PICam™ 5.x Programmer’s Manual

4411-0161
Issue 8
January 7, 2020

www.princetoninstruments.com
Revision History

<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>List of Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue 8</td>
<td>January 7, 2020</td>
<td>Issue 8 of this document incorporates the following changes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated the copyright year.</td>
</tr>
<tr>
<td>Issue 7</td>
<td>June 27, 2019</td>
<td>Issue 7 of this document incorporates the following changes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Global removal of Fax number.</td>
</tr>
<tr>
<td>Issue 6</td>
<td>March 14, 2019</td>
<td>Issue 6 of this document incorporates the following changes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added support for 532 Laser;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Rebranded to Teledyne Princeton Instruments</td>
</tr>
<tr>
<td>Issue 5</td>
<td>August 30, 2018</td>
<td>Issue 5 of this document incorporates the following changes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added PI-MTE3 support;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added FERGIE-ISO-81 support;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Clarified usage of Acquisition-related APIs.</td>
</tr>
<tr>
<td>Issue 4</td>
<td>January 2, 2018</td>
<td>Issue 4 of this document incorporates the following changes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated the copyright date.</td>
</tr>
<tr>
<td>Issue 3</td>
<td>November 27, 2017</td>
<td>Issue 3 of this document incorporates the following changes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated Linux requirements;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated list of supported devices;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated list of sample codes provided;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added PicamCenterWavelengthStatus_FAULTED to Section 5.1.1.5, PicamWavelengthStatus.</td>
</tr>
<tr>
<td>Issue 2</td>
<td>September 7, 2017</td>
<td>Issue 2 of this document incorporates the following changes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added support for Sophia 2048:13.5(\mu) cameras;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added 18-bit support (future enhancement.)</td>
</tr>
<tr>
<td>Issue 1</td>
<td>May 9, 2017</td>
<td>This is the initial release of this document.</td>
</tr>
</tbody>
</table>
# Table of Contents

## Chapter 1: About this Manual
1.1 Manual Organization .................................................. 9
1.2 Potential Compatibility Concerns .................................... 10

## Chapter 2: Introduction to PICam™
2.1 System Overview ......................................................... 13
2.2 Hardware Support ......................................................... 13
2.2.1 Camera Firmware [GigE Cameras Only] .......................... 13
2.3 Supported Operating Systems ......................................... 14
2.3.1 WoW64 Support ..................................................... 14
2.4 Sample Code ............................................................. 14
2.5 Naming Conventions .................................................... 16
2.6 Concepts ................................................................. 16
2.6.1 Handles ............................................................... 18
2.7 Defined Data Types ...................................................... 19
2.8 Include Files ............................................................. 20
2.8.1 Optional and Advanced Files .................................... 20

## Chapter 3: General Library APIs
3.1 Data Type Definitions ................................................... 21
3.2 Programmers’ Reference for General Use Library APIs ........... 22

## Chapter 4: Identification APIs
4.1 Data Type Definitions ................................................... 30
4.2 Structure Definitions ................................................... 31
4.3 Programmers’ API Reference .......................................... 33
4.3.1 Identification APIs ................................................ 34
4.3.2 Access APIs ........................................................ 44
4.3.3 Information APIs .................................................. 58
4.3.4 Demo Camera Identification APIs ............................... 62

## Chapter 5: Configuration APIs
5.1 Data Type Definitions ................................................... 68
5.1.1 Hardware Parameter Enumerations .............................. 68
5.1.2 Parameter Access Enumerations ................................ 102
5.1.3 Parameter Constraint Enumerations ............................. 103
5.2 Data Structure Definitions ............................................. 106
5.2.1 Camera-Specific Parameter Data Structures .................... 106
5.2.2 Shared Camera/Accessory Parameter Data Structures ........ 109
5.3 Parameter Constraints .................................................. 110
5.3.1 Camera-Specific Parameter Constraints ....................... 110
5.3.2 Shared Camera/Accessory Parameter Constraints .......... 114
Chapter 5: Programmers' Reference for Configuration APIs

5.4 Programmers' Reference for Configuration APIs

5.4.1 Camera-Specific Parameter Value APIs
5.4.2 Shared Camera/Accessory Parameter Value APIs
5.4.3 Shared Camera/Accessory Parameter Information APIs
5.4.4 Camera-Specific Parameter Constraints APIs
5.4.5 Shared Camera/Accessory Parameter Constraints APIs
5.4.6 Shared Camera/Accessory Parameter Commitment APIs

Chapter 6: Data Acquisition APIs

6.1 Data Format
6.2 Data Type Definitions
6.2.1 Data Acquisition Enumerations
6.3 Data Acquisition Data Structures
6.4 Programmers' Reference for Acquisition Control APIs

Chapter 7: Advanced Function APIs

7.1 Data Type Definitions
7.1.1 Shared Camera/Accessory Plug and Play Discovery Data Enumerations
7.1.2 Shared Camera/Accessory Access Enumerations
7.1.3 Shared Camera/Accessory Parameter Information Enumerations
7.1.4 Camera-Specific Data Acquisition Enumerations
7.2 Data Structures
7.2.1 Camera-Specific Information Data Structures
7.2.2 Camera-Specific Parameter Validation Data Structures
7.2.3 Camera-Specific Data Acquisition Data Structures
7.3 Callback Functions
7.3.1 Camera-Specific Discovery Callbacks
7.3.2 Accessory-Specific Discovery Callbacks
7.3.3 Camera-Specific Parameter Value Callbacks
7.3.4 Shared Camera/Accessory Parameter Value Callbacks
7.3.5 Camera-Specific Parameter Constraints Callbacks
7.3.6 Shared Camera/Accessory Parameter Constraints Callbacks
7.3.7 Camera-Specific Data Acquisition Callbacks
7.4 Programmers' Reference for Advanced APIs
7.4.1 Camera-Specific Advanced Discovery APIs
7.4.2 Accessory-Specific Advanced Discovery APIs
7.4.3 Camera-Specific Advanced Access APIs
7.4.4 Shared Camera/Accessory Advanced Access APIs
7.4.5 Camera-Specific Information APIs
7.4.6 Accessory-Specific Information APIs
7.4.7 Shared Camera/Accessory Advanced Information APIs
7.4.8 Camera-Specific Advanced Parameter Value APIs
7.4.9 Shared Camera/Accessory Advanced Parameter Value APIs
7.4.10 Shared Camera/Accessory Advanced Parameter Information APIs
7.4.11 Camera-Specific Advanced Parameter Constraints APIs
7.4.12 Shared Camera/Accessory Advanced Parameter Constraints APIs
7.4.13 Camera-Specific Advanced Commitment APIs
7.4.14 Camera-Specific Advanced Acquisition Setup APIs
7.4.15 Camera-Specific Advanced Acquisition Notification APIs
7.4.16 Camera-Specific Advanced Acquisition State Notification APIs
7.4.17 Camera-Specific Advanced Acquisition Control APIs
### Chapter 8: EM Calibration APIs

- **8.1 EM Calibration Applications** ................................................. 283
- **8.2 Structure Definitions** .................................................... 284
  - **8.2.1 EM Calibration Structures** ......................................... 284
- **8.3 Callback Functions** .......................................................... 285
  - **8.3.1 EM Calibration** .......................................................... 285
- **8.4 Programmers’ Reference for EM Calibration APIs** .......................... 286
  - **8.4.1 EM Calibration Access APIs** ........................................ 286
  - **8.4.2 EM Calibration Parameter Value APIs** ................................. 289
  - **8.4.3 EM Calibration Parameter Constraints APIs** .......................... 292
  - **8.4.4 EM Calibration APIs** ................................................. 293

### Appendix A: Available Parameters ............................................ 295
- **A.1 Camera Parameter Information** .......................................... 295
- **A.2 Accessory Parameter Information** ....................................... 304

### Appendix B: EM Gain Calibration Code Sample ................................. 305
- **B.1 EM Gain Calibration Procedure** ......................................... 305

### Appendix C: Firmware Upgrade/Restore ....................................... 307
- **C.1 Firmware Upgrade Procedure** ........................................... 307
- **C.2 Restore Firmware** ............................................................ 309
  - **C.2.1 Precautions** ............................................................. 309
  - **C.2.2 Procedure** ............................................................. 310

### Appendix D: Debugging GigE Cameras .......................................... 313
- **D.1 Debugging** ................................................................. 313
  - **D.1.1 Timeout Period Considerations** ..................................... 314
  - **D.1.2 Following Debugging** ................................................. 314
- **D.2 Timeout Configuration** .................................................... 314

### Appendix E: PICam 5.0 Compatibility Issues .................................. 315
- **E.1 FERGIE: 256F/FT, FERGIE: 256B/FT, FERGIE: 256BR/FT, and eXcelon Variant Cameras** ......................................................... 315
  - **E.1.1 What Changed with PICam 5.0?** ..................................... 315
  - **E.1.2 Code Updates to Retain Existing Behavior** .......................... 316
- **E.2 PI-MAX4: 2048B, PI-MAX4: 2048B-RF Cameras** ............................. 317
  - **E.2.1 What Changed with PICam 5.0?** ..................................... 317
  - **E.2.2 Code Updates to Retain Existing Behavior** .......................... 317
- **E.3 PI-MAX4: 512B/EM, PI-MAX4: 1024B/EM** .................................... 318
  - **E.3.1 What Changed with PICam 5.0?** ..................................... 318
  - **E.3.2 Code Updates to Retain Existing Behavior** .......................... 318
- **E.4 PI-MAX4: 512EM/1024EM Cameras** ......................................... 320
  - **E.4.1 What Changed with PICam 5.0?** ..................................... 320
  - **E.4.2 Code Updates to Retain Existing Behavior** .......................... 320
- **E.5 PI-MTE: 1300B/1300BR Cameras** ........................................... 321
  - **E.5.1 What Changed with PICam 5.0?** ..................................... 321
  - **E.5.2 Code Updates to Retain Existing Behavior** .......................... 321
- **E.6 PI-MTE: 1300R Cameras** .................................................... 322
  - **E.6.1 What Changed with PICam 5.0?** ..................................... 322
  - **E.6.2 Code Updates to Retain Existing Behavior** .......................... 322
- **E.7 PIXIS: 100B/100BR/400B/400BR/1300B/1300BR, and XO/XF/XB/ eXcelon Variant Cameras** ......................................................... 323
  - **E.7.1 What Changed with PICam 5.0?** ..................................... 323
  - **E.7.2 Code Updates to Retain Existing Behavior** .......................... 323
E.8 PIXIS: 100F/100R/100C/400F/400R/1300F/1300F-2, and
   XB Variant Cameras ........................................... 324
E.8.1 What Changed with PICam 5.0? ................................ 324
E.8.2 Code Updates to Retain Existing Behavior ................. 324
E.9.1 What Changed with PICam 5.0? ................................ 325
E.9.2 Code Updates to Retain Existing Behavior ................. 325
E.10 ProEM Cameras (All Models) .................................. 326
E.10.1 What Changed with PICam 5.0? .............................. 326
E.10.2 Code Updates to Retain Existing Behavior ................. 326
E.11 ProEM-HS: 1KB-10 and eXcelon Variant Cameras ............. 327
E.11.1 What Changed with PICam 5.0? .............................. 327
E.11.2 Code Updates to Retain Existing Behavior ................. 327
E.12 ProEM-HS: 512B/512BK/1024B and eXcelon Variant Cameras 328
E.12.1 What Changed with PICam 5.0? .............................. 328
E.12.2 Code Updates to Retain Existing Behavior ................. 328
E.13 ProEM+ (All Models) .......................................... 330
E.13.1 What Changed with PICam 5.0? .............................. 330
E.13.2 Code Updates to Retain Existing Behavior ................. 330
E.14 PyLoN: 100B/100BR/400B/400BR/1300B/1300BR, and eXcelon Variant Cameras .............................................. 331
E.14.1 What Changed with PICam 5.0? .............................. 331
E.14.2 Code Updates to Retain Existing Behavior ................. 331
E.15 PyLoN: 100F/400F/1300F/1300R Cameras ..................... 332
E.15.1 What Changed with PICam 5.0? .............................. 332
E.15.2 Code Updates to Retain Existing Behavior ................. 332

