin/group y data.
   unsigned int *error_code //  error code, used if function returns false.
```

Description

PICM_SetROI selects a region of interest and the binning parameters for data acquisition. This function must be called before calling `PICM_Initialize_System`. Any ROI changes after initializing the system will require calling `PICM_Initialize_System` for those changes to take effect.

Example

```
startx = 100;
starty = 200;
endx = 300;
endy = 400;
groupx = 2;
groupy = 2;
ok = PICM_SetROI( startx, starty, endx, endy, groupx, groupy, &error_code);
```

Error Codes

```
#define CONTROLLER_SETUP_WRONG 0x0001 // Error in getting info from controller,
check to
                                  //see if previous initialize/create commands
                                  // executed without error. Note : ROI not set.

// The following 4 messages are for illegal values for ROI, the function
// PICM_SetROI will try to correct, but user should check code.
#define STARTX_ILLEGAL         0x0004
#define STARTY_ILLEGAL         0x0002
#define ENDY_ILLEGAL           0x0008
#define ENDX_ILLEGAL           0x0010

// The following 2 warnings are start and end values where reversed. The
function swapped
//start end but code should be checked.
#define X_VALUES_SWAPPED       0x0020
#define Y_VALUES_SWAPPED       0x0040

// The following 2 errors are for illegal group/bin sizes.
#define GROUPSIZE_X_ERROR      0x0080
#define GROUPSIZE_Y_ERROR      0x0100
```

```
#define ROI_ERROR              0x0200  // The ROI defined was not accepted by
                                          controller object.
                                       // Check ROI values make sure valued for
                                          CCD and controller.
```

PICM_SetROI_MultiStrip

Sets the regions of interest in multiple region acquisitions.

Prototype

```
INT32 PICM_SetROI_MultiStrip(  INT32 startx, /* 1st pixel of ROI in x dir. (note
starts at 1)  */
 INT32 starty,            /* 1st pixel of ROI in y dir. (note starts at 1) */
 INT32 endx,              /* last pixel of ROI in x dir.                   */
 INT32 endy,              /* last pixel of ROI in y dir.                   */
 INT32 groupx,            /* amount to bin/group x data.                   */
 INT32 groupy,            /* amount to bin/group y data.                   */
 INT32 index,             /* index of strip 1-50.                          */
  UINT32 *error_code      /* Error code, used if function return false.    */);
```

Description

Sets up a multiple region of interest acquisition, this does not download the controller with these regions it only stores them in an array. To download the controller use `PICM_DownLoad_MultROI` after calling this function.

Example

See `PICM_Clear_MultStrip`

PICM_Set_RS170_enable

Enables RS170 output of the controller.

Prototype

```
#include "pifcsfcn.h"
UINT32 PICM_Set_RS170_enable(int mode);
```

Description

Enables the RS170 output of the controller in either NTSC or PAL formats depending on the hardware. This should be called before `PICM_Initialize_System`.

Example

```
void main()
{
   …// Create Controller, Set Interface etc..

   /* Allow data to go to video screen as well */
   mode = 1;
   PICM_Set_RS170_enable(mode);

   ..// Initialize System, Start Controller
}
```

PICM_Set_shutter

Sets the shutter mode online.

Prototype

```
INT32 PICM_Set_shutter(INT32 condition); (see pishtfcn.h, pishtdef.h)
```

Description

`PICM_Set_shutter` sets the shutter mode of operation which tells the controller what to do when `PICMStartContoller` is called. It does not open or close the shutter directly when called. This function does not require `PICM_Initialize_System` for it to take effect.

Example

```
PICM_Set_shutter(SHUTTER_CLOSE);
```

PICM_Set_shuttermode

Sets the shutter mode.

Prototype

```
INT32 PICM_Set_shuttermode(INT32 shuttermode); (see pishtfcn.h, pishtdef.h)
```

Description

`PICM_Set_shuttermode` sets the shutter mode. This function must be called before `PICM_Initialize_System` for it to take effect.

Example

```
PICM_Set_shuttermode(SHUTTER_CLOSE);
```

PICM_Set_shutter_type

Set the shutter type for shutter compensation time.

Prototype

```
INT32  PICM_Set_shutter_type(INT32 shutter); (see pishtfcn.h, pishtdef.h)
```

Description

`PICM_Set_shutter_type` sets the shutter type for shutter compensation time. Should be set according to the type of shutter used.

Example

```
PICM_Set_shutter_type(SMALL_SHUTTER);
```

PICM_Set_Slow_ADC

Selects the slower of a controller's two analog digital converters.

Prototype

```
int PICM_Set_Slow_ADC(void);
```

Description

Set slow ADC activates the slower ADC on controllers with two ADCs. This function does not affect controllers with only one ADC.

Example

```
PICM_Set_Slow_ADC ();
```

PICM_Set_Temperature

Sets the detector temperature.

Prototype

```
int  PICM_Set_Temperature( double temperature );
```

Description

`PICM_Set_Temperature` sets the detector temperature to the specified value.

Example

```
temperature = -10.0;
PICM_Set_Temperature(temperature);
```

PICM_Set_TTL_pattern

Sets the bit pattern on the controller's TTL output port.

Prototype

```
int PICM_Set_TTL_pattern (int pattern);
```

Description

Set TTL pattern sets the controller's TTL output port to the pattern defined by the `pattern` parameter.

Example

```
/* Alternate between highs and lows */
PICM_Set_TTL_pattern (0x55);
```

PICM_SetUserEvent

This Windows 95 only function sets up a device driver to create an event that triggers after every XFrames.

Description

This function should be called in place of the `CreateEvent` function. The first four parameters of `PICM_SetUserEvent` are exactly the same as the `CreateEvent` function. The fifth parameter, which is unique to PI, is for the frequency of events being triggered. This function returns an event handle that can be used with all standard Windows 32 API functions (for instance `WaitForSingleObject` and `WaitForMultipleObjects`). For a further description on the use of events, see *Advanced Windows,* pg 364-366 , by Jeffrey Richter (MicroSoft Press).

Prototype

```
HANDLE_PICM_SetUserEvent(
    LPSECURITY_ATTRIBUTES 1pEventAttributes, // pointer to security attributes
    BOOL bManualReset,                       // flag for manual-reset event
    BOOL bInitialState,                      // flag for initial state
    LPCTSTR 1pName,                          // pointer to event-object name
    int XFrames                              // Every X frames, set an event
    );
```

Example

```
UserEvent = PICM_SetUserEvent (0,        /* Security attributes   */
                               0,        /* Manual Reset Flag     */
                               0,        /* Initial State         */
                               0,        /* Event name            */
                               1);       /* Number of Frames      */
       .
       .
       .
/* Later in the code */

/* In the data collection thread */
WaitForSingleObject (UserEvent,INFINITE);

Controller_ok = PICM_ChkData (&status);

/* Display image */
```

PICM_SizeNeedToAllocate

Returns the amount of memory needed to store a single frame of data.

Prototype

```
long PICM_SizeNeedToAllocate (void);
```

Description

PICM_SizeNeedToAllocate returns the amount of memory the program should allocate to store incoming, collected data.

Example

```
/* Allocate a buffer */
Buffer_size = PICM_SizeNeedToAllocate ();
```

PICM_Start_Controller

Initiates a data acquisition cycle.

Prototype

```
int _export FAR PASCAL PICM_Start_controller(void);
```

Description

PICM_Start_controller starts the detector system's data acquisition cycle. As the system collects data, the PICM_ChkData function can be used to poll the controller for status information. When PICM_ChkData indicates that data collection has been completed, the data will be available through the buffer pointer passed to PICM_Initialize_System.

Example

```
/*  Startup Controller & acquire data  */
PICM_Start_controller();


done_flag = FALSE;
/* Wait for data to be collected. */
while (!done_flag)
    done_flag = PICM_ChkData(&status);

if (status & DONEDCOK)
```

PICM_Stop_Controller

Deactivates all controller functions and prepares for safe shut down.

Prototype

```
int _export FAR PASCAL PICM_Stop_controller(void);
```

Description

PICM_Stop_controller should be called just before PICM_CleanUp to allow the hardware controller to perform its own internal shut down and clean up options. When PICM_Stop_controller is called, data acquisition cannot be started without calling PICM_Initialize_System again. If you are using a ST-133 (SpectroMAX or MicroMAX),

ST-138, DC-131 (PentaMAX) in the autostop mode (default), then you should not call `PICM_Stop_controller` since the hardware automatically stops.

Example

```
/* Clean up when done.      */
/* release big buffer       */
/* stop controller          */
PICM_Stop_controller();
PICM_CleanUp();
GlobalUnlock(hglb);
GlobalFree(hglb);
initialize_flag = FALSE;
```

PICM_UnlockCurrentFrame

Unlocks the currently locked frame in the dma buffer.

Prototype

```
int  PICM_UnlockCurrentFrame(void);
```

Description

PICM_UnlockCurrentFrame will unlock the frame that is currently locked in the dma buffer. This function must be called before attempting to lock another frame in the buffer. Only one frame at a time can be locked.

Example

```
PICM_UnlockCurrentFrame();
```

Chapter 12
Applications Strategies

In most cases the application will only need to call **PICM_CreateController** as well as **PICM_SetInterfaceCard** one time. This means there will only be one call to **PICM_CleanUp** when the application exits. I've seen people call **Create/Set/Initialize/Start/ChkData/Cleanup** in a function under a button Acquire in their application. This will work but will be dramatically slower then setting everything up once and just calling start controller when ever you want data. Also, calling **Create** and **CleanUp** multiple times forces the DLLs to load and unload. This shouldn't hurt the application, but again it slows it down. There are times when you do need to call **PICM_Initialize_System** more than once in an application. These occur when you make changes to the way you want the system to run. They include region of interest change, exposure change, data collection mode change, and some others. In general, whenever you call a **PICM_Setxxx** function you are probably going to have to initialize the system for the change to take effect. If you are using threads to acquire data there is one more thing you need to be sure of. If you are calling PICM functions in the thread, then the thread needs to exit to completion before you call **PICM_CleanUp**. Otherwise you will destroy the controller object that the thread is depending on, and probably have an access violation.

This page intentionally left blank.

Appendix A

Headers and Calibration Structures

Saving Data To A WinView/WinSpec File

Version 1.43 Header

All WinView or WinSpec files (version 1.43) must begin with the following 4100 byte header:

```
                typedef WINXHEAD {
0               int dioden;           /* CCD X dimension.                        */
2               int avgexp;           /* Not used by WinView                     */
4               int exposure;         /* exposure if -1 see lexpos               */
6               int datarange;        /* Not used by WinView                     */
8               int mode;             /* Not used by WinView                     */
10              float wexsy;          /* Not used by WinView                     */
14              int asyavg;           /* Not used by WinView                     */
16              int asyseq;           /* Not used by WinView                     */
18              int linefreq;         /* Not used by WinView                     */
20              int date0;            /* Not used by WinView                     */
22              int date1;            /* Not used by WinView                     */
24              int date2;            /* Not used by WinView                     */
26              int date3;            /* Not used by WinView                     */
28              int date4;            /* Not used by WinView                     */
30              int ehour;            /* Not used by WinView                     */
32              int eminute;          /* Not used by WinView                     */
34              int noscan;           /* # of stripes collected if -1 see        */
                                      /* lnoscan.                                */
36              int fastacc;          /* Not used by WinView                     */
38              int avgtime;          /* Not used by WinView                     */
40              int dmatotal;         /* Not used by WinView                     */
42              int faccount;         /* X dimension : Actual dim of image.      */
44              int stdiode;          /* Not used by WinView                     */
46              float nanox;          /* Not used by WinView                     */
50              float calibdio[10];   /* Not used by WinView                     */
90              char fastfile[16];    /* fast access file. Not used by WinView   */
106             int asynen;           /* Not used by WinView                     */
108             int datatype;         /* 0 -> float (4 byte)                     */
                                      /* 1 -> long integer (4 byte)              */
                                      /* 2 -> integer (2 byte)                   */
                                      /* 3 -> unsigned integer (2 byte)          */
                                      /* 4 -> String/char (1 byte)               */
                                      /* 5 -> double (8 bytes) Not implemented   */
                                      /* 6 -> byte (1 byte)                      */
                                      /* 7 -> unsigned byte (1 byte)             */
110             float calibnan[10];   /* Not used by WinView                     */
```

```
150          int rtanum;          /* Not used by WinView                    */
152          int astdiode;        /* Not used by WinView                    */
154          int int78;           /* Not used by WinView                    */
156          int int79;           /* Not used by WinView                    */
158          double calibpol[4];  /* Not used by WinView                    */
190          int int96;           /* Not used by WinView                    */
192          int int97;           /* Not used by WinView                    */
194          int int98;           /* Not used by WinView                    */
196          int int99;           /* Not used by WinView                    */
198          int int100;          /* Not used by WinView                    */
200          char exprem[5][80];  /* comments                               */
600          int int301;          /* Not used by WinView                    */
602          char label[16];      /* Not used by WinView                    */
618          int gsize;           /* Not used by WinView                    */
620          int lfloat;          /* Not used by WinView                    */
622          char califile[16];   /* calibration file. Not used by WinView  */
638          char bkgdfile[16];   /* background file. Not used by WinView   */
654          int srccmp;          /* Not used by WinView                    */
656          int stripe;          /* number of stripes per frame            */
658          int scramble;        /* 0 - scramble, 1 - unscramble           */
660          long lexpos;         /* exposure val 32-bits(when exposure=-1) */
664          long lnoscan;        /* no. of scan 32-bits(when noscan = -1)  */
668          long lavgexp;        /* no. of accum 32-bits(when avgexp = -1) */
672          char stripfil[16];   /* strip file. Not used by WinView        */
688          char version[16];    /* SW version & date "01.000 02/01/90"    */
704          int controller_type; /* 1-new st120, 2-old st120,              */
                                  /* 3-st130 type 1, 4-st130 type 2,        */
                                  /* 5-st138, 6-DC131, and ST133.           */

             /*  YT_FILE_HEADER  */
             /* The YT variables are not used by WinView.                   */
706           int    yt_file_defined;      /* set TRUE for YT data file     */
708           int    yt_fh_calib_mode;     /* calibration type              */
710           int    yt_fh_calib_type;     /* time-unit (calibration type)  */
712           int    yt_fh_element[12];    /* element number                */
736           double yt_fh_calib_data[12]; /* data                          */
832           float  yt_fh_time_factor;    /* time-factor                   */
836           float  yt_fh_start_time;     /* start time                    */
840       int    reverse_flag;            /* set to 1 if data should be    */
                                          /* reversed, 0 don't reverse     */
              };
```

Version 1.6 Header

(Scheduled for release August, 1996)

All WinView/WinSpec files (version1.6) must begin with the following 4100 byte header. Data files created under previous versions of WinView/WinSpec *can still be read correctly*. However, files created under the new versions (1.6 and higher) ***cannot*** be read by previous versions of WinView/WinSpec.

Header Structure Listing

```
                                 Decimal Byte
                                    Offset
                                 -----------
unsigned int  dioden;           /*    0  num of physical pixels (X axis)   */
int           avgexp;           /*    2  number of accumulations per scan  */
                                /*         if > 32767, set to -1 and       */
                                /*         see lavgexp below (668)         */
int           exposure;         /*    4  exposure time (in milliseconds)   */
                                /*         if > 32767, set to -1 and       */
```

```
                                    /*          see lexpos below (660)          */
    unsigned int  xDimDet;          /*    6  Detector x dimension of chip       */
    int           mode;             /*    8  timing mode                        */
    float         exp_sec;          /*   10  alternative exposure, in secs.     */
    int           asyavg;           /*   14  number of asynchron averages       */
    int           asyseq;           /*   16  number of asynchron sequential     */
    unsigned int  yDimDet;          /*   18  y dimension of CCD or detector.    */
    char          date[10];         /*   20  date as MM/DD/YY                   */
    int           ehour;            /*   30  Experiment Time: Hours (as binary) */
    int           eminute;          /*   32  Experiment Time: Minutes(as binary)*/
    int           noscan;           /*   34  number of multiple scans           */
                                    /*       if noscan == -1 use lnoscan        */
    int           fastacc;          /*   36                                     */
    int           seconds;          /*   38  Experiment Time: Seconds(as binary)*/
    int           DetType;          /*   40  CCD/DiodeArray type                */
    unsigned int  xdim;             /*   42  actual # of pixels on x axis       */
    int           stdiode;          /*   44  trigger diode                      */
    float         nanox;            /*   46                                     */
    float         calibdio[10];     /*   50  calibration diodes                 */
    char          fastfile[16];     /*   90  name of pixel control file         */
    int           asynen;           /*  106  asynchron enable flag  0 = off     */
    int           datatype;         /*  108  experiment data type               */
                                    /*         0 =   FLOATING POINT             */
                                    /*         1 =   LONG INTEGER               */
                                    /*         2 =   INTEGER                    */
                                    /*         3 =   UNSIGNED INTEGER           */
    float         calibnan[10];     /*  110  calibration nanometer              */
    int           BackGrndApplied;  /*  150  set to 1 if background sub done    */
    int           astdiode;         /*  152                                     */
    unsigned int  minblk;           /*  154  min. # of strips per skips         */
    unsigned int  numminblk;        /*  156  # of min-blocks before geo skps    */
    double        calibpol[4];      /*  158  calibration coefficients           */
    unsigned int  ADCrate;          /*  190  ADC rate                           */
    unsigned int  ADCtype;          /*  192  ADC type                           */
    unsigned int  ADCresolution;    /*  194  ADC resolution                     */
    unsigned int  ADCbitAdjust;     /*  196  ADC bit adjust                     */
    unsigned int  gain;             /*  198  gain                               */
    char          exprem[5][80];    /*  200  experiment remarks                 */
    unsigned int  geometric;        /*  600  geometric operations rotate 0x01   */
                                    /*       reverse 0x02, flip 0x04            */
    char          xlabel[16];       /*  602  Intensity display string           */
    unsigned int  cleans;           /*  618  cleans                             */
    unsigned int  NumSkpPerCln;     /*  620  number of skips per clean.         */
    char          califile[16];     /*  622  calibration file name (CSMA)       */
    char          bkgdfile[16];     /*  638  background file name               */
    int           srccmp;           /*  654  number of source comp. diodes      */
    unsigned int  ydim;             /*  656  y dimension of raw data.           */
    int           scramble;         /*  658  0 = scrambled, 1 = unscrambled     */
    long          lexpos;           /*  660  long exposure in milliseconds      */
                                    /*         used if exposure set to -1       */
    long          lnoscan;          /*  664  long num of scans                  */
                                    /*         used if noscan set to -1         */
    long          lavgexp;          /*  668  long num of accumulations          */
                                    /*         used if avgexp set to -1         */
    char          stripfil[16];     /*  672  stripe file (st130)                */
    char          version[16];      /*  688  version & date:"01.000 02/01/90"   */
    int           type;             /*  704   1 = new120 (Type II)              */
                                    /*        2 = old120 (Type I )              */
                                    /*        3 = ST130                         */
                                    /*        4 = ST121                         */
                                    /*        5 = ST138                         */
```

```
                                    /*        6 = DC131 (PentaMAX)              */
                                    /*        7 = ST133 (MicroMAX/SpectroMax),  */
                                    /*        8 = ST135 (GPIB)                  */
                                    /*        9 = VICCD                         */
                                    /*       10 = ST116 (GPIB)                  */
                                    /*       11 = OMA3 (GPIB)                   */
                                    /*       12 = OMA4                          */
int           flatFieldApplied;     /*  706  Set to 1 if flat field was applied */
int           spare[8];             /*  708  reserved                           */
int           kin_trig_mode         /*  724  Kinetics Trigger Mode              */
char          empty[702];           /*  726  EMPTY BLOCK FOR EXPANSION          */
float         clkspd_us;            /* 1428 Vert Clock Speed in micro-sec       */
int           HWaccumFlag;          /* 1432 set to 1 if accum done by Hardware  */
int           StoreSync;            /* 1434 set to 1 if store sync used.        */
int           BlemishApplied;       /* 1436 set to 1 if blemish removal applied */
int           CosmicApplied;        /* 1438 set to 1 if cosmic ray removal done */
int           CosmicType;           /* 1440 if cosmic ray applied, this is type */
float         CosmicThreshold;      /* 1442 Threshold of cosmic ray removal.    */
long          NumFrames;            /* 1446 number of frames in file.           */
float         MaxIntensity;         /* 1450 max intensity of data (future)      */
float         MinIntensity;         /* 1454 min intensity of data (future)      */
char          ylabel[LABELMAX];     /* 1458 y axis label.                       */
unsigned int  ShutterType;          /* 1474 shutter type.                       */
float         shutterComp;          /* 1476 shutter compensation time.          */
unsigned int  readoutMode;          /* 1480 Readout mode, full,kinetics, etc    */
unsigned int  WindowSize;           /* 1482 window size for kinetics only.      */
unsigned int  clkspd;               /* 1484 clock speed for kinetics &          */
                                    /*      frame transfer.                     */
unsigned int  interface_type;       /* 1486 computer interface (isa-taxi,       */
                                    /*      pci, eisa, etc.)                    */
unsigned long ioAdd1;               /* 1488 I/O address of interface card.      */
unsigned long ioAdd2;               /* 1492 if more than one address for card.  */
unsigned long ioAdd3;               /* 1496                                     */
unsigned int  intLevel;             /* 1500 interrupt level interface card      */
unsigned int  GPIBadd;              /* 1502 GPIB address (if used)              */
unsigned int  ControlAdd;           /* 1504 GPIB controller address (if used)   */
unsigned int  controllerNum;        /* 1506 if multiple controller system will  */
                                    /*       have controller # data came from.  */
                                    /*       (Future Item)                      */
unsigned int  SWmade;               /* 1508 Software which created this file    */
int           NumROI;               /* 1510 number of ROIs used. if 0 assume 1  */
                                    /* 1512 - 1630  ROI information             */
struct ROIinfo {                    /*                                          */
 unsigned int startx;               /* left x start value.                      */
 unsigned int endx;                 /* right x value.                           */
 unsigned int groupx;               /* amount x is binned/grouped in hw.        */
 unsigned int starty;               /* top y start value.                       */
 unsigned int endy;                 /* bottom y value.                          */
 unsigned int groupy;               /* amount y is binned/grouped in hw.        */
} ROIinfoblk[10];                   /*    ROI Starting Offsets:                 */
                                    /*            ROI  1 = 1512                 */
                                    /*            ROI  2 = 1524                 */
                                    /*            ROI  3 = 1536                 */
                                    /*            ROI  4 = 1548                 */
                                    /*            ROI  5 = 1560                 */
                                    /*            ROI  6 = 1572                 */
                                    /*            ROI  7 = 1584                 */
                                    /*            ROI  8 = 1596                 */
                                    /*            ROI  9 = 1608                 */
                                    /*            ROI 10 = 1620                 */
char          FlatField[120];       /* 1632 Flat field file name.               */
```

```
char          background[120];   /* 1752 Background sub. file name.          */
char          blemish[120];      /* 1872 Blemish file name.                  */
float         software_ver;      /* 1992 Software version.                   */
char          UserInfo[1000];    /* 1996-2995 user data.                     */
long          WinView_id;        /* 2996 Set to 0x01234567L if file was      */
                                 /* created by WinX                          */

--------------------------------------------------------------------------------
```

Calibration Structures

There are three structures for the calibrations

- The Area Inside the Calibration Structure (below) is repeated two times.