List of Figures

Figure 2-1: Basic PICam Structure .................................. 17
Figure 2-2: Block Diagram of Camera-Specific Handle Hierarchy 18
Figure 6-1: Data Format Diagram .................................. 180
Figure 7-1: PICam Structure - Advanced .......................... 191
Figure 8-1: Typical EM Gain Calibration Dialog ................. 306
Figure C-1: Firmware Upgrade: Typical IP Engine Selection Dialog 307
Figure C-2: Firmware Upgrade: Selecting Device to be Upgraded. 308
Figure C-3: Firmware Upgrade: Typical Updating Dialog ........ 308
Figure C-4: Firmware Upgrade: Upgrade Complete ............... 309
Figure C-5: Firmware Restore: Typical IP Engine Selection Dialog 310
Figure C-6: Firmware Restore: Selecting Device to be Restored 310
Figure C-7: Firmware Restore: Typical Updating Dialog .......... 311
Figure C-8: Firmware Restore: Complete .......................... 311
List of Tables

Revision History ........................................................................................................... 2
Table 1-1: Index to Code Updates for PiCam 5.0 Support, by Camera ......................... 10
Table 2-1: List of Sample Code Files Provided ............................................................ 14
Table 2-2: Data Type Definitions .................................................................................. 19
Table 3-1: Data Enumeration Definitions for General Library APIs ............................. 21
Table 4-1: Data Type Definitions for Hardware APIs .................................................. 30
Table 5-1: PicamActiveShutter Enumerator Definitions ............................................. 69
Table 5-2: PicamAdcAnalogGain Enumerator Definitions .......................................... 69
Table 5-3: PicamAdcQuality Enumerator Definitions .................................................. 70
Table 5-4: PicamCcdCharacteristicsMask Enumerator Definitions ........................... 71
Table 5-5: PicamCenterWavelengthStatus Enumerator Definitions ......................... 72
Table 5-6: PicamConstraintType Enumerator Definitions ......................................... 72
Table 5-7: PicamCoolingFanStatus Enumerator Definitions ..................................... 73
Table 5-8: PicamEMIccdGainControlMode Enumerator Definitions ......................... 73
Table 5-9: PicamGateTrackingMask Enumerator Definitions .................................... 74
Table 5-10: PicamGatingMode Enumerator Definitions ............................................. 74
Table 5-11: PicamGatingSpeed Enumerator Definitions ............................................. 75
Table 5-12: PicamGratingCoating Enumerator Definitions ....................................... 75
Table 5-13: PicamGratingType Enumerator Definitions ............................................ 76
Table 5-14: PicamIntensifierOptionsMask Enumerator Definitions ......................... 76
Table 5-15: PicamIntensifierStatus Enumerator Definitions ...................................... 77
Table 5-16: PicamLaserOutputMode Enumerator Definitions .................................... 77
Table 5-17: PicamLaserStatus Enumerator Definitions ............................................... 77
Table 5-18: PicamLightSource Enumerator Definitions ............................................. 78
Table 5-19: PicamLightSourceStatus Enumerator Definitions ................................... 78
Table 5-20: PicamModulationTrackingMask Enumerator Definitions ...................... 79
Table 5-21: PicamOrientationMask Enumerator Definitions ...................................... 79
Table 5-22: PicamOutputSignal Enumerator Definitions ........................................... 80
Table 5-23: PicamParameter Enumerator Definitions ............................................... 81
Table 5-24: PicamPhosphorType Enumerator Definitions ......................................... 92
Table 5-25: PicamPhotocathodeSensitivity Enumerator Definitions ......................... 92
Table 5-26: PicamPixelFormat Enumerator Definitions ............................................. 93
Table 5-27: PicamPixelFormat Enumerator Definitions ............................................. 93
Table 5-28: PicamReadoutControlMode Enumerator Definitions ............................. 94
Table 5-29: PicamSensorTemperatureStatus Enumerator Definitions .................... 94
Table 5-30: PicamSensorType Enumerator Definitions ............................................. 95
Table 5-31: PicamShutterStatus Enumerator Definitions .......................................... 95
Table 5-32: PicamShutterTimingMode Enumerator Definitions .................................. 96
Table 5-33: PicamShutterType Enumerator Definitions ............................................. 97
Table 5-34: PicamTimeStampsMask Enumerator Definitions ..................................... 97
Table 5-35: PicamTriggerCoupling Enumerator Definitions ...................................... 98
Table 5-36: PicamTriggerDetermination Enumerator Definitions ............................. 98
Table 5-37: PicamTriggerResponse Enumerator Definitions ...................................... 99
Table 5-38: PicamTriggerSource Enumerator Definitions .......................................... 99
Table 5-39: PicamTriggerStatus Enumerator Definitions ......................................... 100
Table 5-40: PicamTriggerTermination Enumerator Definitions .................................. 100
Table 5-41: PicamVacuumStatus Enumerator Definitions .......................................... 100
Table 5-42: PicamValueType Enumerator Definitions ................................................ 101
Table 5-43: PicamValueAccess Enumerator Definitions ............................................ 102
Table 5-44: PicamConstraintScope Enumerator Definitions ...................................... 103
Table 5-45: PicamConstraintSeverity Enumerator Definitions ................................. 103
Table 5-46: PicamConstraintCategory Enumerator Definitions ............................... 104
Table 5-47: PicamRoisConstraintRulesMask Enumerator Definitions .................................. 105
Table 6-1: PicamAcquisitionErrorsMask Enumerator Definitions ................................. 181
Table 7-1: PicamDiscoveryAction Enumerator Definitions .................................. 192
Table 7-2: PicamHandleType Enumerator Definitions ........................................... 193
Table 7-3: PicamDynamicsMask Enumerator Definitions ........................................ 194
Table 7-4: PicamAcquisitionState Enumerator Definitions ................................... 195
Table 7-5: PicamAcquisitionStateErrorsMask Enumerator Definitions .................. 195
Table A-1: Symbol Key for Table A-2 and Table A-3 ............................................. 295
Table A-2: Parameter Information and Camera Support ........................................ 295
Table A-3: Parameter Information and Accessory Support .................................... 304
Table E-1: PICam 4.x (and Older) Code Changes: FERGIE: 256F/FT, 256B/FT, 256BR/FT, and eXcelon Variants ............... 316
Table E-2: PICam 4.x (and Older) Code Changes: PI-MAX4: 2048B/2048B-RF ..................................... 317
Table E-3: PICam 4.x (and Older) Code Changes: PI-MAX4: 512B/EM, 1024B/EM ......................................................... 318
Table E-4: PICam 4.x (and Older) Code Changes: PI-MAX4: 512EM/1024EM ... 320
Table E-5: PICam 4.x (and Older) Code Changes: PI-MTE: 1300B/1300BR .......... 321
Table E-6: PICam 4.x (and Older) Code Changes: PI-MTE: 1300R ....................... 322
Table E-7: PICam 4.x (and Older) Code Changes: PIXIS: 100B/100BR/400B/400BR/1300B/1300BR, and XO/XF/XB/eXcelon Variants ............................................. 323
Table E-8: PICam 4.x (and Older) Code Changes: PIXIS: 100F/100R/100C/400F/400R/1300F/1300F-2, and XB Variants ........ 324
Table E-9: PICam 4.x (and Older) Code Changes: PIXIS: 512F, PIXIS-XO: 512F, and PIXIS-XF: 512F .......................... 325
Table E-10: PICam 4.x (and Older) Code Changes: ProEM (All Models) ................ 326
Table E-11: PICam 4.x (and Older) Code Changes: ProEM-HS: 1KB-10 and eXcelon Variants ......................................................... 327
Table E-12: PICam 4.x (and Older) Code Changes: ProEM-HS: 512B/512BK/1024B and eXcelon Variants ............................................. 328
Table E-13: PICam 4.x (and Older) Code Changes: ProEM+ (All Models) ................ 329
Table E-14: PICam 4.x (and Older) Code Changes: PyLoN: 100B/100BR/400B/400BR/1300B/1300BR, and eXcelon Variants .... 330
Table E-15: PICam 4.x (and Older) Code Changes: PyLoN: 100F/400F/1300F/1300R ......................................................... 331
Chapter 1: About this Manual

This manual describes terms and concepts used in PICam and provides descriptions of functions, parameters, and values used to create a user-designed interface to Teledyne Princeton Instruments cameras and accessories.

This manual includes information about:

- Basic PICam functions (`picam.h`)
- Complex PICam functions (`picam_advanced.h`)
- Accessory Control functions (`picam_accessory.h`)
- EM Gain Calibration functions (`picam_em_calibration.h`)

**NOTE:**
Functions that are specific to a particular OEM are included in `picam_special.h` and are not described in this manual.

1.1 Manual Organization

This manual includes the following chapters:

- **Chapter 1, About this Manual**
  This chapter provides general information about this manual, as well as contact information for Princeton Instruments.

- **Chapter 2, Introduction to PICam™**
  This chapter provides information about concepts, terms, and data types used in PICam. It also provides information about the general sequence of making functions calls when writing a program.

- **Chapter 3, General Library APIs**
  Provides programming reference information for each of the basic functions (`picam.h`).

- **Chapter 4, Identification APIs**
  Provides programming reference information for each of the basic functions (`picam.h`).

- **Chapter 5, Configuration APIs**
  Provides programming reference pages for each of the basic functions (`picam.h`).

- **Chapter 6, Data Acquisition APIs**
  Provides programming reference pages for each of the basic functions (`picam.h`).

- **Chapter 7, Advanced Function APIs**
  Provides programming reference information about advanced functions included in `picam_advanced.h`.

- **Chapter 8, EM Calibration APIs**
  Provides programming reference information for EM Calibration functions included in `picam_em_calibration.h`. 
• Appendix A, Available Parameters
  Provides parameter information and camera support for customer-accessible parameters.

• Appendix B, EM Gain Calibration Code Sample
  Provides information about building and using the EMGainCalibration.exe sample file included with PICam.

• Appendix C, Firmware Upgrade/Restore
  Provides information about upgrading GigE camera firmware to be compatible with PICam 5.x. Information is also provided about restoring firmware to PICam 3.x.

• Appendix D, Debugging GigE Cameras
  Provides information about using the Heartbeat Timeout system variable.

• Appendix E, PICam 5.0 Compatibility Issues
  Provides information about required code modifications that may be required when upgrading to PICam 5.0 from earlier releases.

• Warranty and Service
  Provides warranty information for Princeton Instruments products. Contact information is also provided.

Wherever possible, this manual uses the headings in the PICam header files (i.e., pil_platform.h, picam.h, picam_advanced.h, picam_accessory.h, and picam_em_calibration.h) when grouping functions.

1.2 Potential Compatibility Concerns

Beginning with PICam 5.0, usage of the suite of Left/Right Margin Parameters has been modified for scenarios where Readout Orientation is not Normal. Additional information about this change is provided in Chapter E, PICam 5.0 Compatibility Issues, on page 315.

Although it is extremely rare to change any of these parameters or make coding decisions based on their values, if either of these have been incorporated in code developed for a camera listed in Table 1-1, refer to the specified section for information about coding changes required to maintain current camera behavior when upgrading to PICam 5.0.

Table 1-1: Index to Code Updates for PICam 5.0 Support, by Camera (Sheet 1 of 2)

<table>
<thead>
<tr>
<th>Camera/Camera Family</th>
<th>Section and Page #</th>
</tr>
</thead>
<tbody>
<tr>
<td>FERGIE: 256F/FT, FERGIE: 256B/FT, FERGIE: 256BR/FT, and eXcelon Variant Cameras</td>
<td>Section E.1 on page 315</td>
</tr>
<tr>
<td>PI-MAX4: 2048B, PI-MAX4: 2048B-RF Cameras</td>
<td>Section E.2 on page 317</td>
</tr>
<tr>
<td>PI-MAX4: 512B/EM, PI-MAX4: 1024B/EM</td>
<td>Section E.3 on page 318</td>
</tr>
<tr>
<td>PI-MAX4: 512EM/1024EM Cameras</td>
<td>Section E.4 on page 320</td>
</tr>
<tr>
<td>PI-MTE: 1300B/1300BR Cameras</td>
<td>Section E.5 on page 321</td>
</tr>
<tr>
<td>PI-MTE: 1300R Cameras</td>
<td>Section E.6 on page 322</td>
</tr>
<tr>
<td>PIXIS: 100B/100BR/400B/400BR/1300B/1300BR, and XO/XF/XB/eXcelon Variant Cameras</td>
<td>Section E.7 on page 323</td>
</tr>
<tr>
<td>PIXIS: 100F/100R/100C/400F/400R/1300F/1300F-2, and XB Variant Cameras</td>
<td>Section E.8 on page 324</td>
</tr>
<tr>
<td>Camera/Camera Family</td>
<td>Section and Page #</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>PIXIS: 512F, PIXIS-XO: 512F, PIXIS-XF: 512F Cameras</td>
<td>Section E.9 on page 325</td>
</tr>
<tr>
<td>ProEM Cameras (All Models)</td>
<td>Section E.10 on page 326</td>
</tr>
<tr>
<td>ProEM-HS: 1KB-10 and eXcelon Variant Cameras</td>
<td>Section E.11 on page 327</td>
</tr>
<tr>
<td>ProEM-HS: 512B/512BK/1024B and eXcelon Variant Cameras</td>
<td>Section E.12 on page 328</td>
</tr>
<tr>
<td>ProEM+ (All Models)</td>
<td>Section E.13 on page 330</td>
</tr>
<tr>
<td>PyLoN: 100B/100BR/400B/400BR/1300B/1300BR, and eXcelon Variant Cameras</td>
<td>Section E.14 on page 331</td>
</tr>
<tr>
<td>PyLoN: 100F/400F/1300F/1300R Cameras</td>
<td>Section E.15 on page 332</td>
</tr>
</tbody>
</table>
This page is intentionally blank.
Chapter 2: Introduction to PICam™

PICam is an ANSI C library of hardware control and data acquisition functions.

2.1 System Overview

To use PICam, a system must include supported hardware and a host computer with the PICam runtime installed.

2.2 Hardware Support

Version 5.x of the PICam library supports the following Princeton Instruments hardware:

- BLAZE Family
- FERGIE
- FERGIE-ISO-81
- FERGIE Accessories
- KURO
- PI-MAX3
- PI-MAX4
- PI-MAX4:RF
- PI-MAX4:EM
- PI-MTE
- PI-MTE3
- PIoNIR/NIRvana
- NIRvana-LN
- PIXIS Family
- ProEM
- ProEM+
- ProEM-HS
- PyLoN
- PyLoN-IR
- Quad-RO
- SOPHIA Family

2.2.1 Camera Firmware [GigE Cameras Only]

For GigE cameras, PICam 5.x is not backwards compatible with prior releases of PICam. Therefore, when using PICam 5.x with any GigE camera, the camera’s firmware must be PICam 5.x compatible. Upgrading PICam 3.x camera firmware is easily achieved using the Upgrade Tool supplied by Princeton Instruments.

The key symptom of a firmware mismatch between PICam and a GigE camera is the inability to see the camera from within PICam. When this occurs, the firmware within the camera must be updated to be compatible with the version of PICam being used.

- For information about installing PICam 5.x firmware onto a GigE camera with PICam 3.x firmware, refer to Section C.1, Firmware Upgrade Procedure, on page 307.
- For information about restoring firmware, refer to Section C.2, Restore Firmware, on page 309.
2.3 Supported Operating Systems
PICam currently supports the following 64-bit operating systems:
- Windows® 7;
- Windows 8/8.1;
- Windows 10;
- RedHat® Enterprise Linux®, version 7.x (RHEL7.x).

**NOTE:**
The following hardware is currently not supported by Linux PICam:
- FERGIE Accessories;
- KURO;
- PI-MTE;
- Quad-RO.

In the future, the functions described in this manual may work with additional operating systems.

2.3.1 WoW64 Support
PICam supports WoW64 which enables 32-bit programs to work with PICam and operate Princeton Instruments detectors in a 64-bit Windows operating system.

**NOTE:**
64-bit programs link with **picam.dll**.
32-bit programs link with **picam32.dll**.

2.4 Sample Code
Code samples are provided with PICam. When the PICam Software Development Kit (SDK) is installed, these samples are installed, by default, in the PICam installation directory.

**NOTE:**
The specific directory in which code samples are installed varies by operating system.

<table>
<thead>
<tr>
<th>Sample Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessory</td>
<td>This sample demonstrates control of hardware accessories.</td>
</tr>
<tr>
<td>Acquire</td>
<td>This is the basic data acquisition sample. It calls Picam_Acquire() and waits for all frames to be completed. The second part of this sample waits in a loop for N frames, acquiring 1 frame at a time.</td>
</tr>
<tr>
<td>AcquisitionState</td>
<td>This sample demonstrates an advanced acquisition scenario where the program can be notified when the camera transitions through important acquisition states (e.g., the beginning of readout.)</td>
</tr>
<tr>
<td>Advanced</td>
<td>This sample illustrates features of picam_advanced.h.</td>
</tr>
</tbody>
</table>
Table 2-1: List of Sample Code Files Provided (Sheet 2 of 2)

<table>
<thead>
<tr>
<th>Sample Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure</td>
<td>This sample illustrates how to change settings during camera setup as well as online while polling for data.</td>
</tr>
<tr>
<td>EMGainCalibration</td>
<td>This sample illustrates how to set up EM Gain Calibration. For additional information about incorporating this sample into production code, refer to Appendix B, EM Gain Calibration Code Sample.</td>
</tr>
<tr>
<td>Gating</td>
<td>This sample illustrates how to set up repetitive and sequential gating. Also demonstrates RF features on cameras which support RF functionality.</td>
</tr>
<tr>
<td>Kinetics</td>
<td>This sample provides a sequence of API calls used to request acquisition of image data using the kinetics window capture mode. The demo also illustrates how to make calls to utilize external triggering of captures. The captured pixel data are stored to a raw data file.</td>
</tr>
<tr>
<td>Metadata</td>
<td>This sample enables metadata (i.e., Time Stamp(s) and Frame Tracking.) It illustrates how to extract metadata from the data stream.</td>
</tr>
<tr>
<td>MultiCam</td>
<td>This example opens multiple (i.e., 2,) cameras and collects data from all simultaneously.</td>
</tr>
<tr>
<td>ParamInfo</td>
<td>This sample accesses all parameter information for all hardware parameters, and then prints them to the screen.</td>
</tr>
<tr>
<td>Poll</td>
<td>This sample illustrates how to use the polling method for collecting data by using Picam_WaitForAcquisitionUpdate().</td>
</tr>
<tr>
<td>Rois</td>
<td>This sample demonstrates the API for setting a simple single region of interest. It also shows how to set up a camera for multiple regions of interest and then acquires data for the given region(s).</td>
</tr>
<tr>
<td>SaveData</td>
<td>This sample acquires data synchronously and writes the returned data buffer to disk.</td>
</tr>
<tr>
<td>Spectrograph</td>
<td>This sample moves the center wavelength asynchronously and waits for it to complete.</td>
</tr>
<tr>
<td>WaitForTrig</td>
<td>This sample waits for an external trigger to start data acquisition.</td>
</tr>
</tbody>
</table>
2.5 Naming Conventions

The following naming conventions are used in PICam:

- All primitive types have a typedef with a \texttt{pi} prefix (e.g., \texttt{piint}, \texttt{pi64s}).
- All functions defined by PICam are prefixed with \texttt{Picam} and return an error code of \texttt{PicamError} (e.g., \texttt{Picam_GetParameterIntegerValue}, \texttt{Picam_CloseCamera}).
- All functions that allocate memory to store the results of a function call return a pointer to a constant allocation of the appropriate type. For example:
  - \texttt{Picam_GetEnumerationString} returns a string by taking the address of a pointer to a constant string. In other words an argument to the function is \texttt{const pichar**}.
  - \texttt{Picam_GetParameterCollectionConstraint} returns a collection constraint by taking the address of a pointer to a constant collection constraint. In other words, an argument to the function is \texttt{const PicamCollectionConstraint**}.
- All functions that allocate an array of memory to store the results of the function call return a pointer to a constant array allocation of the appropriate type as well as the number of items in the array. For example, \texttt{Picam_GetParameters} returns an array of parameters by taking the addresses of a pointer to a constant parameter array and a count. In other words, two arguments to the function are \texttt{const PicamParameter**} and \texttt{piint*}.
- All functions that free memory allocated by PICam have a \texttt{Picam_Destroy} prefix (e.g., \texttt{Picam_DestroyString}, \texttt{PicamDestroy_CollectionConstraints}, \texttt{PicamDestroyRois}).
- All types defined by PICam are prefixed with \texttt{Picam} and have a typedef to \texttt{<TypeName>} (e.g., \texttt{PicamParameter}, \texttt{PicamRoi}).
- All enum type members defined by PICam are prefixed with \texttt{<EnumName>\_} (e.g., \texttt{PicamValueType} enum has a \texttt{PicamValueType\_Integer} constant).
- All enum types that represent multiple values with bit masks have a \texttt{Mask} suffix (e.g., \texttt{PicamCcdCharacteristicsMask}, \texttt{PicamTimeStampsMask}).