```
xcalibration,               /* 3000 - 3488 x axis calibration           */
ycalibration,               /* 3489 - 3977 y axis calibration           */
```

Start of X Calibration Structure

```
double        offset;           /* 3000  offset for absolute data scaling   */
double        factor;           /* 3008  factor for absolute data scaling   */
char          current_unit;     /* 3016  selected scaling unit              */
char          reserved1;        /* 3017  reserved                           */
char          string[40];       /* 3018  special string for scaling         */
char          reserved2[40];    /* 3058  reserved                           */
char          calib_valid;      /* 3098  flag if calibration is valid       */
char          input_unit;       /* 3099  current input units for            */
                                /*       "calib_value"                      */
char          polynom_unit;     /* 3100  linear UNIT and used               */
                                /*       in the "polynom_coeff"             */
char          polynom_order;    /* 3101  ORDER of calibration POLYNOM       */
char          calib_count;      /* 3102  valid calibration data pairs       */
double        pixel_position[10];/* 3103  pixel pos. of calibration data    */
double        calib_value[10];  /* 3183  calibration VALUE at above pos     */
double        polynom_coeff[6]; /* 3263  polynom COEFFICIENTS               */
double        laser_position;   /* 3311  laser wavenumber for relativ WN    */
char          reserved3;        /* 3319  reserved                           */
unsigned char new_calib_flag;   /* 3320  If set to 200, valid label below   */
char          calib_label[81];  /* 3321  Calibration label (NULL term'd)    */
char          expansion[87];    /* 3402  Calibration Expansion area         */

--------------------------------------------------------------------------------
```

Start of Y Calibration Structure

```
double        offset;           /* 3489  offset for absolute data scaling   */
double        factor;           /* 3497  factor for absolute data scaling   */
char          current_unit;     /* 3505  selected scaling unit              */
char          reserved1;        /* 3506  reserved                           */
char          string[40];       /* 3507  special string for scaling         */
char          reserved2[40];    /* 3547  reserved                           */
char          calib_valid;      /* 3587  flag if calibration is valid       */
char          input_unit;       /* 3588  current input units for            */
                                /*       "calib_value"                      */
char          polynom_unit;     /* 3589  linear UNIT and used               */
                                /*       in the "polynom_coeff"             */
char          polynom_order;    /* 3590  ORDER of calibration POLYNOM       */
char          calib_count;      /* 3591  valid calibration data pairs       */
double        pixel_position[10];/* 3592  pixel pos. of calibration data    */
double        calib_value[10];  /* 3672  calibration VALUE at above pos     */
```

```
double        polynom_coeff[6];  /* 3752  polynom COEFFICIENTS               */
double        laser_position;    /* 3800  laser wavenumber for relativ WN    */
char          reserved3;         /* 3808  reserved                           */
unsigned char new_calib_flag;    /* 3809  If set to 200, valid label below   */
char          calib_label[81];   /* 3810  Calibration label (NULL term'd)    */
char          expansion[87];     /* 3891  Calibration Expansion area         */
```

End of Calibration Structures

```
char          Istring[40];       /* 3978  special Intensity scaling string   */
char          empty3[80];        /* 4018  empty block to reach 4100 bytes    */
int           lastvalue;         /* 4098 Always the LAST value in the header */
```

WINX Header Structure (with actual offsets)

(12-June-96)

dioden	0	0
avgexp	2	2
exposure	4	4
xDimDet	6	6
mode	8	8
exp_sec	10	A
asyavg	14	E
asyseq	16	10
yDimDet	18	12
date	20	14
ehour	30	1E
eminute	32	20
noscan	34	22
fastacc	36	24
seconds	38	26
DetType	40	28
xdim	42	2A
stdiode	44	2C
nanox	46	2E
calibdio (10 x float).....	50	32
fastfile	90	5A
asynen	106	6A
datatype	108	6C
calibnan (10 x float).....	110	6E
BackGrndApplied	150	96
astdiode	152	98
minblk	154	9A
numminblk	156	9C
calibpol (4 x double).....	158	9E
ADCrate	190	BE
ADCtype	192	C0
ADCresolution	194	C2
ADCbitAdjust	196	C4
gain	198	C6
exprem[0] (comment 1).....	200	C8
exprem[1] (comment 2).....	280	118
exprem[2] (comment 3).....	360	168
exprem[3] (comment 4).....	440	1B8
exprem[4] (comment 5).....	520	208
geometric	600	258
xlabel	602	25A
cleans	618	26A
NumSkpPerCln	620	26C
califile	622	26E
bkgdfile	638	27E
srccmp	654	28E

ydim	656	290
scramble	658	292
lexpos	660	294
lnoscan	664	298
lavgexp	668	29C
stripfil	672	2A0
version	688	2B0
type	704	2C0
flatFieldApplied	706	2C2
spare	708	2C4
kin_trig_mode	724	2D4
empty	726	2D6
clkspd_us	1428	594
HWaccumFlag	1432	598
StoreSync	1434	59A
BlemishApplied	1436	59C
CosmicApplied	1438	59E
CosmicType	1440	5A0
CosmicThreshold	1442	5A2
NumFrames	1446	5A6
MaxIntensity	1450	5AA
MinIntensity	1454	5AE
ylabel	1458	5B2
ShutterType	1474	5C2
shutterComp	1476	5C4
readoutMode	1480	5C8
WindowSize	1482	5CA
clkspd	1484	5CC
interface_type	1486	5CE
ioAdd1	1488	5D0
ioAdd2	1492	5D4
ioAdd3	1496	5D8
intLevel	1500	5DC
GPIBadd	1502	5DE
ControlAdd	1504	5E0
controllerNum	1506	5E2
SWmade	1508	5E4
NumROI	1510	5E6
ROIinfoblk[1] - startx	1512	5E8
ROIinfoblk[1] - endx	1514	5EA
ROIinfoblk[1] - groupx	1516	5EC
ROIinfoblk[1] - starty	1518	5EE
ROIinfoblk[1] - endy	1520	5F0
ROIinfoblk[1] - groupy	1522	5F2
ROIinfoblk[2] - startx	1524	5F4
ROIinfoblk[2] - endx	1526	5F6
ROIinfoblk[2] - groupx	1528	5F8
ROIinfoblk[2] - starty	1530	5FA
ROIinfoblk[2] - endy	1532	5FC
ROIinfoblk[2] - groupy	1534	5FE
ROIinfoblk[3] - startx	1536	600
ROIinfoblk[3] - endx	1538	602
ROIinfoblk[3] - groupx	1540	604
ROIinfoblk[3] - starty	1542	606
ROIinfoblk[3] - endy	1544	608
ROIinfoblk[3] - groupy	1546	60A
ROIinfoblk[4] - startx	1548	60C
ROIinfoblk[4] - endx	1550	60E
ROIinfoblk[4] - groupx	1552	610
ROIinfoblk[4] - starty	1554	612
ROIinfoblk[4] - endy	1556	614
ROIinfoblk[4] - groupy	1558	616
ROIinfoblk[5] - startx	1560	618
ROIinfoblk[5] - endx	1562	61A
ROIinfoblk[5] - groupx	1564	61C
ROIinfoblk[5] - starty	1566	61E

```
ROIinfoblk[5]  - endy ......... 1568   620
ROIinfoblk[5]  - groupy ....... 1570   622
ROIinfoblk[6]  - startx ....... 1572   624
ROIinfoblk[6]  - endx ......... 1574   626
ROIinfoblk[6]  - groupx ....... 1576   628
ROIinfoblk[6]  - starty ....... 1578   62A
ROIinfoblk[6]  - endy ......... 1580   62C
ROIinfoblk[6]  - groupy ....... 1582   62E
ROIinfoblk[7]  - startx ....... 1584   630
ROIinfoblk[7]  - endx ......... 1586   632
ROIinfoblk[7]  - groupx ....... 1588   634
ROIinfoblk[7]  - starty ....... 1590   636
ROIinfoblk[7]  - endy ......... 1592   638
ROIinfoblk[7]  - groupy ....... 1594   63A
ROIinfoblk[8]  - startx ....... 1596   63C
ROIinfoblk[8]  - endx ......... 1598   63E
ROIinfoblk[8]  - groupx ....... 1600   640
ROIinfoblk[8]  - starty ....... 1602   642
ROIinfoblk[8]  - endy ......... 1604   644
ROIinfoblk[8]  - groupy ....... 1606   646
ROIinfoblk[9]  - startx ....... 1608   648
ROIinfoblk[9]  - endx ......... 1610   64A
ROIinfoblk[9]  - groupx ....... 1612   64C
ROIinfoblk[9]  - starty ....... 1614   64E
ROIinfoblk[9]  - endy ......... 1616   650
ROIinfoblk[9]  - groupy ....... 1618   652
ROIinfoblk[10] - startx ....... 1620   654
ROIinfoblk[10] - endx ......... 1622   656
ROIinfoblk[10] - groupx ....... 1624   658
ROIinfoblk[10] - starty ....... 1626   65A
ROIinfoblk[10] - endy ......... 1628   65C
ROIinfoblk[10] - groupy ....... 1630   65E
FlatField ..................... 1632   660
background .................... 1752   6D8
blemish ....................... 1872   750
software_ver .................. 1992   7C8
UserInfo ...................... 1996   7CC
WinView_id .................... 2996   BB4
X Calib: offset ............... 3000   BB8
X Calib: factor ............... 3008   BC0
X Calib: current_unit ......... 3016   BC8
X Calib: reserved1 ............ 3017   BC9
X Calib: string ............... 3018   BCA
X Calib: reserved2 ............ 3058   BF2
X Calib: calib_valid .......... 3098   C1A
X Calib: input_unit ........... 3099   C1B
X Calib: polynom_unit ......... 3100   C1C
X Calib: polynom_order ........ 3101   C1D
X Calib: calib_count .......... 3102   C1E
X Calib: pixel_position[1] .... 3103   C1F
X Calib: pixel_position[2] .... 3111   C27
X Calib: pixel_position[3] .... 3119   C2F
X Calib: pixel_position[4] .... 3127   C37
X Calib: pixel_position[5] .... 3135   C3F
X Calib: pixel_position[6] .... 3143   C47
X Calib: pixel_position[7] .... 3151   C4F
X Calib: pixel_position[8] .... 3159   C57
X Calib: pixel_position[9] .... 3167   C5F
X Calib: pixel_position[10] ... 3175   C67
X Calib: calib_value[1] ....... 3183   C6F
X Calib: calib_value[2] ....... 3191   C77
X Calib: calib_value[3] ....... 3199   C7F
X Calib: calib_value[4] ....... 3207   C87
X Calib: calib_value[5] ....... 3215   C8F
X Calib: calib_value[6] ....... 3223   C97
X Calib: calib_value[7] ....... 3231   C9F
```