2.6 Concepts

\textbf{Figure 2-1} is a high-level block diagram of the basic PICam structure. Hardware that is powered on and plugged into the host computer is initially represented by hardware IDs. The content of the hardware ID will be unique for each piece of hardware. From the hardware ID, basic information can be garnered such as availability and basic information. It is also from a hardware ID that hardware can be opened. Once opened, the hardware can be configured by adjusting the values of its parameters. The permitted values a parameter can take are defined by its constraints. Different hardware items not only possess different parameters, but different rules for interacting with those parameters. This information for each parameter may also be queried. Once a piece of hardware has been configured, data can be acquired from it.
* Value and Constraint Types are parameter dependent
2.6.1 Handles

Most PICam APIs require handles to identify the specific hardware with which they are currently interacting. When hardware is brought online, it is assigned a specific handle that is then used to identify it throughout the active session.

The following handle(s) may be passed as an API parameter:

- **accessory**
  Identifies a specific non-camera accessory within the system.

- **camera**
  Identifies a specific camera within the system.
  When `camera` is passed to an API, PICam determines the appropriate actions depending on the API that has been called.
  This handle is passed as a Basic API parameter.

- **camera_or_accessory**
  Identifies hardware within the system that can be either a camera or an accessory.

- **device**
  Identifies a specific PHYSICAL camera within the system.
  When `device` is passed to an API, any resulting interaction or configuration performed by the API is done on a physical camera that is attached to the system.
  This handle is passed as an Advanced API parameter, and must be used in conjunction with `model`.

- **model**
  Identifies a specific VIRTUAL camera within application memory.
  When `model` is passed to an API, any parameter configuration changes are temporarily stored in system memory (i.e., within the host computer.) The actual camera configuration remains unchanged until an API is called that commits values to the `device` (i.e., the physical camera.)
  This handle is passed as an Advanced API parameter, and must be used in conjunction with `device`.

Figure 2-2 illustrates the hierarchical relationship between PICam camera-specific handles and with which set of APIs they are valid.

**Figure 2-2: Block Diagram of Camera-Specific Handle Hierarchy**
2.7 Defined Data Types

The typedefs are given in the header file `pil_platform.h`.

**Table 2-2: Data Type Definitions**

<table>
<thead>
<tr>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>piint</td>
<td>Integer native to platform</td>
</tr>
<tr>
<td>piflt</td>
<td>Floating point native to platform</td>
</tr>
<tr>
<td>piBln</td>
<td>Boolean native to platform</td>
</tr>
<tr>
<td>piChar</td>
<td>Character native to platform</td>
</tr>
<tr>
<td>piByte</td>
<td>Byte native to platform</td>
</tr>
<tr>
<td>piBool</td>
<td>C++ Boolean native to platform</td>
</tr>
</tbody>
</table>

**Table 2-3: Sized Data Type Definitions**

<table>
<thead>
<tr>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>pi8s</td>
<td>8-bit signed integer</td>
</tr>
<tr>
<td>pi8u</td>
<td>8-bit unsigned integer</td>
</tr>
<tr>
<td>pi16s</td>
<td>16-bit signed integer</td>
</tr>
<tr>
<td>pi16u</td>
<td>16-bit unsigned integer</td>
</tr>
<tr>
<td>pi32s</td>
<td>32-bit signed integer</td>
</tr>
<tr>
<td>pi32u</td>
<td>32-bit unsigned integer</td>
</tr>
<tr>
<td>pi64s</td>
<td>64-bit signed integer</td>
</tr>
<tr>
<td>pi64u</td>
<td>64-bit unsigned integer</td>
</tr>
<tr>
<td>pi32f</td>
<td>32-bit floating point</td>
</tr>
<tr>
<td>pi64f</td>
<td>64-bit floating point</td>
</tr>
</tbody>
</table>
2.8 Include Files

Any program using PICam must include the following header files:

- `pil_platform.h`
  Princeton Instruments’ library platform support. This is included indirectly via `picam.h`.
- `picam.h`
  Princeton Instruments’ camera control Application Programming Interface (API.)

2.8.1 Optional and Advanced Files

The following files are optional and only need to be included when one or more of the functions found within them are required:

- `picam_special.h`
  Only include `picam_special.h` when using a special function defined in that file.
- `picam_advanced.h`
  This is the Princeton Instruments advanced camera control API. This header file contains advanced functionality such as camera discovery, change notification, circular buffering, user state, defect map, and data acquisition callbacks.
- `picam_accessory.h`
  This header contains functionality exclusively for accessory control.
- `picam_em_calibration.h`
  This header EM Gain Calibration file provides the APIs and functionality needed to perform EM gain calibration for a ProEM camera.
Chapter 3: General Library APIs

The first section of picam.h includes functions to:
- Determine if the PICam library has been initialized;
- Initialize the library;
- Uninitialize the library;
- Retrieve the version.

This section also includes error codes that may be returned from any PICam function.

The first step in using the PICam library is library initialization. This is typically done at the start of the program. Once the library has been initialized, PICam function can then be called. The success of every function call is determined by the error code that is returned. It is paramount this error code be checked as most results are invalidated if a function fails. To facilitate debugging, PICam can convert an error code into a string. In fact, any PICam enum can be converted into a string. Once the program is finished with the library, it should clean up and uninitialized the library. This often occurs during program shutdown.

3.1 Data Type Definitions

Refer to Table 3-1 for information about data definitions.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamError</td>
<td>enum</td>
<td>The set of error codes returned from all APIs declared as PICAM_API.</td>
</tr>
<tr>
<td>PicamEnumeratedType</td>
<td>enum</td>
<td>The set of all PICam enumeration types.</td>
</tr>
</tbody>
</table>
3.2 Programmers' Reference for General Use Library APIs

This section provides a detailed programmers' reference guide for the following APIs, including their syntax and behavior:

- Library Version
  - Picam_GetVersion()

- Library Initialization
  - Picam_IsLibraryInitialized()
  - Picam_InitializeLibrary()
  - Picam_UninitializeLibrary()

- General String Handling
  - Picam_DestroyString()
  - Picam_GetEnumerationString()

3.2.1 Picam_GetVersion()

Description
Picam_GetVersion() returns PICam version information.

The following version information may be requested:

- Major
  This is the Major release version which is incremented with each major feature addition or breaks backward-compatibility.

- Minor
  This is the Minor release version which is incremented with minor feature additions.

- Distribution
  This is the Distribution version which is incremented with bug fix releases.

- Released
  This is the date of the current official release in the format YYMM. When a release is classified as a Beta release, requesting this information returns a zero (0).

**NOTE:**

Picam_GetVersion() may be called prior to initializing the library with Picam_InitializeLibrary().

Syntax
The syntax of Picam_GetVersion() is:

```c
PICAM_API Picam_GetVersion(
    piint* major,
    piint* minor,
    piint* distribution,
    piint* released);
```

continued on next page
Input Parameters
Input parameters for Picam_GetVersion() are:

- **major**: Used to request Major version.
  - Valid values are:
    - `&major`
      - Indicates that the Major version is to be returned.
    - `0/null`
      - Indicates that the Major version is not to be returned.

- **minor**: Used to request Minor version.
  - Valid values are:
    - `&minor`
      - Indicates that the Minor version is to be returned.
    - `0/null`
      - Indicates that the Minor version is not to be returned.

- **distribution**: Used to request Distribution version.
  - Valid values are:
    - `&distribution`
      - Indicates that the Distribution version is to be returned.
    - `0/null`
      - Indicates that the Distribution version is not to be returned.

- **released**: Used to request official Release date.
  - Valid values are:
    - `&released`
      - Indicates that the Release date is to be returned.
    - `0/null`
      - Indicates that the Release date is not to be returned.

Output Parameters
Output Parameters for Picam_GetVersion() are:

- **major**: Returns the Major version.
- **minor**: Returns the Minor version.
- **distribution**: Returns the Distribution version.
- **released**: Returns the Released version.

Examples
If the PICam version is **4.2.1.1006**, it indicates the following version information:

- Major version: 4
- Minor version: 2
- Distribution version: 1
- Release Date: **1006** [i.e., June, 2010.]

Similarly, if the PICam version is **5.1.2.0**, it indicates the following version information:

- Major version: 5
- Minor version: 1
- Distribution version: 2
- Release Date: 0 indicating a Beta release.
3.2.2 Picam_IsLibraryInitialized()

Description
Picam_IsLibraryInitialized() determines if the library has been initialized.

**NOTE:**

Picam_IsLibraryInitialized() may be called prior to initializing the library using Picam_InitializeLibrary().

Syntax
The syntax of Picam_IsLibraryInitialized() is:

```
PICAM_API Picam_IsLibraryInitialized (pibln* inited);
```

Input Parameters
There are no input parameters associated with Picam_IsLibraryInitialized().

Output Parameters
Output parameters for Picam_IsLibraryInitialized() are:

- **inited**: Indicates the initialization status for the library. Valid values are:
  - **True**: Indicates that the library has been initialized.
  - **False**: Indicates that the library remains uninitialized.

Related APIs
For additional information, refer to the following related APIs:

- Picam_InitializeLibrary()
3.2.3 Picam_InitializeLibrary()

Description
Picam_InitializeLibrary() initializes the library and prepares it for use.