```
X Calib: calib_value[8] ....... 3239   CA7
X Calib: calib_value[9] ....... 3247   CAF
X Calib: calib_value[10] ...... 3255   CB7
X Calib: polynom_coeff[1] ..... 3263   CBF
X Calib: polynom_coeff[2] ..... 3271   CC7
X Calib: polynom_coeff[3] ..... 3279   CCF
X Calib: polynom_coeff[4] ..... 3287   CD7
X Calib: polynom_coeff[5] ..... 3295   CDF
X Calib: polynom_coeff[6] ..... 3303   CE7
X Calib: laser_position ....... 3311   CEF
X Calib: reserved3 ............ 3319   CF7
X Calib: new_calib_flag ....... 3320   CF8
X Calib: calib_label .......... 3321   CF9
X Calib: expansion ............ 3402   D4A
Y Calib: offset ............... 3489   DA1
Y Calib: factor ............... 3497   DA9
Y Calib: current_unit ......... 3505   DB1
Y Calib: reserved1 ............ 3506   DB2
Y Calib: string ............... 3507   DB3
Y Calib: reserved2 ............ 3547   DDB
Y Calib: calib_valid .......... 3587   E03
Y Calib: input_unit ........... 3588   E04
Y Calib: polynom_unit ......... 3589   E05
Y Calib: polynom_order ........ 3590   E06
Y Calib: calib_count .......... 3591   E07
Y Calib: pixel_position[1] .... 3592   E08
Y Calib: pixel_position[2] .... 3600   E10
Y Calib: pixel_position[3] .... 3608   E18
Y Calib: pixel_position[4] .... 3616   E20
Y Calib: pixel_position[5] .... 3624   E28
Y Calib: pixel_position[6] .... 3632   E30
Y Calib: pixel_position[7] .... 3640   E38
Y Calib: pixel_position[8] .... 3648   E40
Y Calib: pixel_position[9] .... 3656   E48
Y Calib: pixel_position[10] ... 3664   E50
Y Calib: calib_value[1] ....... 3672   E58
Y Calib: calib_value[2] ....... 3680   E60
Y Calib: calib_value[3] ....... 3688   E68
Y Calib: calib_value[4] ....... 3696   E70
Y Calib: calib_value[5] ....... 3704   E78
Y Calib: calib_value[6] ....... 3712   E80
Y Calib: calib_value[7] ....... 3720   E88
Y Calib: calib_value[8] ....... 3728   E90
Y Calib: calib_value[9] ....... 3736   E98
Y Calib: calib_value[10] ...... 3744   EA0
Y Calib: polynom_coeff[1] ..... 3752   EA8
Y Calib: polynom_coeff[2] ..... 3760   EB0
Y Calib: polynom_coeff[3] ..... 3768   EB8
Y Calib: polynom_coeff[4] ..... 3776   EC0
Y Calib: polynom_coeff[5] ..... 3784   EC8
Y Calib: polynom_coeff[6] ..... 3792   ED0
Y Calib: laser_position ....... 3800   ED8
Y Calib: reserved3 ............ 3808   EE0
Y Calib: new_calib_flag ....... 3809   EE1
Y Calib: calib_label .......... 3810   EE2
Y Calib: expansion ............ 3891   F33
Istring ....................... 3978   F8A
empty3 ........................ 4018   FB2
lastvalue ..................... 4098  1002
```

Start of Data

The data follows the header beginning at offset 4100.

In WinView/WinSpec, the data is always stored exactly as it is collected. The order of the data depends on the placement of the shift register.

In the diagram below, the shift register is on the RIGHT SIDE of the chip. Each COLUMN of data is first shifted RIGHT into the shift register and then DOWN. The data is read (and stored) in this order:

```
First column read:    XnYm, XnYm-1, ... XnY2, XnY1

Last column read :    X1Ym, X1Ym-1, ... X1Y2, X1Y1

 .---------------------------------------.---.
 |   X1Y1       X2Y1       ...       XnY1   | S |
 |   X1Y2       X2Y2       ...       XnY2   | H |  |
 |     :         :                     :     | I |  |
 |   X1Ym-1     X2Ym-1     ...       XnYm-1 | F |  V
 |   X1Ym       X2Ym       ...       XnYm   | T |
 `---------------------------------------|---|
                  ------>                | | |
                                         |
```

In the diagram below, the shift register is on the BOTTOM of the chip. Each ROW of data is first shifted DOWN into the shift register and then RIGHT. The data is read (and stored) in this order:

```
First row read:    XnYm, ... X2Ym, X1Ym

Last row read :    XnY1, ... X2Y1, X1Y1

 .---------------------------------------.
 |   X1Y1       X2Y1       ...       XnY1   |
 |   X1Y2       X2Y2       ...       XnY2   |  |
 |     :         :                     :     |  |
 |   X1Ym-1     X2Ym-1     ...       XnYm-1 |  V
 |   X1Ym       X2Ym       ...       XnYm   |
 `---------------------------------------|---.
 |             SHIFT REGISTER             | | |
                 ----->                    |
                                           `-----> A/D
```

Reading Strips and Frames of Data

Data Files (up to and including WinX version 1.4.3)
Data is stored as sequential points. The X, Y and Frame dimensions are determined by the header. The X dimension is in "faccount" (Offset 42). The total number of strips in the file is in "noscan" (Offset 34) OR if the total number is > 32767, "noscan" is set to -1 and the number of strips is stored as a LONG in "lnoscan" (Offset 664). The number of strips per frame is stored in "stripe" (Offset 656). To determine the number of frames, divide "noscan" (or "lnoscan") by "stripe".

Thus:

```
char header[4100];
int  X_dimension;
int  temp_y_dim;
long Y_dimension;
```

```
int  Total_strips;
int  Num_frames;

X_dimension = (int)header[42];

temp_y_dim  = (int)header[34];
if( temp_y_dim == -1 )
    Y_dimension = (long)header[664];
else
    Y_dimension = (long)temp_y_dim;

Total_strips = (int)header[656];
Num_frames   = (long)Total_strips / Y_dimension;
```

Data Files from WinX version 1.6 and Beyond
Data is still stored as sequential points. The X, Y and Frame dimensions are determined by the header.

- The X dimension of the stored data is in "xdim" (Offset 42).
- The Y dimension of the stored data is in "ydim" (Offset 656).
- The number of frames of data stored is in "NumFrames" (Offset 1446).

Thus (modifying the example above):

```
char header[4100];
unsigned int X_dimension;
unsigned int Y_dimension;
long         Num_frames;

     /* Now there is Direct Access of data dimensions */
X_dimension = (unsigned int)header[42];
Y_dimension = (unsigned int)header[664];
Num_frames  = (long)header[1446];
```

Example File Write

```
void Test_WriteTestFile
(
 char     *runfile,               /* data file to write to.   */
 void huge *big_buffer,           /* buffer holding data.     */
 int byte_size_of_pixel           /* size of a pixel in bytes */
 )
{


    /*  Generate a TEST WinView DATA file       */
    FILE * fileptr;    /*  Output File Pointer  */
    struct WINXHEAD * header;
    char huge *address;
    int jndx;

    header = (struct WINXHEAD *) malloc((size_t)sizeof(struct WINXHEAD));

    /*  fill in some of the header              */
    header->noscan = -1;
```

```
    header->datatype = 3;  //unsigned integer

    header->stripe   = PICM_Get_pixeldimension_y();
    /* if multiple frame, * frameNum */
    header->lnoscan  = header->stripe;
    header->faccount = PICM_Get_pixeldimension_x();
    header->dioden   = header->faccount;
    header->scramble = 1;

    fileptr = fopen( runfile, "w+b" );

    /*  write WinView HEADER to Output File     */
    fwrite( header,
            (size_t)4100,
            (size_t)1,
            fileptr );

    /*  release Header Memory                   */
    free( header );

    /*  write data to Output File               */
    address = (char huge*)big_buffer;

    for( jndx = 0; jndx < PICM_Get_pixeldimension_y(); jndx++ )
    {
        fwrite( address,
                byte_size_of_pixel,
                PICM_Get_pixeldimension_x(),
                fileptr );

        address = address +
                  PICM_Get_pixeldimension_x() *
                  byte_size_of_pixel;
    }

    /*  close Output File  */
    fclose( fileptr );


}   /*  end of Test_WriteTestFile()  */
```

This page intentionally left blank.

Appendix B

CCD Chips and Diode Arrays

CCDs