Syntax
The syntax of Picam_InitializeLibrary() is:

```
PICAM_API Picam_InitializeLibrary (void);
```

Usage
Unless specifically noted otherwise, Picam_InitializeLibrary() MUST be called prior to calling any additional Library API routine.

\[\text{NOTE:} \quad \text{Picam_UninitializeLibrary()} \text{ MUST be called prior to program termination.}\]

Input Parameters
There are no input parameters associated with Picam_InitializeLibrary().

Output Parameters
There are no output parameters associated with Picam_InitializeLibrary().

Related APIs
For additional information, refer to the following related APIs:

- Picam_UninitializeLibrary()
3.2.4 Picam_UninitializeLibrary()

Description
Picam_UninitializeLibrary() frees resources that have been used by the API Library, including open cameras and memory.

NOTE: Picam_UninitializeLibrary() MUST be called prior to program termination.

Syntax
The syntax of Picam_UninitializeLibrary() is:

```c
PICAM_API Picam_UninitializeLibrary (void);
```

Input Parameters
There are no input parameters associated with Picam_UninitializeLibrary().

Output Parameters
There are no output parameters associated with Picam_UninitializeLibrary().

Related APIs
For additional information, refer to the following related APIs:

- Picam_IsLibraryInitialized()
- Picam_InitializeLibrary()
3.2.5 Picam_DestroyString()

Description
Picam_DestroyString() releases PICam-allotted memory that has been associated with a specified character string, s.

NOTE: If the character string, s, is null, Picam_DestroyString() has no effect.

Syntax
The syntax of Picam_DestroyString() is:

```c
PICAM_API Picam_DestroyString(const pichar* s);
```

Input Parameters
Input parameters for Picam_DestroyString() are:

- s: Pointer to the character string for which memory is to be released.

Output Parameters
There are no output parameters associated with Picam_DestroyString().
3.2.6 Picam_GetEnumerationString()

Description
Picam_GetEnumerationString() determines what enumeration strings have been defined for the specified enumerated type. Returns an allocated string representation of the enumeration type with value in s.

NOTE: Picam_DestroyString() must be called to free the allocated memory associated with string s.

Syntax
The syntax of Picam_GetEnumerationString() is:

```
PICAM_API Picam_GetEnumerationString(
    PicamEnumeratedType type,
    piint value,
    const pichar** s);
```

Input Parameters
Input parameters for Picam_GetEnumerationString() are:

- `type`: The type for which enumeration strings are being requested.
- `value`: The numeric value associated with enumeration string being requested.

Output Parameters
Output parameters for Picam_GetEnumerationString() are:

- `s`: Pointer to the enumeration string.

Related APIs
For additional information, refer to the following related APIs:

- Picam_DestroyString()
Chapter 4: Identification APIs

The APIs in this section of picam.h deal with determining what hardware is available or being used in another instance, retrieving information from firmware, opening and closing a hardware, and connecting/disconnecting a demo camera.

Once the library has been initialized, all hardware that is powered on and connected to the host computer will have a corresponding hardware ID. Accessing hardware is as simple as opening available hardware using its corresponding ID.

NOTE: It is recommended that the Advanced API be used for device discovery if it is necessary to detect newly connected hardware after the library has been initialized.

A demo camera is a software-simulated camera. This allows program development without a camera connected. A demo camera can be instantiated by choosing a particular camera model and connecting it. Once connected, it can be interacted with as any other camera.

Once hardware (possibly a demo camera) is no longer used, it should be closed.

The following factors affect hardware availability to the program:

- Connectivity
  In order for hardware to be detected by the program it must be:
  — Connected to the host computer;
  — The hardware must be powered on.

- Open Elsewhere
  Hardware can only be controlled by a single instance of a program. If hardware has already been opened by another program (i.e., it is open elsewhere,) it is unavailable and cannot be used until it is closed.

Basic information identifies the model, computer interface, and serial number of the hardware (as well as the sensor for cameras.)

Additional information contained in the hardware's firmware can be read if the specified hardware is connected and provides the logic program IDs and revision levels. This information may not be available for hardware that has been opened elsewhere (in another process).
### 4.1 Data Type Definitions

Refer to Table 4-1 for information about data type definitions for hardware APIs.

**Table 4-1: Data Type Definitions for Hardware APIs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamModel</td>
<td>enum</td>
<td>The hardware model. Series models represent a model family and may be used to represent older hardware whose exact model is not known.</td>
</tr>
<tr>
<td>PicamComputerInterface</td>
<td>enum</td>
<td>The interface used to communicate with the hardware.</td>
</tr>
<tr>
<td>PicamStringSize</td>
<td>enum</td>
<td>Fixed sizes limiting the maximum size of some picam strings.</td>
</tr>
<tr>
<td>PicamHandle</td>
<td>void*</td>
<td>A PICam allocated resource.</td>
</tr>
</tbody>
</table>
4.2 Structure Definitions

This section provides information about structures required by the hardware APIs.

4.2.1 PicamCameraID

Structure Definition
The structure definition for PicamCameraID is:

```c
typedef struct PicamCameraID {
    PicamModel model;
    PicamComputerInterface computer_interface;
    pichar sensor_name [ ];
    pichar serial_number [ ];
} PicamCameraID;
```

Variable Definitions
The variables required for PicamCameraID are:

- **model**: This is the camera model.
- **computer_interface**: This is the method by which the camera communicates with the host computer.
- **sensor_name**: This is the name of the sensor in the camera.
- **serial_number**: This is the unique serial number that corresponds with the camera.

4.2.2 PicamAccessoryID

Structure Definition
The structure definition for PicamAccessoryID is:

```c
typedef struct PicamAccessoryID {
    PicamModel model;
    PicamComputerInterface computer_interface;
    pichar serial_number [ ];
} PicamAccessoryID;
```

Variable Definitions
The variables required for PicamAccessoryID are:

- **model**: This is the accessory model.
- **computer_interface**: This is the method by which the accessory communicates with the host computer.
- **serial_number**: This is the unique serial number that corresponds with the accessory.
4.2.3 PicamFirmwareDetail

Structure Definition
The structure definition for PicamFirmwareDetail is:

```c
typedef struct PicamFirmwareDetail {
    pichar name [ ];
    pichar detail [ ];
} PicamFirmwareDetail;
```

Variable Definitions
The variables required for PicamFirmwareDetail are:

- **name**: This is the name of a hardware device containing firmware.
- **detail**: This stores information about the hardware device, such as version number.

4.2.4 PicamCalibrationPoint

Structure Definition
The structure definition for PicamCalibrationPoint is:

```c
typedef struct PicamCalibrationPoint {
    piflt x;
    piflt y;
} PicamCalibrationPoint;
```

Variable Definitions
The variables required for PicamCalibrationPoint are:

- **x**: This is the x-coordinate of the calibration point.
- **y**: This is the y-coordinate of the calibration point.

4.2.5 PicamCalibration

Structure Definition
The structure definition for PicamCalibration is:

```c
typedef struct PicamCalibration {
    const PicamCalibrationPoint*point_array;
    pint point_count;
} PicamCalibration;
```

Variable Definitions
The variables required for PicamCalibration are:

- **point_array**: This is an array of one or more calibration points.
- **point_count**: This is the number of calibration points.
4.3 Programmers’ API Reference

This section provides a detailed programmers’ reference guide for the following APIs:

- **Identification APIs**
  - Picam_DestroyCameraIDs()
  - PicamAccessory_DestroyAccessoryIDs()
  - Picam_GetAvailableCameraIDs()
  - PicamAccessory_GetAvailableAccessoryIDs()
  - Picam_GetUnavailableCameraIDs()
  - PicamAccessory_GetUnavailableAccessoryIDs()
  - Picam_IsCameraIDConnected()
  - PicamAccessory_IsAccessoryIDConnected()
  - Picam_IsCameraIDOpenElsewhere()
  - PicamAccessory_IsAccessoryIDOpenElsewhere()

- **Access APIs**
  - Picam_DestroyHandles()
  - Picam_OpenFirstCamera()
  - PicamAccessory_OpenFirstAccessory()
  - Picam_OpenCamera()
  - PicamAccessory_OpenAccessory()
  - Picam_CloseCamera()
  - PicamAccessory_CloseAccessory()
  - Picam_GetOpenCameras()
  - PicamAccessory_GetOpenAccessories()
  - Picam_IsCameraConnected()
  - PicamAccessory_IsAccessoryConnected()
  - Picam_IsCameraFaulted()
  - Picam_GetCameraID()
  - PicamAccessory_GetAccessoryID()

- **Information APIs**
  - Picam_DestroyFirmwareDetails()
  - Picam_GetFirmwareDetails()
  - PicamAccessory_GetFirmwareDetails()
  - Picam_DestroyCalibrations()

- **Demo Camera Identification APIs**
  - Picam_DestroyModels()
  - Picam_GetAvailableDemoCameraModels()
  - Picam_ConnectDemoCamera()
  - Picam_DisconnectDemoCamera()
  - Picam_IsDemoCamera()
4.3.1 Identification APIs

This section provides programming information for camera and accessory Identification APIs.

4.3.1.1 Picam_DestroyCameraIDs()

Description
Picam_DestroyCameraIDs() releases PICam-alloted memory associated with id_array.

NOTE: id_array may be a single PicamCameraID allocated by PICam.

If id_array is a null array, calling Picam_DestroyCameraIDs() has no effect.

Syntax
The syntax for Picam_DestroyCameraIDs() is:

```
PICAM_API Picam_DestroyCameraIDs (const PicamCameraID* id_array);
```

Input Parameters
Input parameters for Picam_DestroyCameraIDs() are:

- id_array: Pointer to the id_array for which memory is to be released.

Output Parameters
There are no output parameters associated with Picam_DestroyCameraIDs().

Related APIs
For additional information, refer to the following related APIs:

- Picam_GetAvailableCameraIDs();
- Picam_GetUnavailableCameraIDs().
4.3.1.2 PicamAccessory_DestroyAccessoryIDs()

Description
PicamAccessory_DestroyAccessoryIDs() releases PICam-alloted memory associated with id_array.

NOTE: id_array may be a single PicamAccessoryID allocated by PICam.

If id_array is a null array, calling PicamAccessory_DestroyAccessoryIDs() has no effect.

Syntax
The syntax for PicamAccessory_DestroyAccessoryIDs() is:

PICAM_API PicamAccessory_DestroyAccessoryIDs (const PicamAccessoryID* id_array);

Input Parameters
Input parameters for PicamAccessory_DestroyAccessoryIDs() are:

id_array: Pointer to the id_array for which memory is to be released.

Output Parameters
There are no output parameters associated with PicamAccessory_DestroyAccessoryIDs().

Related APIs
For additional information, refer to the following related APIs:

- PicamAccessory_GetAvailableAccessoryIDs();
- PicamAccessory_GetUnavailableAccessoryIDs().
4.3.1.3 Picam_GetAvailableCameraIDs()

Description
Picam_GetAvailableCameraIDs() dynamically creates an array of length N. This array stores camera IDs for all available cameras.

**NOTE:**
Cameras that have been disconnected or are currently open in another process are not available.

**NOTE:**
Prior to program termination, memory that has been dynamically allocated to id_array must be released by calling Picam_DestroyCameraIDs().

Syntax
The syntax for Picam_GetAvailableCameraIDs() is:

```c
PICAM_API Picam_GetAvailableCameraIDs(
    const PicamCameraID** id_array,
    pint* id_count);
```

Input Parameters
There are no input parameters associated with Picam_GetAvailableCameraIDs().

Output Parameters
Output parameters for Picam_GetAvailableCameraIDs() are:

- **id_array**: Pointer to the memory address for the array in which the list of available camera IDs is stored. When there are no available camera IDs, a null value is returned.
- **id_count**: The total number of available camera IDs stored in id_array. This equals the length of the array that has been created. When there are no available camera IDs, a value of 0 [zero] is returned.

Related APIs
For additional information, refer to the following related APIs:

- Picam_DestroyCameraIDs().
PicamAccessory_GetAvailableAccessoryIDs()

Description
PicamAccessory_GetAvailableAccessoryIDs() dynamically creates an array of length N. This array stores accessory IDs for all available accessories.

**NOTE:**
Accessories that have been disconnected or are currently open in another process are not available.

**NOTE:**
Prior to program termination, memory that has been dynamically allocated to id_array must be released by calling PicamAccessory_DestroyAccessoryIDs().

Syntax
The syntax for PicamAccessory_GetAvailableAccessoryIDs() is:

```c
PICAM_API PicamAccessory_GetAvailableAccessoryIDs(
    const PicamAccessoryID** id_array,
    piint* id_count);
```

Input Parameters
There are no input parameters associated with PicamAccessory_GetAvailableAccessoryIDs().

Output Parameters
Output parameters for PicamAccessory_GetAvailableAccessoryIDs() are:

- **id_array**: Pointer to the memory address for the array in which the list of available accessory IDs is stored. When there are no available accessory IDs, a null value is returned.
- **id_count**: The total number of available accessory IDs stored in id_array. This equals the length of the array that has been created. When there are no available accessory IDs, a value of 0 [zero] is returned.

Related APIs
For additional information, refer to the following related APIs:

- PicamAccessory_DestroyAccessoryIDs().
**4.3.1.5 Picam_GetUnavailableCameraIDs()**

**Description**

`Picam_GetUnavailableCameraIDs()` dynamically creates an array of length N. This array stores camera IDs for all unavailable cameras.

**NOTE:**

Cameras that have been disconnected or are currently open in another process are not available.

**NOTE:**

Prior to program termination, memory that has been dynamically allocated to `id_array` must be released by calling `Picam_DestroyCameraIDs()`.

**Syntax**

The syntax for `Picam_GetUnavailableCameraIDs()` is:

```c
PICAM_API Picam_GetUnavailableCameraIDs(
    const PicamCameraID** id_array,
    piint* id_count);
```

**Input Parameters**

There are no input parameters associated with `Picam_GetUnavailableCameraIDs()`.

**Output Parameters**

Output parameters for `Picam_GetUnavailableCameraIDs()` are:

- `id_array`: Pointer to the memory address for the array in which the list of unavailable camera IDs is stored. When there are no unavailable camera IDs, a null value is returned.
- `id_count`: The total number of unavailable camera IDs stored in `id_array`. This equals the length of the array that has been created. When there are no unavailable camera IDs, a value of 0 [zero] is returned.

**Related APIs**

For additional information, refer to the following related APIs:

- `Picam_DestroyCameraIDs()`.
4.3.1.6  PicamAccessory_GetUnavailableAccessoryIDs()

Description
PicamAccessory_GetUnavailableAccessoryIDs() dynamically creates an array of length N. This array stores accessory IDs for all unavailable accessories.

**NOTE:**
Accessories that have been disconnected or are currently open in another process are not available.

**NOTE:**
Prior to program termination, memory that has been dynamically allocated to id_array must be released by calling PicamAccessory_DestroyAccessoryIDs().

Syntax
The syntax for PicamAccessory_GetUnavailableAccessoryIDs() is:

```c
PICAM_API PicamAccessory_GetUnavailableAccessoryIDs(
    const PicamAccessoryID** id_array,
    pint* id_count);
```

Input Parameters
There are no input parameters associated with PicamAccessory_GetUnavailableAccessoryIDs().

Output Parameters
Output parameters for PicamAccessory_GetUnavailableAccessoryIDs() are:

- **id_array**: Pointer to the memory address for the array in which the list of unavailable accessory IDs is stored. When there are no unavailable accessory IDs, a null value is returned.
- **id_count**: The total number of unavailable accessory IDs stored in id_array. This equals the length of the array that has been created. When there are no unavailable accessory IDs, a value of 0 [zero] is returned.

Related APIs
For additional information, refer to the following related APIs:

- PicamAccessory_DestroyAccessoryIDs().
4.3.1.7 Picam_IsCameraIDConnected()

Description
Picam_IsCameraIDConnected() determines if a specified camera ID is plugged into
the host computer and turned on.

Syntax
The syntax for Picam_IsCameraIDConnected() is:

```
PICAM_API Picam_IsCameraIDConnected(
    const PicamCameraID* id,
    pibln* connected);
```

Input Parameters
Input parameters for Picam_IsCameraIDConnected() are:

- `id`: Specifies the ID of the camera for which the connection status is
  being tested.

Output Parameters
Output parameters for Picam_IsCameraIDConnected() are:

- `connected`: Returns the connection status for the specified camera ID.
  Valid values are:
  - `True`
    Indicates that the specified camera ID is connected to the
    host computer and is turned on;
  - `False`
    Indicates that the specified camera ID is not connected to the
    host computer or is not turned on.
4.3.1.8 PicamAccessory_IsAccessoryIDConnected()

Description
PicamAccessory_IsAccessoryIDConnected() determines if a specified accessory ID is plugged into the host computer and turned on.

Syntax
The syntax for PicamAccessory_IsAccessoryIDConnected() is:

```c
PICAM_API PicamAccessory_IsAccessoryIDConnected(
    const PicamAccessoryID* id,
    pibln* connected);
```

Input Parameters
Input parameters for PicamAccessory_IsAccessoryIDConnected() are:

id: Specifies the ID of the accessory for which the connection status is being tested.

Output Parameters
Output parameters for PicamAccessory_IsAccessoryIDConnected() are:

connected: Returns the connection status for the specified accessory ID. Valid values are:

- True
  Indicates that the specified accessory ID is connected to the host computer and is turned on;
- False
  Indicates that the specified accessory ID is not connected to the host computer or is not turned on.
4.3.1.9 Picam_IsCameraIDOpenElsewhere()

Description

Picam_IsCameraIDOpenElsewhere() determines if a specified camera ID has been opened by another process.

Syntax

The syntax for Picam_IsCameraIDOpenElsewhere() is:

```c
PICAM_API Picam_IsCameraIDOpenElsewhere(
    const PicamCameraID* id,
    pibln* open_elsewhere);
```

Input Parameters

Input parameters for Picam_IsCameraIDOpenElsewhere() are:

- id: Specifies the ID of the camera for which the connection status is being tested.

Output Parameters

Output parameters for Picam_IsCameraIDOpenElsewhere() are:

- open_elsewhere: Returns the connection status for the specified camera ID. Valid values are:
  - True
    Indicates that the specified camera ID is currently open in another process;
  - False
    Indicates that the specified camera ID is not currently open in another process.
**4.3.1.10 PicamAccessory_IsAccessoryIDOpenElsewhere()**

**Description**

PicamAccessory_IsAccessoryIDOpenElsewhere() determines if a specified accessory ID has been opened by another process.

**Syntax**

The syntax for PicamAccessory_IsAccessoryIDOpenElsewhere() is:

```c
PICAM_API PicamAccessory_IsAccessoryIDOpenElsewhere(
    const PicamAccessoryID* id,
    pibln* open_elsewhere);
```

**Input Parameters**

Input parameters for PicamAccessory_IsAccessoryIDOpenElsewhere() are:

- **id**: Specifies the ID of the accessory for which the connection status is being tested.

**Output Parameters**

Output parameters for PicamAccessory_IsAccessoryIDOpenElsewhere() are:

- **open_elsewhere**: Returns the connection status for the specified accessory ID. Valid values are:
  - **True**: Indicates that the specified accessory ID is currently open in another process;
  - **False**: Indicates that the specified accessory ID is not currently open in another process.
4.3.2 Access APIs

This section provides programming information for camera and accessory Access APIs.

4.3.2.1 Picam_DestroyHandles()

Description

Picam_DestroyHandles() releases memory that has been allocated by PICam for use by handle_array.

NOTE: handle_array may be a single PicamHandle allocated by PICam.

If handle_array is a null array, calling Picam_DestroyHandles() has no effect.

NOTE: Picam_DestroyHandles() releases the memory used to store the handles. It does NOT free the resources to which the handles refer.

Syntax

The syntax for Picam_DestroyHandles() is:

PICAM_API Picam_DestroyHandles(
    const PicamHandle* handle_array);

Input Parameters

Input parameters for Picam_DestroyHandles() are:

    handle_array: Pointer to array memory that is to be released.

Output Parameters

There are no output parameters associated with Picam_DestroyHandles().
4.3.2.2  *Picam_OpenFirstCamera()*

**Description**

*Picam_OpenFirstCamera()* opens the first available camera, and returns a handle to the camera.

---

**NOTE:**

Prior to program termination, all open cameras must be closed by calling *Picam_CloseCamera()*.

---

**Syntax**

The syntax for *Picam_OpenFirstCamera()* is:

```c
PICAM_API Picam_OpenFirstCamera(
    PicamHandle* camera);
```

**Input Parameters**

There are no input parameters associated with *Picam_OpenFirstCamera()*.

**Output Parameters**

Output parameters for *Picam_OpenFirstCamera()* are:

- `camera`: The handle corresponding to the camera that has been opened.

**Advanced API Usage**

When used in conjunction with Advanced APIs, the handle returned is for the model.

**Related APIs**

For additional information, refer to the following related APIs:

- *Picam_CloseCamera()*.
4.3.2.3  PicamAccessory_OpenFirstAccessory()

Description
PicamAccessory_OpenFirstAccessory() opens the first available accessory, and returns a handle to the accessory.