```
                        /*              DLL/WinView        Y     X      */
CUSTOM_CCD = -1,        /*  Custom User defined CCD chip               */
      /*** CUSTOM_CCD AS A CHIP TYPE IS NO LONGER USED !!!! ***/
NO_CCD_SENSOR,          /*        PN#  CCD                 X     Y      */
EEV_256x1024_3ph,       /* EEV 256x1024 3-phase           256  1024    */
EEV_576x384_3ph,        /* EEV 576x384 3-phase            576   384    */
EEV_1152x298_3ph,       /* EEV 1152x298 3-phase          1152   298    */
EEV_1152x1242_3ph,      /* EEV 1152x1242                 1152  1242    */
KDK_512x768,            /* KODAK 512x768                  512   768    */
KDK_1035x1317,          /* KODAK 1035x1317               1035  1317    */
KDK_1024x1280,          /* KODAK 1024x1280               1024  1280    */
KDK_2044x2033,          /* KODAK 2044x2033               2044  2033    */
KDK_2048x3072,          /* KODAK 2048x3072               2048  3072    */
                        /* Fast kodak special, will be removed in future */
KODAKFAST1400,          /* Use custom chip               1035  1317    */
                        /* Overscan special, will be removed in future */
OSTK1024B,              /* Use custom chip               1050  1050    */
PID_330x1100_8phH,      /* PI 330x1100 8 phase (horz)     330  1100    */
PID_532x1752,           /* PI 532x1752                    532  1752    */
RET_400x1200,           /* RET 400x1200                   400  1200    */
RET_512x512,            /* RET 512x512                    512   512    */
NOT_USED2,              /* old EEV 576x384  (not used)    576   384    */
RET_1024x1024,          /* RET 1K x 1K                   1024  1024    */
RET_2048x2048,          /* RET 2K x 2K                   2048  2048    */
TEK_512x512_B_100ns,    /* TEK512x512B Back  [100ns]      512   512    */
TEK_512x512_F_100ns,    /* TEK512x512F Front [100ns]      512   512    */
TEK_1024x1024_B_100ns, /* TEK1024x1024B Back [100ns]     1024  1024    */
TEK_1024x1024_F_100ns, /* TEK1024x1024F Front[100ns]     1024  1024    */
TEK_2048x2048,          /* TEK 2K x 2K                   2048  2048    */
THM_576x384,            /* TH576x384                      576   384    */
EEV_256x1024_6ph,       /* EEV 256x1024  6 PHASE          256  1024    */
EEV_1024x512_FT,        /* EEV frame transfer            1024   512    */
NOT_USED3,              /* Use custom chip                512  1024    */
NOT_USED4,              /* Use custom chip                512  1024    */
EEV_576x384_6ph,        /* EEV576x384   6 PHASE           576   384    */
EEV_1152x298_6ph,       /* EEV1152x298  6 PHASE          1152   298    */
EEV_1152x1242_6ph,      /* EEV1152x1242 6 PHASE          1152  1242    */
NOT_USED5,              /* Use custom chip               1024  2048    */
NOT_USED6,              /* Use custom chip               1024  2048    */
TEK_1024x1024_B_200ns, /* TEK1024x1024B Back Illm        1024  1024    */
TEK_1024x1024_F_200ns, /* TEK1024x1024F Front Illm       1024  1024    */
KDK_1024x1536,          /* KODAK 1024x1536               1024  1536    */
TEK_512x512_B_200ns,    /* TEK512x512B [200ns]            512   512    */
TEK_512x512_F_200ns,    /* TEK512x512F [200ns]            512   512    */
```

```
    NOT_USED1,              /* NOT USED                          ?     ?    */
    TEK_512x512D_B,         /* TEK512x512D Back Illm           512   512    */
    TEK_512x512D_F,         /* TEK512x512D Front Illm          512   512    */
    HAM_64x1024,            /* HAMMAMATSU   64 x 1024           64  1024    */
    HAM_128x1024,           /* HAMMAMATSU  128 x 1024          128  1024    */
    HAM_256x1024,           /* HAMMAMATSU  256 x 1024          256  1024    */
    EEV_256x1024_8ph,       /* EEV 256 x 1024 8 PHASE          256  1024    */
    EEV_1152x770_3ph,       /* EEV1152x770  3 PHASE           1152   770    */
    EEV_1152x770_6ph,       /* EEV 1152x770 6 PHASE           1152   770    */
    TEK_1024x1024_B_42usV, /* TEK1024x1024B Back Illm         1024  1024    */
    PID_330x1100_6phH,      /* PI 330x1100 6 PHASE (horz)      330  1100    */
    EEV_256x1024_6ph_CCD30,/* EEV 256x1024  6 PHASE CCD30      256  1024    */
    TEK_1024x1024D_B,       /* TEK1024x1024D Back Illm         1024  1024    */
    TEK_1024x1024D_F,       /* TEK1024x1024D Front Illm        1024  1024    */
    TEK_1024x1024D_B_T3,    /* TEK1024x1024D Back Illm         1024  1024    */
    THM_512x512,            /* Thomson 512X512 Front Illum      512   512    */
    THM_256x1024,           /* Thomson 256X1024 FI MPP          256  1024    */
    THM_2048x1024_FT,       /* Thomson 2048X1024 FT            1024  1024    */
    SIT_800x2000_B,         /* SIT 800x2000 Back Illm           800  2000    */
    SIT_800x2000_F,         /* SIT 800x2000 Front Illm          800  2000    */
    PID_240x330_MCT,        /* TEST CHIP # 1                                 */
    OEEV_1203x1336_3ph,     /* EEV 1203x1336                   1203  1336    */
    OEEV_1203x1336_6ph,     /* EEV 1203x1336                   1203  1336    */
    PI_800x1000_B,          /* PI 800x1000 back                 800  1000    */
    PI_64x1024,             /* PI special   64 x 1024           64  1024    */
    PI_128x1024,            /* PI special  128 x 1024          128  1024    */
    PI_256x1024,            /* PI special  256 x 1024          256  1024    */
    KDK_4096x4096,          /* Kodak 4096x4096                 4096  4096    */
    EEV_100x1340_6ph_CCD36,/* EEV 100x1340  6 PHASE CCD36       100  1340    */
    PID_1030x1300,          /* PI Special 1030x1300            1030  1300    */
    VICCD_NTSC_480x640,     /* Video Chip - N. American Std     480   640    */
    VICCD_CCIR_576x768,     /* Video Chip - European Std        576   768    */
    PID_582x782,            /* PI Special 582x782               582   782    */
    PID_2500x600_B,         /* PI 2500x600 Back                2500   600    */
    PID_2500x600_F,         /* PI 2500x600 Front               2500   600    */
    TEST_CHIP_2,            /* TEST CHIP # 2                                 */
    EEV_400x1340,           /* EEV 400x1340                     400  1340    */
    EEV_700x1340,           /* TEST CHIP # 3                                 */
    EEV_1024x1024,          /* TEST CHIP # 4                                 */
    EEV_1024x1024_FT,       /* EEV frame transfer              1024  1024    */
    EEV_1300x1340,          /* EEV 1300x1340                   1300  1340    */
    SIT_2048x2048_B,        /* SITE 2048x2048 Back             2048  2048    */
    SIT_2048x2048_F,        /* SITE 2028x2048 Front            2048  2048    */
    TEST_CHIP_6,            /* TEST CHIP # 6                                 */
    TEST_CCD36_00,          /* Special ccd36 for EEV.           110  1356    */
    TEST_CCD36_10,          /* Special ccd36 for EEV.           410  1356    */
    TEST_CCD36_20,          /* Special ccd36 for EEV.           710  1356    */
    TEST_CCD36_40,          /* Special ccd36 for EEV.          1330  1356    */
    THM_2048x2048,          /* Thomson 2048x2048               2048  2048    */
    OEEV_1300x1340,         /* EEV 1300x1340 [OverScan]        1300  1340    */
       EPIX_1300x1024,         /* Epix Controller */
    END_OF_CCD_LIST         /* END OF ENUMERATED CCD TYPE,                   */
```

Diode Arrays

```
/* sensor type definition */
/* #### used by PICM_CREATECONTROLLER param 2 (if diode array) #### */
enum ctrl_PDA_sensor