NOTE: Prior to program termination, all open accessories must be closed by calling PicamAccessory_CloseAccessory().

Syntax
The syntax for PicamAccessory_OpenFirstAccessory() is:

```
PICAM_API PicamAccessory_OpenFirstAccessory(
    PicamHandle* accessory);
```

Input Parameters
There are no input parameters associated with PicamAccessory_OpenFirstAccessory().

Output Parameters
Output parameters for PicamAccessory_OpenFirstAccessory() are:

```
accessory: The handle corresponding to the accessory that has been opened.
```

Related APIs
For additional information, refer to the following related APIs:
- PicamAccessory_CloseAccessory().
4.3.2.4 Picam_OpenCamera()

Description
Picam_OpenCamera() opens a specified camera, and returns a handle to the camera.

**NOTE:**
Prior to program termination, all open cameras must be closed by calling Picam_CloseCamera().

Syntax
The syntax for Picam_OpenCamera() is:

```
PICAM_API Picam_OpenCamera(
    const PicamCameraID* id,
    PicamHandle* camera);
```

Input Parameters
Input parameters for Picam_OpenCamera() are:

- `id`: The id for camera to be opened.

Output Parameters
Output parameters for Picam_OpenCamera() are:

- `camera`: The handle corresponding to the open camera.

Advanced API Usage
When used in conjunction with Advanced APIs, the handle returned is for the camera model.

Related APIs
For additional information, refer to the following related APIs:

- Picam_CloseCamera().
4.3.2.5 PicamAccessory_OpenAccessory()

Description
PicamAccessory_OpenAccessory() opens a specified accessory, and returns a
handle to the accessory.

NOTE: Prior to program termination, all open accessories must be
closed by calling PicamAccessory_CloseAccessory().

Syntax
The syntax for PicamAccessory_OpenAccessory() is:

```c
PICAM_API PicamAccessory_OpenAccessory(
    const PicamAccessoryID* id,
    PicamHandle* accessory);
```

Input Parameters
Input parameters for PicamAccessory_OpenAccessory() are:

- id: The id for accessory to be opened.

Output Parameters
Output parameters for PicamAccessory_OpenAccessory() are:

- accessory: The handle corresponding to the open accessory.

Related APIs
For additional information, refer to the following related APIs:

- PicamAccessory_CloseAccessory().
4.3.2.6 Picam_CloseCamera()

Description
Picam_CloseCamera() releases all resources that have been associated with a specified camera.

Syntax
The syntax for Picam_CloseCamera() is:

```c
PICAM_API Picam_CloseCamera(
    PicamHandle camera);
```

Input Parameters
Input parameters for Picam_CloseCamera() are:

- camera: The handle associated with the camera that is to be closed.

Output Parameters
There are no output parameters associated with Picam_CloseCamera().

Advanced API Usage
When used in conjunction with Advanced APIs, camera can be a handle to either the:

- device, or
- model.

In either case, when Picam_CloseCamera() is called, it always closes both the specified device and model.
4.3.2.7 PicamAccessory_CloseAccessory()

Description
PicamAccessory_CloseAccessory() releases all resources that have been associated with a specified accessory.

Syntax
The syntax for PicamAccessory_CloseAccessory() is:

```
PICAM_API PicamAccessory_CloseAccessory(
    PicamHandle accessory);
```

Input Parameters
Input parameters for PicamAccessory_CloseAccessory() are:

accessory: The handle associated with the accessory that is to be closed.

Output Parameters
There are no output parameters associated with PicamAccessory_CloseAccessory().
4.3.2.8 Picam_GetOpenCameras()

Description
Picam_GetOpenCameras() dynamically creates an array of length N. This array stores camera handles for all open cameras in the current process.

**NOTE:** Prior to program termination, memory that has been dynamically allocated to camera_array must be released by calling Picam_DestroyHandles().

Syntax
The syntax for Picam_GetOpenCameras() is:

```
PICAM_API Picam_GetOpenCameras(
    const PicamHandle** camera_array,
    pint* camera_count);
```

Input Parameters
There are no input parameters associated with Picam_GetOpenCameras().

Output Parameters
Output parameters for Picam_GetOpenCameras() are:

- `camera_array`: Pointer to the memory address for the array in which the list of camera handles is stored. When there are no available camera handles, a null value is returned.
- `camera_count`: The total number of camera handles stored in camera_array. This equals the length of the array that has been created. When there are no available camera handles, a value of 0 [zero] is returned.

Advanced API Usage
When used in conjunction with Advanced APIs, this array (camera_array) stores a list of model handles.

Related APIs
For additional information, refer to the following related APIs:

- Picam_DestroyHandles().
4.3.2.9 PicamAccessory_GetOpenAccessories()

Description

PicamAccessory_GetOpenAccessories() dynamically creates an array of length N. This array stores accessory handles for all open accessories in the current process.

NOTE: Prior to program termination, memory that has been dynamically allocated to accessory_array must be released by calling Picam_DestroyHandles().

Syntax

The syntax for PicamAccessory_GetOpenAccessories() is:

```
PICAM_API PicamAccessory_GetOpenAccessories(
    const PicamHandle** accessory_array,
    plint* accessory_count);
```

Input Parameters

There are no input parameters associated with PicamAccessory_GetOpenAccessories().

Output Parameters

Output parameters for PicamAccessory_GetOpenAccessories() are:

- accessory_array: Pointer to the memory address for the array in which the list of accessory handles is stored. When there are no available accessory handles, a null value is returned.
- accessory_count: The total number of accessory handles stored in accessory_array. This equals the length of the array that has been created. When there are no available accessory handles, a value of 0 [zero] is returned.

Related APIs

For additional information, refer to the following related APIs:

- Picam_DestroyHandles().
4.3.2.10  Picam_IsCameraConnected()

Description
Picam_IsCameraConnected() determines if the specified camera is plugged into the host computer and is turned on.

Syntax
The syntax for Picam_IsCameraConnected() is:

```
PICAM_API Picam_IsCameraConnected(
    PicamHandle camera,
    pibln* connected);
```

Input Parameters
Input parameters for Picam_IsCameraConnected() are:

- camera: The handle for the camera for which the status is being determined.

Output Parameters
Output parameters for Picam_IsCameraConnected() are:

- connected: Returns the connection status for the specified camera. Valid values are:
  - True
    Indicates that the specified camera is connected to the host computer and is turned on.
  - False
    Indicates that the specified camera is not connected to the host computer and/or not turned on.

Advanced API Usage
When used in conjunction with Advanced APIs, camera can be a handle to either the:
- device, or
- model.

Both device and model share the same connected state.
4.3.2.11 PicamAccessory_IsAccessoryConnected()

Description
PicamAccessory_IsAccessoryConnected() determines if the specified accessory is plugged into the host computer and is turned on.

Syntax
The syntax for PicamAccessory_IsAccessoryConnected() is:

```c
PICAM_API PicamAccessory_IsAccessoryConnected(
    PicamHandle accessory,
    pibln* connected);
```

Input Parameters
Input parameters for PicamAccessory_IsAccessoryConnected() are:

- accessory: The handle for the accessory for which the status is being determined.

Output Parameters
Output parameters for PicamAccessory_IsAccessoryConnected() are:

- connected: Returns the connection status for the specified accessory. Valid values are:
  - True
    Indicates that the specified accessory is connected to the host computer and is turned on.
  - False
    Indicates that the specified accessory is not connected to the host computer and/or not turned on.
4.3.2.12 Picam_IsCameraFaulted()

Description
Picam_IsCameraFaulted() determines if the specified camera has experienced a critical malfunction and is in need of service. Any acquisition in progress will be stopped and further acquisitions are not possible until the camera has been serviced.

Syntax
The syntax for Picam_IsCameraFaulted() is:

```
PICAM_API Picam_IsCameraFaulted(
    PicamHandle camera,
    pibln* faulted);
```

Input Parameters
Input parameters for Picam_IsCameraFaulted() are:

- camera: The handle for the camera for which the status is being determined.

Output Parameters
Output parameters for Picam_IsCameraFaulted() are:

- faulted: Returns the faulted status for the specified camera.
  Valid values are:
  - True
    Indicates that the specified camera has experienced a critical malfunction.
  - False
    Indicates that the specified camera is working properly.

Advanced API Usage
When used in conjunction with Advanced APIs, camera can be a handle to either the:

- device, or
- model.

Both device and model share the same faulted state.
4.3.2.13 Picam_GetCameraID()

Description
Picam_GetCameraID() returns the ID associated with a specified camera handle.

Syntax
The syntax for Picam_GetCameraID() is:

```c
PICAM_API Picam_GetCameraID(
    PicamHandle camera,
    PicamCameraID* id);
```

Input Parameters
Input parameters for Picam_GetCameraID() are:

- `camera`: The handle associated with the camera for which the ID is to be determined.

Output Parameters
Output parameters for Picam_GetCameraID() are:

- `id`: The camera ID associated with the specified handle.

Advanced API Usage
When used in conjunction with Advanced APIs, `camera` can be a handle to either the:

- device, or
- model.

Both device and model share the same camera ID.
4.3.2.14 PicamAccessory_GetAccessoryID()

Description
PicamAccessory_GetAccessoryID() returns the ID associated with a specified accessory handle.

Syntax
The syntax for PicamAccessory_GetAccessoryID() is:

```c
PICAM_API PicamAccessory_GetAccessoryID(
    PicamHandle accessory,
    PicamAccessoryID* id);
```

Input Parameters
Input parameters for PicamAccessory_GetAccessoryID() are:

- accessory: The handle associated with the accessory for which the ID is to be determined.

Output Parameters
Output parameters for PicamAccessory_GetAccessoryID() are:

- id: The accessory ID associated with the specified handle.
4.3.3 Information APIs

This section provides programming information for camera and accessory Information APIs.

4.3.3.1 Picam_DestroyFirmwareDetails()

Description

Picam_DestroyFirmwareDetails() releases memory that has been allocated for use by the firmware_array.

NOTE: firmware_array may be a single PicamFirmwareDetail allocated by PICam.

If firmware_array is a null array, calling Picam_DestroyFirmwareDetails() has no effect.

Syntax

Picam_DestroyCalibrations()
The syntax for Picam_DestroyFirmwareDetails() is:

PICAM_API Picam_DestroyFirmwareDetails(
    const PicamFirmwareDetail* firmware_array);

Input Parameters

Input parameters for Picam_DestroyFirmwareDetails() are:

    firmware_array: Pointer to the memory location where the array is stored.

Output Parameters

There are no output parameters associated with Picam_DestroyFirmwareDetails().