    NO_PDA_SENSOR =1000,
```

```
DA0128S,                /* single  128 */
DA0256S,                /* single  256 */
DA0512S,                /* single  512 */
DA1024S,                /* single 1024 */
DA2048S,                /* single 2048 */
DA0128D,                /* dual    128 */
DA0256D,                /* dual    256 */
DA0512D,                /* dual    512 */
DA1024D,                /* dual   1024 */
DA2048D,                /* dual   2048 ! */
DA256S_INGAS,           /* Single 256 INGAS */
DA512S_INGAS,           /* Single 512 INGAS */
DA128S_GE,              /* Single 128 GE    */
DA256S_GE,              /* Single 256 GE    */
PDA_DUMMY               /* indicates end of list  */
```

This page intentionally left blank.

Appendix C

Timing Modes vs. Controller Model

Introduction

The following table lists the controller timing modes, the controller models, and the hardware version at which the listed timing modes became available. These timing modes are invoked using the Acquisition Timing Mode definitions as listed in the supplied file pitimdef.h. Note that these timing modes behave a little differently for the various controller models. The following paragraphs separately discuss the timing modes for each controller. Those discussions are followed by a brief explanation of the Acquisition Timing Mode definitions.

	Default	External Sync		External Sync Continuous Cleans		Store	Ext.	Line
	FreeRun	Normal	PreOpen	Normal	PreOpen	Enable	Trig	Sync
PentaMAX v0 -v4	yes	yes	yes	no	no	no	na	na
PentaMAX v5	yes	yes	yes	yes	yes	no	na	na
MicroMAX v0-v1	yes	yes	yes	yes	yes	no	na	na
MicroMAX v2	yes	yes	yes	yes	yes	no	na	na
ST138	yes	yes	yes	yes	yes	yes	na	na
ST130	yes	yes	yes	no	no	yes	na	na
ST121	yes	yes (no shutter, just Ext Sync)		no	no	yes	yes	yes
ST120	yes	yes (no shutter, just Ext. Sync)		no	no	no	yes	yes

PentaMAX

Introduction

The Princeton Instruments PentaMAX system has been designed to allow the greatest possible flexibility when synchronizing data collection with an experiment.

The following table lists the timing mode combinations. Use this chart in combination with the detailed descriptions to determine the optimal timing configuration.

Mode	Shutter
Freerun	Normal
External Sync	Normal
External Sync	Preopen

Standard Timing Modes

The two basic PentaMAX triggering modes are Freerun and External Sync. These timing modes are combined with the Shutter options to provide the widest variety of timing modes for precision experiment synchronization.

Shutter Modes

The shutter options available include Normal, Preopen, Disable Opened or Disable Closed. Disable simply means that the shutter will not operate during the experiment. Disable closed is useful for making dark charge measurements, or when no shutter is present in the system. Preopen, available in the External Sync mode, opens the shutter as soon as the PentaMAX is ready to receive an External Sync pulse. This is required if the time between the External Sync pulse and the event is less than a few milliseconds, the time it takes the shutter to open.

Freerun timing

In the Freerun mode the controller does not synchronize with the experiment in any way. The shutter opens as soon as the previous readout is complete, and remains open for the exposure time, t_{exp}. Any External Sync signals are ignored. This mode is useful for experiments with a constant light source, such as a CW laser or a DC lamp. Other experiments that can utilize this mode are high repetition studies, where the number of shots that occur during a single shutter cycle is so large that it appears to be continuous illumination.

Other experimental equipment can be synchronized to the PentaMAX system by using the NOTSCAN signal.

External Sync timing

In this mode all exposures are synchronized to an external source. This mode can be used in combination with Normal or Preopen Shutter operation. In the Normal Shutter mode,

the controller waits for an External Sync pulse, then opens the shutter for the programmed exposure period. As soon as the exposure is complete the shutter closes and the CCD array is read out. The shutter requires 5-10 msec to open completely, depending on the model of shutter.

Since the shutter requires up to 10 msec to fully open, the External Sync pulse provided by the experiment must precede the actual signal by at least that much time. If not, the shutter will not be open during the entire signal, or the signal may be missed completely.

Also, since the amount of time from the initialization of the experiment to the first External Sync pulse is not fixed, an accurate background subtraction may not be possible for the first readout. In multiple-shot experiments this is easily overcome by simply discarding the first frame.

In the Preopen Shutter mode, on the other hand, shutter operation is only partially synchronized to the experiment. As soon as the controller is ready to collect data the shutter opens. Upon arrival of the first External Sync pulse at the PentaMAX, the shutter remains open for the specified exposure period, closes, and the CCD is read out. As soon as readout is complete the shutter reopens and waits for the next frame.

The Preopen mode is useful in cases where an External Sync pulse cannot be provided 5-10 msec before the actual signal occurs. Its main drawback is that the CCD is exposed to any ambient light while the shutter is open between frames. If this ambient light is constant, and the triggers occur at regular intervals, this background can also be subtracted, providing that it does not saturate the CCD. As with Normal Shutter mode, accurate background subtraction may not be possible for the first frame.

Also note that, in addition to signal from ambient light, dark charge accumulates during the “wait” time (t_w). Any variation in the external sync frequency also affects the amount of dark charge, even if light is not falling on the CCD during this time.

MicroMAX

Standard Timing Modes

The basic MicroMAX timing modes are Free Run, External Sync, and External Sync with Continuous Cleans. These timing modes are combined with the Shutter options to provide the widest variety of timing modes for precision experiment synchronization.

Shutter Modes

The shutter options available include Normal, PreOpen, Disable Opened or Disable Closed. Disable simply means that the shutter will not operate during the experiment. Disable closed is useful for making dark charge measurements, or when no shutter is present in the system. PreOpen, available in the External Sync mode, opens the shutter as soon as the MicroMAX is ready to receive an External Sync pulse. This is required if the time between the External Sync pulse and the event is less than a few milliseconds, the time it takes the shutter to open.

Freerun timing

In the Free Run mode the controller does not synchronize with the experiment in any way. The shutter opens as soon as the previous readout is complete, and remains open for the exposure time, t_{exp}. Any External Sync signals are ignored. This mode is useful for experiments with a constant light source, such as a CW laser or a DC lamp. Other experiments that can utilize this mode are high repetition studies, where the number of shots that occur during a single shutter cycle is so large that it appears to be continuous illumination.

Other experimental equipment can be synchronized to the MicroMAX system by using the $\overline{\text{SCAN}}$ (NOTSCAN) signal.

External Sync timing

In this mode all exposures are synchronized to an external source. This mode can be used in combination with Normal or PreOpen Shutter operation. In Normal Shutter mode, the controller waits for an External Sync pulse, then opens the shutter for the programmed exposure period. As soon as the exposure is complete, the shutter closes and the CCD array is read out. The shutter requires 5-10 msec to open completely, depending on the model of shutter.

Since the shutter requires up to 10 msec to fully open, the External Sync pulse provided by the experiment must precede the actual signal by at least that much time. If not, the shutter will not be open for the duration of the entire signal, or the signal may be missed completely.

Also, since the amount of time from initialization of the experiment to the first External Sync pulse is not fixed, an accurate background subtraction may not be possible for the first readout. In multiple-shot experiments this is easily overcome by simply discarding the first frame.

In the PreOpen Shutter mode, on the other hand, shutter operation is only partially synchronized to the experiment. As soon as the controller is ready to collect data the shutter opens. Upon arrival of the first External Sync pulse at the MicroMAX, the shutter remains open for the specified exposure period, closes, and the CCD is read out. As soon as readout is complete the shutter reopens and waits for the next frame.

The PreOpen mode is useful in cases where an External Sync pulse cannot be provided 5-10 msec before the actual signal occurs. Its main drawback is that the CCD is exposed to any ambient light while the shutter is open between frames. If this ambient light is constant, and the triggers occur at regular intervals, this background can also be subtracted, providing that it does not saturate the CCD. As with the Normal Shutter mode, accurate background subtraction may not be possible for the first frame.

Also note that, in addition to signal from ambient light, dark charge accumulates during the “wait” time (t_w). Any variation in the external sync frequency also affects the amount of dark charge, even if light is not falling on the CCD during this time.

External Sync with Continuous Cleans Timing

The third timing mode available with the MicroMAX camera is called Continuous Cleans. In addition to the standard “cleaning” of the array, which occurs after the

controller is enabled, Continuous Cleans will remove any charge from the array until the moment the External Sync pulse is received.

Once the External Sync pulse is received, cleaning of the array stops as soon as the current row is shifted, and frame collection begins. With Normal Shutter operation the shutter is opened for the set exposure time. With PreOpen Shutter operation the shutter is open during the continuous cleaning, and once the External Sync pulse is received the shutter remains open for the set exposure time, then closes. If the vertical rows are shifted midway when the External Sync pulse arrives, the pulse is saved until the row shifting is completed, to prevent the CCD from getting "out of step." As expected, the response latency is on the order of one vertical shift time, from 1-30 µsec depending on the array. This latency does not prevent the incoming signal from being detected, since photo generated electrons are still collected over the entire active area. However, if the signal arrival is coincident with the vertical shifting, image smearing of up to one pixel is possible. The amount of smearing is a function of the signal duration compared to the single vertical shift time.

ST138

Store Enable Option

This Store Enable option is available with the Freerun, External Sync and Continuous Cleans timing modes. It is an option that can be used when frames must be collected at a constant rate, but only selected ones of these frames need to be stored.

The camera continuously repeats the readout/exposure cycle, either with or without external synchronization. At the start of each readout the controller checks to see if a pulse was received by the External Trigger port on the back of the controller. If an External Trigger was received at any time since the beginning of the previous readout, the digitized data is sent to the computer. If no trigger occurs during this time the digitized data is discarded. Figure 27 shows the basic Store Strobe timing. The signals are TTL logic signals, from 0 to 5 V.

The second trigger pulse in the timing diagram in Figure 27 arrives during a readout. The data that is currently being read out is discarded, but the trigger event is stored in a "latch." At the beginning of the subsequent readout this latch is checked and reset. Since the latch had recorded a pulse the last readout in the diagram is digitized.

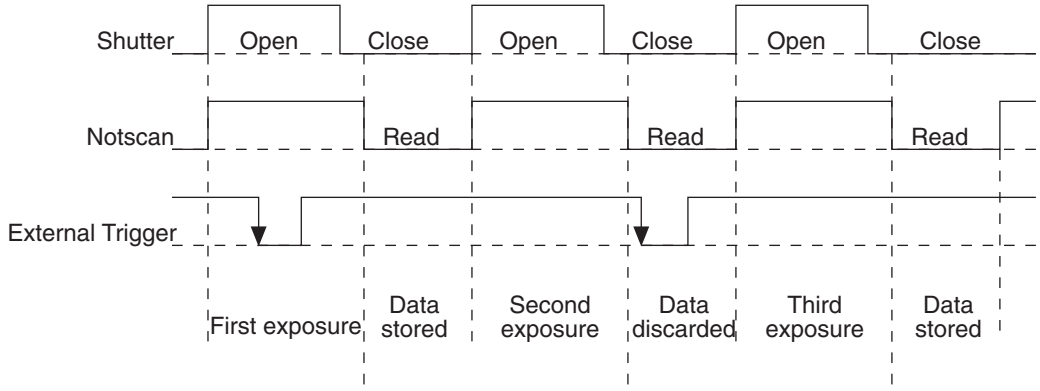


Figure 27. Store Enable timing diagram

This mode is most useful when exposures must always be taken at precise intervals, but only some frames are useful. For data where events are erratic but a background must be subtracted precisely, this option in combination with one of the timing modes described in the following paragraphs offers the highest performance.

Standard Triggering Modes

The three triggering modes available for any ST-138 based system are Freerun, External Sync, and Continuous Cleans. These timing modes are combined with the Store Enable and Shutter options to provide the widest variety of timing modes for precision experiment synchronization.

Shutter Modes

The shutter options available include Normal, Preopen, or Disable. Disable simply means that the shutter will not open at all during the experiment. This is only useful for making dark charge measurements, or when no shutter is present in the system. Preopen, available in External Sync or Continuous Cleans mode, opens the shutter as soon as the controller is ready to receive an External Sync pulse but before the pulse is received. This is required if the time between the External Sync pulse and the event is less than the few milliseconds it takes the shutter to open.

Freerun timing

In the Freerun mode the controller does not synchronize with the experiment in any way. The shutter opens as soon as the previous readout is complete, and remains open for the exposure time, t_{exp}. Any External Sync or External Trigger signals are ignored. This mode is useful for experiments with a constant light source, such as a CW laser or a DC lamp. Other experiments that can utilize this mode are high repetition studies, where the number of shots that occur during a single shutter cycle is so large that it appears to be continuous illumination.

Other experimental equipment can be synchronized to the controller by using the signals provided at the Trigger Out, Shutter Monitor, or Notscan port. These signals are all TTL logic signals, from 0 to 5 V.

External Sync timing

In this mode all exposures are synchronized to an external source. This mode can be used in combination with Normal or Preopen Shutter operation. In the Normal Shutter mode, the controller waits for an External Sync pulse, then opens the shutter for the programmed exposure period. As soon as the exposure is complete the shutter closes and the CCD data are read out. The time required for the shutter to open completely depends on the shutter type, typically 5 ms for a small or remote shutter, and from 16 to 20 ms for a large shutter.

Since the shutter requires up to 10 msec to fully open, the External Sync pulse provided by the experiment must precede the actual signal by at least that much time. If not, the shutter will not be open during the entire signal, or the signal may be missed completely.

Also, since the amount of time from the initialization of the experiment to the first trigger pulse is not fixed, an accurate background subtraction may not be possible for the first readout. In multiple-shot experiments this is easily overcome by simply discarding the first frame.

In the Preopen Shutter mode, on the other hand, shutter operation is only partially synchronized to the experiment. As soon as the controller is ready to collect data the shutter opens. Upon arrival of the first External Sync pulse to the controller, the shutter remains open for the specified exposure period, closes, and the CCD is readout. As soon as readout is complete the shutter re-opens and waits for the next frame.

The Preopen mode is useful in cases where an External Sync pulse cannot be provided 5-10 msec before the actual signal occurs. Its main drawback is that the CCD is exposed to any ambient light incident on the detector while the shutter is open between frames. If this ambient light is constant, and the triggers occur at regular intervals, this background can also be subtracted, providing that it does not saturate the CCD. As with the Normal Shutter mode, accurate background subtraction may not be possible for the first frame.

Also note that in addition to signal from ambient light, dark charge accumulates during the "wait" time (t_w). Any variation in trigger frequency also affects the amount of dark charge, even if light is not falling on the CCD during this time.

Continuous Cleans timing

The third timing mode available with the ST-138 Controller is called Continuous Cleans. In addition to the standard "cleaning" of the array, which occurs after the controller is enabled, Continuous Cleans will remove any charge from the array until the moment the External Sync pulse is received. This continuous cleaning is accomplished by driving both the vertical and horizontal clocks of the chip simultaneously.

Once the External Sync pulse is received, cleaning of the array stops as soon as the current row is shifted, and frame collection begins. With Normal Shutter operation the shutter is opened for the set exposure time. With Preopen Shutter operation the shutter is open during the continuous cleaning, and once the External Sync pulse is received the shutter remains open for the set exposure time, then closes.

If the vertical rows are shifted midway when the External Sync pulse arrives, the pulse is saved until the row shifting is completed, to prevent the CCD from getting "out of step." As expected, the response latency is on the order of one vertical shift time, from 1-30 µsec depending on the array. This latency does not prevent the incoming signal from being detected, since photo generated electrons are still collected over the entire active area. However, if the signal arrival is coincident with the vertical shifting, image smearing of up to one pixel is possible. The amount of smearing is a function of the signal duration compared to the single vertical shift time.

ST130

Introduction

The triggering modes available for any ST-130 or ST-135 based system are Freerun and External Sync. These timing modes are combined with the Store Enable and Shutter options to provide the widest variety of timing modes for precision experiment synchronization.

Shutter Options

The shutter options include Normal, (where no options are selected), Preopen, or Disable. Disable simply means that the shutter will not open at all during the experiment. This is only useful for making dark charge measurements, or when no shutter is present in the system. Preopen, available in External Sync or Continuous Cleans mode, opens the shutter as soon as the controller is ready to receive an External Sync pulse. This is required if the time between the External Sync pulse and the event is less than a few milliseconds, the time it takes the shutter to open.

Freerun

In the Freerun mode the controller does not synchronize with the experiment in any way. The shutter opens as soon as the previous readout is complete, and remains open for the exposure time, t_{exp}. Any External Sync or External Trigger signals are ignored. This mode is useful for experiments with a constant light source, such as a CW laser or a DC lamp. Other experiments that can utilize this mode are high repetition studies, where the number of shots that occur during a single shutter cycle is so large that it appears to be continuous illumination.

Other experimental equipment can be synchronized to the controller by using the signals at the Trigger Out, Shutter Monitor, or Notscan port. All are TTL logic signals, from 0 to 5 V.

External Sync timing & Shutter Modes

In this mode all exposures are synchronized to an external source. This mode can be used in combination with Normal or Preopen Shutter operation. In the Normal Shutter mode, the controller waits for an External Sync pulse, then opens the shutter for the programmed exposure period. As soon as the exposure is complete the shutter closes and the CCD data are read out. The shutter requires 5-10 msec to open completely.

Since the shutter requires up to 10 msec to fully open, the External Sync pulse provided by the experiment must precede the actual signal by at least that much time. If not, the shutter will not be open during the entire signal, or the signal may be missed completely.

Also, since the amount of time from the initialization of the experiment to the first trigger pulse is not fixed, an accurate background subtraction may not be possible for the first readout. In multiple-shot experiments this is easily overcome by simply discarding the first frame.

In the Preopen Shutter mode, on the other hand, shutter operation is only partially synchronized to the experiment. As soon as the controller is ready to collect data the shutter opens. Upon arrival of the first External Sync pulse to the controller, the shutter remains open for the specified exposure period, closes, and the CCD is readout. As soon as readout is complete the shutter re-opens and waits for the next frame.

The Preopen mode is useful in cases where an External Sync pulse cannot be provided 5-10 msec before the actual signal occurs. Its main drawback is that the CCD is exposed to any ambient light incident on the detector while the shutter is open between frames. If this ambient light is constant, and the triggers occur at regular intervals, this background can also be subtracted, providing that it does not saturate the CCD. As with the Normal Shutter mode, accurate background subtraction may not be possible for the first frame.

Also note that in addition to signal from ambient light, dark charge accumulates during the “wait” time (t_w). Any variation in trigger frequency also affects the amount of dark charge, even if light is not falling on the CCD during this time.

ST12x

Introduction

The ST12x controllers allow considerable flexibility with regard to ways of synchronizing data collection with the user's experiment. The timing modes are described below.

Freerun

In the Freerun mode, there is no synchronization with the experiment. The array is simply scanned once every Exposure period. This mode is useful for experiments with a constant light source, such as a CW laser or a DC lamp. Other experiments that can utilize this mode are high repetition studies, where the number of shots that occur during a single exposure is so large that it appears to be continuous illumination.

External Sync

External Sync mode operates based on a TTL low pulse sent to the External Sync input on the rear panel of the controller. Readout continues until the number of Accumulations is reached. As long as the exposure time is set to the minimum value, data storage begins immediately after the sync pulse.

For ST-120 systems in this mode, Exposure acts as a trigger counter. Array readout will occur only after the number of external sync pulses equals the number of Controller Units set in Exposure. For example, setting Exposure to 5 will result in a spectrum being

stored after every 5 external sync pulses. This mode only works for external sync frequencies less than the maximum readout frequency.

For ST-121 systems in this mode, Exposure acts as a programmable delay between the external sync trigger pulse and the actual scan of the array.

This mode of operation is most useful when the same number of events must be stored per spectrum.

Line Sync

Line Sync synchronizes the scanning of the array to the AC power line frequency, either 50 or 60 Hz. In this mode, as in External Sync, Exposure acts as a programmable delay between the line sync pulse and the actual scan of the array. Data storage begins the next time diode #1 is read. This mode reduces signal modulations due to line frequency.

External Trigger

External trigger mode begins storing data upon arrival of a TTL low pulse to the Trigger In BNC on the rear panel of the controller. Array scanning is identical to Freerun mode, once again controlled by the Exposure parameter. Data storage operates independently of array readout. If a pulse is received by the Trigger In port and the array is not being read, the next complete spectrum is stored.

If a pulse is received by the Trigger In port *during* readout of the array, *the next diode is stored.* Readout continues to the end of the array, waits the programmed Exposure, then reads the array only until the spectrum is completed. For example, if the trigger came during readout of diode 200, diode 201 on (to the last diode of the array) will be stored. After the Exposure period (immediately if the Exposure is set to the minimum value) diodes 1 to 200 would be read out.

Whatever pattern is established by the first spectrum is repeated for every spectrum. The software automatically shifts the two parts of the spectrum to display the diodes in the correct order.

Unlike External Sync, this mode allows accurate background subtraction since the time between scans is always the same. Make sure that the Store flag at the bottom of the screen is toggled on.

External Sync and External Trigger

External Sync and External Trigger mode is a combination of the features of External Sync mode and External Trigger mode. *Readout* of the array is controlled by TTL signals through the External Sync BNC. *Storage* of the data is controlled by TTL signals through the External Trigger BNC. As in the description for External Trigger mode, if an External Sync pulse arrives during the readout of the array, storage begins with *the next diode.*

Spectral data is handled the same way as the External Trigger case. Again, Exposure acts as a delay between the Ext Sync pulse and the start of the scan.

Line Sync and External Trigger

Line Sync and External Trigger mode is nearly identical to the External Trigger mode, except that the readout of the array is synchronized to the line frequency.

Event Counter

Event Counter mode (ST-121 only) allows the ST-121 to collect data after an integer number of External Sync pulses. The Exposure value, rather than setting the delay between the sync pulse and the readout, instead determines how many sync pulses must arrive before each scan is started. For example, if the minimum readout time is 5 msec, setting the Exposure to 20 msec instructs the controller to wait until 4 External Sync pulses are received before reading out the array. The time between pulses need not be constant, and the maximum pulse rate is 1 MHz.

Store Enable Option

The store enable option is found in combination with Freerun, External Sync, and Line Sync. Readout of the array operates identically to the corresponding mode, e.g., in the Line Sync, Store Enable mode, readout is synchronized to the line frequency.

At the moment the array readout begins, the controller checks an internal flag to determine if a TTL low pulse was received by the External Trigger BNC during the last Readout/Exposure time period. If not, the array is read and the data discarded.

If a pulse was received any time since the beginning of the previous readout, the flag is cleared and the readout is stored. Notice that any External Trigger received during a readout only affects the *next* readout. This is an important distinction.

Note: Any External Trigger signal arriving at the controller before the system starts looking for them (i.e., before the Trigger Out output goes low) are ignored by the system, and will not result in a spectrum being sent to the computer.

In External Sync, Store Enable mode, the Exposure parameter again acts as a delay between the External Sync pulse and the readout of the array.

Acquisition Timing Mode Definitions

CTRL_FREERUN

Establishes Freerun mode operation in all controllers. Because the details of Freerun mode operation differ depending on the controller model, you should see the description of the Freerun Mode for your particular controller. The location of each description is as follows.

PentaMAX:	page 132 (freerun timing)
MicroMAX:	page 134 (freerun timing)
ST138:	page 136 (freerun timing)
ST130:	page 138 (freerun)
ST12x:	page 139 (freerun)

CTRL_LINESYNC

Establishes line synchronized operation in the ST120 and ST121 controllers. For details, see the discussion of Line Sync operation on page 140.

CTRL_EXTSYNC_NORMAL

Establishes Ext Sync operation in the Normal shutter mode in all controllers. Because the details will differ depending on the controller model, you should see the description of the Timing Modes for your particular controller. The starting page for the individual controller discussions are as follows.

PentaMAX:	page 132 (Shutter Modes and External Sync Timing)
MicroMAX:	page 133 (Shutter Modes and External Sync Timing
ST138:	page 135 (Shutter Modes and External Sync Timing)
ST130:	page 138 (Shutter Modes and Ext Sync Timing and Shutter Options)
ST12x:	page 139 (External Sync)

Note that for the ST12x Controller, `CTRL_EXTSYNC`, `CTRL_EXTSYNC_NORMAL` and `CTRL_EXTSYNC_PREOPEN` all do the same thing. There is no shutter on diode arrays.

In the case of the ST130 and ST138, `CTRL_EXTSYNC_NORMAL` establishes the following conditions.

Mode: External Sync
Store Enable: Off
Shutter: Normal

CTRL_EXTSYNC_PREOPEN

Establishes Ext Sync operation in the Preopen shutter mode in all controllers. Because the details will differ depending on the controller model, you should see the description of the Timing Modes for your particular controller. The starting page for the individual controller discussions are as follows.

PentaMAX:	page 132 (Shutter Modes and External Sync Timing)
MicroMAX:	page 133 (Shutter Modes and External Sync Timing)
ST138:	page 135 (Shutter Modes and External Sync Timing)
ST130:	page 138 (Shutter Modes and Ext Sync Timing and Shutter Options)
ST12x:	page 139 (External Sync)

Note that for the ST12x Controller, `CTRL_EXTSYNC, CTRL_EXTSYNC_NORMAL` and `CTRL_EXTSYNC_PREOPEN` all do the same thing. There is no shutter on diode arrays.

In the case of the ST130 and ST138, `CTRL_EXTSYNC_PREOPEN` establishes the following conditions.

Mode: External Sync
Store Enable: Off
Shutter: Preopen

CRTL_EXTTRIG_NORMAL

Establish Ext Trigger mode operation in the Normal shutter mode for ST138 and ST12x controllers. Because the details will differ depending on the controller model, you should see the description of the Timing Modes for your particular controller. The starting page for the individual controller discussions are as follows.

ST138: page 135 (External Sync Timing and Continuous Cleans Timing)
ST12x: page 139 (External Trigger)

The operation for an ST12x controller is as follows. The array cleans until Ext Trig goes high. Then it reads out the array starting at whatever pixel it was at when the trigger was detected.

In the case of the ST138, `CRTL_EXTTRIG_NORMAL` establishes the following conditions.

Mode: Continuous Cleans
Store Enable: Off
Shutter: Normal

CTRL_EXTTRIG_PREOPEN

Establish Ext Trigger mode operation in the Preopen shutter mode for ST138 and ST12x controllers. Because the details will differ depending on the controller model, you should see the description of the Timing Modes for your particular controller. The starting page for the individual controller discussions are as follows.

ST138: page 135 (External Sync Timing and Continuous Cleans Timing)
ST12x: page 139 (External Trigger)

Operation will the same as for `CTRL_EXTTRIG` described above..

In the case of the ST138, `CRTL_EXTTRIG_PREOPEN` establishes the following conditions.

Mode: Continuous Cleans
Store Enable: Off
Shutter: Preopen

CTRL_FR_STORE_TRIG

Establishes Store Enable operation (Freerun mode) in ST138, ST130 and ST121 controllers. Because the details will differ depending on the controller model, you should see the description of the Timing Modes for your particular controller. The starting page for the individual controller discussions are as follows.

ST138: page 135 (Store Enable Option)
ST121: page 139 (Store Enable Option)

In the case of the ST138, `CTRL_FR_STORE_TRIG` establishes the following conditions.

Mode: Freerun
Store Enable: On
Shutter: Normal

CTRL_SN_STORE_TRIG

Establishes Store Enable operation (External Sync) in the ST138, ST130 & ST121 controllers. Because the details will differ depending on the controller model, you should see the description of the Timing Modes for your particular controller. The starting page for the individual controller discussions are as follows.

ST138: page 135 (Store Enable Option)
ST121: page 139 (Store Enable Option)

In the case of the ST138, `CTRL_SN_STORE_TRIG` establishes the following conditions.

Mode: External Sync
Store Enable: On
Shutter: Normal

CTRL_SP_STORE_TRIG

Establishes Store Enable operation (External Sync) in the ST138, ST130 & ST121 controllers. Because the details will differ depending on the controller model, you should see the description of the Timing Modes for your particular controller. The starting page for the individual controller discussions are as follows.

ST138: page 135 (Store Enable Option)
ST121: page 139 (Store Enable Option)

In the case of the ST121 controller, it works the same as `CTRL_SN_STORE_TRIG`.

In the case of the ST138, `CTRL_SP_STORE_TRIG` establishes the following conditions.

Mode: External Sync
Store Enable: On
Shutter: Preopen

CTRL_TN_STORE_TRIG

Establishes Store Enable operation with continuous cleans (ST138 only). *See the discussion of the ST138 timing modes on page 135(store enable option, continuous cleans timing).*

`CTRL_TN_STORE_TRIG` establishes the following conditions.

Mode: Continuous Cleans
Store Enable: On
Shutter: Normal

CTRL_TP_STORE_TRIG

Establishes Store Enable operation with continuous cleans (ST138 only). *See the discussion of the ST138 timing modes on page 135 (store enable option, continuous cleans timing).*

`CTRL_TP_STORE_TRIG` establishes the following conditions.

Mode: Continuous Cleans
Store Enable: On
Shutter: Preopen

CTRL_EVENT_COUNTER

Activates the Event Counter function in the ST121 controller (only). The Event Counter is described on page 141.

CTRL_EXT_SYNC_EXT_TRIG

In the ST121 (only), `CTRL_EXT_SYNC_EXT_TRIG` establishes simultaneous External Sync and External Trigger operation together. *See the discussion of the ST-121 timing modes on page 139 (line sync and external trigger).*

CTRL_LINE_SYNC_STR_ENA

In the ST121 (only), `CTRL_LINE_SYNC_STR_ENA` establishes simultaneous Line Sync and Store Enable operation. *See the discussion of the ST121 timing modes on page 139 (line sync and external trigger).*

This page intentionally left blank.

Index

A

A/D rate, selecting....45
Add/Remove Programs....31
application_type....69
Application_type....44
Asynchronous mode acquisition....57
Auto-Stop hardware feature....45

B

Background subtraction....133, 134
buffer, size....53

C

Calibration Structures....117
CCD Chips....127
collecting data....53
Compatibility of WinView/16 and WinView/32....30
Compiler Settings....35
Computer requirement....21
Continuous Cleans timing....137
Continuous Cleans....134
Control Panel....30
Controller parameters....47
Controller and Array types....41
Controller Object
  creating....41
  destroying....44
Controller_type....41, 69, 70, 78
`Create/Set/Initialize/Start/ChkData/Cleanup`....111

D

Dark charge....133, 134
Data acquisition Events....58
data collection....53
data overrun....36, 57
Data_Collection_Mode....41, 44, 70, 78
Detector_type....41, 71, 72, 78, 88
Device drivers....37
Diode Array....*See* Detector_type
Diode Arrays....128

E

EISA Interface card....37
Example
  file write....123
Example program....63
Exit Setup button....29
exposure setting....48
External Sync
  background subtraction....133, 134
  dark charge accumulation....133, 134
  input pulse....132, 134
  shutter synchronization....133, 134
  timing....132, 134
  trigger mode....132
External Sync Normal....136, 138
External Synchronization....132, 134

F

Files....33
Focus mode....58
Free Run
  experiments best suited for....134
  timing....134
Freerun....136, 138
  experiments best suited for....132
  timing....132
  trigger mode....132

G

Graphics card....21

H

Hard disk requirement....20
hardware initialization....53
Hardware requirements....20
Header structure with offsets....118
Headers and Calibration structures
  Ver 1.43 Header....113
  Ver 1.6 Header....114

I

Imaging Option....49
initializing
  hardware....53
  software....78
installing
  EasyDLL95
    aborting installation....29
    from disk....21
    from FTP site....22
    installing other files later....29
    multiple versions....30
    typical, compact or custom....26
  PICM for Windows 3.1....19

installing (cont.)
PICM for Windows/95 20
interface card..... 47
Interface Limitations
ISA..... 36
PCI..... 37
Interface_card function 73
ISA interface card 36

L

Latency..... 135

M

Manual, using..... 17
memory
allocation 36, 53
buffer 53
clean up..... 44, 74
considerations and requirements..... 36
requirements 20
Mouse requirement..... 21
multiple controllers
ISA..... 36
PCI..... 37
Multithreaded, multitasking environments 58

N

Named Event..... 59
nframe mode..... 36
Nframe mode..... 57
NT log-on requirements 21

O

Operating system requirement..... 21

P

PCI interface card..... 37
Pentium 21
PI_PCI.SYS 37
PICM
basic operation of..... 16
files for Windows 3.1..... 19
files for Windows 95..... 32
functionality..... 15
functions 69
installing for Windows 95..... 20
introduction to..... 15
overview 35
system requirements..... 20
Windows 3.1 Example Programs..... 16
focus 17
focuspen 17
General 16
imaging..... 17
imgx..... 17
kinetics 17
PICM Windows Example Programs (cont.)
multfram 17
multstrp..... 17
writing code with PICM functions 35
PICM__Get_Actual_Temperature..... 83
PICM_ChkData 53, 54, 57, 58, 73, 74, 96, 108
`PICM_ChkData( &error )` 56
PICM_CleanUp 36, 44, 45, 54, 55, 74, 75, 108, 109, 111
PICM_Clear_MultStrip 75
PICM_Clear_user_rois..... 76
PICM_CMGetDoubleParam 76
PICM_CMGetLongParam..... 77
PICM_CMSetDoubleParam..... 77
PICM_CMSetLongParam 51, 77
PICM_CreateController 36, 41, 44, 69, 70, 73, 78, 79, 88, 111
PICM_CreateControllerNvram 79, 80
PICM_Download_MultROI..... 80
PICM_Easy_Focusing..... 80
PICM_FindController 81
PICM_FindPCICards 82
PICM_Generate_sideeffects..... 82
PICM_Get_acqmode..... 83
PICM_Get_AutoStop 83
PICM_Get_cleanscans 84
PICM_Get_controller_version 84
PICM_Get_MinBlk..... 85
PICM_Get_normal_rois 85
PICM_Get_num_strips_per_clean 86
PICM_Get_NumMinBlk..... 86
PICM_Get_pixeldimension_x..... 61, 86
PICM_Get_pixeldimension_y..... 61, 86, 87
PICM_Get_RS170_enable 87
PICM_Get_sensor_x 61, 87, 88
PICM_Get_sensor_y 61, 87, 88
PICM_Get_shutter_type..... 88
PICM_Get_shuttermode..... 88
PICM_Get_Temperature..... 88
PICM_Get_Temperature_Status 89
PICM_Get_TTL_pattern..... 61, 89
PICM_GetEnumParam..... 84
PICM_GetEnumString 85
PICM_Initialize_RS170 89
PICM_Initialize_System .....36, 53, 54, 56, ...58, 74, 90, 111
PICM_IsAvail 91
PICM_LoadNvramDefaults..... 95
`PICM_LockCurrentFrame` ...55, 57, 58, ...96
PICM_ResetUserBuffer 96
PICM_Set_acqmode..... 97
PICM_Set_AutoStop..... 57, 97
PICM_Set_cleanscans..... 98

PICM_Set_controller_version....97
PICM_Set_EasyDLL_DC....58, 98
PICM_Set_Fast_ADC....99
PICM_Set_MinBlk....101
PICM_Set_MultiStrip_Flag....101
PICM_Set_num_strips_per_clean....102
PICM_Set_NumMinBlk....102
PICM_Set_RS170_enable....104
PICM_Set_shutter....105
PICM_Set_shutter_type....105
PICM_Set_shuttermode....105
PICM_Set_Slow_ADC....106
PICM_Set_Temperature....106
PICM_Set_TTL_pattern....61, 106
PICM_SetExposure....48, 99
PICM_SetInterfaceCard....47, 73, 100, 101, 111
PICM_SetNewUserBuffer....101
PICM_SetROI....49, 103
PICM_SetROI_MultiStrip....104
PICM_SetUserEvent....58, 107
PICM_Setxxx....111
PICM_SizeNeedToAllocate....53, 107, 108
PICM_Start_controller....53, 54, 74, 108
PICM_Start_Controller....108
PICM_Stop_controller....54, 108, 109
PICM_Stop_Controller....108
PICM_UnlockCurrentFrame....55, 110
pievtfcn.h....58
pigendef.h....58
pigenfcn.h....55
pigenfcn.h....58
pimltfcn.h....57
PIVXDISA.VXD....37
PIVXDPCI.VXD....37
PIXCM.DLL....15
pixel dimensions....61
Pre Open Shutter mode....134
Preopen Shutter mode....133
Programming Reference....69

R

Reading strips and data....122
Reg files....39
Registry....21
  NT....39
  Windows 95....38
Response latency....135
RS170....51

S

sensor size....61
Setup program....21
Shutter mode
  preopen....133
Shutter modes
  Disable....132
  Normal....132
  Preopen....132, 134
Synchronous mode acquisition....57

T

Timing
  ST-121....139
Timing mode definitions
  acquisition....141
Timing modes....131, 133
  Continuous Cleans....134
  table of....132
Triggering modes....132, 136
TTL signals....61

U

Uninstalling and reinstalling....30
Utility functions....61

V

Video Protocol....51

W

Windows 95 Registry....38
Windows NT Registry....39
WinView/32 & WinView/16 in same computer....30

This page intentionally left blank.