Related APIs

For additional information, refer to the following related APIs:

    • Picam_GetFirmwareDetails()
4.3.3.2 Picam_GetFirmwareDetails()  

Description  
Picam_GetFirmwareDetails() dynamically creates an array of length N. This array stores firmware details associated with a specified camera ID.

**NOTE:** Prior to program termination, memory that has been dynamically allocated to firmware_array must be released by calling Picam_DestroyFirmwareDetails().

Syntax  
The syntax for Picam_GetFirmwareDetails() is:

```c
PICAM_API Picam_GetFirmwareDetails(
    const PicamCameraID* id,
    const PicamFirmwareDetail** firmware_array,
    piint* firmware_count);
```

Input Parameters  
Input parameters for Picam_GetFirmwareDetails() are:

- *id*: Camera id for which firmware details are to be retrieved.

Output Parameters  
Output parameters for Picam_GetFirmwareDetails() are:

- *firmware_array*: Pointer to the memory address for the array in which firmware information is stored.  
  When no information is stored, a null value is returned.

- *firmware_count*: The total number of firmware details stored in firmware_array.  
  This equals the length of the array that has been created.  
  When no information is available, a value of 0 [zero] is returned.

Related APIs  
For additional information, refer to the following related APIs:

- Picam_DestroyFirmwareDetails().
4.3.3.3  PicamAccessory_GetFirmwareDetails()

Description
PicamAccessory_GetFirmwareDetails() dynamically creates an array of length N. This array stores firmware details associated with a specified accessory ID.

**NOTE:** Prior to program termination, memory that has been dynamically allocated to firmware_array must be released by calling Picam_DestroyFirmwareDetails().

Syntax
The syntax for PicamAccessory_GetFirmwareDetails() is:

```c
PICAM_API PicamAccessory_GetFirmwareDetails(
    const PicamAccessoryID* id,
    const PicamFirmwareDetail** firmware_array,
    pint* firmware_count);
```

Input Parameters
Input parameters for PicamAccessory_GetFirmwareDetails() are:

`id`: Accessory id for which firmware details are to be retrieved.

Output Parameters
Output parameters for PicamAccessory_GetFirmwareDetails() are:

`firmware_array`: Pointer to the memory address for the array in which firmware information is stored. When no information is stored, a null value is returned.

`firmware_count`: The total number of firmware details stored in firmware_array. This equals the length of the array that has been created. When no information is available, a value of 0 [zero] is returned.

Related APIs
For additional information, refer to the following related APIs:

- Picam_DestroyFirmwareDetails().
4.3.3.4 Picam_DestroyCalibrations()

Description
Picam_DestroyCalibrations() releases memory that has been allocated for use by
the calibrations_array.

NOTE: calibrations_array may be a single
PicamCalibrationDetail allocated by PICam.

If calibrations_array is a null array, calling Picam_DestroyCalibrations() has no
effect.

Syntax
The syntax for Picam_DestroyCalibrations() is:

PICAM_API Picam_DestroyCalibrations(
    const PicamCalibration* calibrations_array);

Input Parameters
Input parameters for Picam_DestroyCalibrations() are:

    calibrations_array: Pointer to the memory location where the array is stored.

Output Parameters
There are no output parameters associated with Picam_DestroyCalibrations().
4.3.4 Demo Camera Identification APIs

This section provides programming information for Demo Camera Identification APIs.

4.3.4.1 Picam_DestroyModels()

Picam_DestroyModels() releases memory that has been allocated for use by the model_array.

**NOTE:**

model_array may be a single PicamModel allocated by PICam.

If model_array is a null array, calling Picam_DestroyModels() has no effect.

**Syntax**

The syntax for Picam_DestroyModels() is:

```
PICAM_API Picam_DestroyModels(
    const PicamModel* model_array);
```

**Input Parameters**

Input parameters for Picam_DestroyModels() are:

- model_array: Pointer to the memory location where the array is stored.

**Output Parameters**

There are no output parameters associated with Picam_DestroyModels().

**Related APIs**

For additional information, refer to the following related APIs:

- Picam_GetAvailableDemoCameraModels()
### 4.3.4.2 Picam_GetAvailableDemoCameraModels()

**Description**

`Picam_GetAvailableDemoCameraModels()` dynamically creates an array of length \( N \). This array stores a list of virtual camera models which are available for use in Demo Mode.

**NOTE:**

Prior to program termination, memory that has been dynamically allocated to `model_array` must be released by calling `Picam_DestroyModels()`.

**Syntax**

The syntax for `Picam_GetAvailableDemoCameraModels()` is:

```c
PICAM_API Picam_GetAvailableDemoCameraModels(
    const PicamModel** model_array,
    piint* model_count);
```

**Input Parameters**

There are no input parameters associated with `Picam_GetAvailableDemoCameraModels()`.

**Output Parameters**

Output parameters for `Picam_GetAvailableDemoCameraModels()` are:

- `model_array`: Pointer to the memory address for the array in which the list of virtual camera models is stored. When there are no virtual camera models available, a null value is returned.

- `model_count`: The total number of virtual models being stored in `model_array`. This equals the length of the array that has been created. When there are no virtual models available, a value of 0 [zero] is returned.

**Related APIs**

For additional information, refer to the following related APIs:

- `Picam_DestroyModels()`.
4.3.4.3 Picam_ConnectDemoCamera()

Description
Picam_ConnectDemoCamera() establishes a connection with the specified virtual camera.

Syntax
The syntax for Picam_ConnectDemoCamera() is:

```
PICAM_API Picam_ConnectDemoCamera(
    PicamModel  model,
    const pichar* serial_number,
    PicamCameraID* id);
```

Input Parameters
Input parameters for Picam_ConnectDemoCamera() are:

- model: Model for the virtual camera for which a connection is to be established.
- serial_number: Serial number of the virtual camera for which a connection is to be established.

Output Parameters
Output parameters for Picam_ConnectDemoCamera() are:

- id: ID of the virtual camera for which a connection is to be established.

**NOTE:**

id is an optional parameter and may be null.
4.3.4.4 *Picam_DisconnectDemoCamera()*

**Description**

*Picam_DisconnectDemoCamera()* breaks an established connection with the specified virtual camera.

**Syntax**

The syntax for *Picam_DisconnectDemoCamera()* is:

```
PICAM_API Picam_DisconnectDemoCamera(
    const PicamCameraID* id);
```

**Input Parameters**

Input parameters for *Picam_DisconnectDemoCamera()* are:

- *id*: ID of the virtual camera for which the connection is to be broken.

**Output Parameters**

There are no output parameters associated with *Picam_DisconnectDemoCamera()*.
4.3.4.5  Picam_IsDemoCamera()

Description
Picam_IsDemoCamera() determines if the specified camera is a virtual camera.

Syntax
The syntax for Picam_IsDemoCamera() is:

```c
PICAM_API Picam_IsDemoCamera(
   const PicamCameraID* id,
   pibln* demo);
```

Input Parameters
Input parameters for Picam_IsDemoCamera() are:

- id: ID of the camera being identified.

Output Parameters
Output parameters for Picam_IsDemoCamera() are:

- demo: Indicates if the specified camera is a software-simulated camera.
  Valid values are:
  - True
    Indicates that the specified camera is a virtual camera.
  - False
    Indicates that the specified camera is an actual physical camera.
Chapter 5: Configuration APIs

The functions in this grouping set or query parameter values, parameter information, and parameter constraints that characterize hardware. A parameter is a hardware setting. Parameters have varying qualities as well as values and constraints. A parameter may have several different values but constraints determine which kinds of values a parameter can have based on hardware type, read/write capability, or other parameters used or to be used in describing and setting up specific hardware. If a camera has been opened, it can be configured by changing its parameters through software and applying them to the hardware. If an accessory has been opened, it can be configured directly. Each different hardware model has a different set of parameters. Parameters contain different attributes.

The most important parameter attribute is its value. Values are represented by different types (i.e., integer, floating point, enumeration, etc.)

All parameter values can be read, but not all can be written. This is determined by the parameter’s value access:

- Read/Write
- Read Only.

NOTE: A special case of value access is when a parameter value can be written, but only one particular value is permitted. This is called read/write trivial.

Parameter values that can be written have constraints. Constraints describe the set of values a parameter value can take. The nature of this set determines the constraint type (e.g., a numeric range, a set of options, etc.) It is useful to describe a different constraint based on purpose. This is where constraint categories come into play. These categories differentiate "Is this parameter capable of \( x \)?" from "Based on the current configuration, is it valid to set parameter to \( x \)?"

Due to the complex nature of configuration, some parameters override others when certain values are set. A parameter is relevant if it has an effect on the current configuration.

For a camera, most parameters are only used for acquisition setup. However, this is not always the case. Some parameters can be modified while the hardware is acquiring. These parameters are deemed onlineable. Note that Accessories are not onlineable since they do not acquire data.

Still other parameters reflect the current state of the hardware. These parameters only have meaning when read directly from hardware and are termed readable.

Another parameter may reflect the status of hardware that is not directly controllable by the software (e.g., may be changed due to external influences,) yet it’s value may impact the decisions and/or further progress of the software. Such a parameter is a waitable status.

For a camera, once the parameter values have been adjusted as desired they must be committed to the hardware before the hardware can be used.
5.1 Data Type Definitions

This section provides programming information about PICam data type definitions.

5.1.1 Hardware Parameter Enumerations

This section provides information about the following hardware parameter enumerations:

NOTE: Enumerations are listed alphabetically.

- PicamActiveShutter
- PicamAdcAnalogGain
- PicamAdcQuality
- PicamCcdCharacteristicsMask
- PicamCenterWavelengthStatus
- PicamConstraintType
- PicamCoolingFanStatus
- PicamEMIccdgainControlMode
- PicamGateTrackingMask
- PicamGatingMode
- PicamGatingSpeed
- PicamGratingCoating
- PicamGratingType
- PicamIntensifierOptionsMask
- PicamIntensifierStatus
- PicamLaserOutputMode
- PicamLaserStatus
- PicamLightSource
- PicamLightSourceStatus
- PicamModulationTrackingMask
- PicamOrientationMask
- PicamOutputSignal
- PicamParameter
- PicamPhosphorType
- PicamPhotocathodeSensitivity
- PicamPhotonDetectionMode
- PicamPixelFormat
- PicamReadoutControlMode
- PicamSensorTemperatureStatus
- PicamSensorType
- PicamShutterStatus
- PicamShutterTimingMode
- PicamShutterType
- PicamTimeStampsMask
- PicamTriggerCoupling
- PicamTriggerDetermination
- PicamTriggerResponse
- PicamTriggerSource
- PicamTriggerStatus
- PicamTriggerTermination
- PicamVacuumStatus
- PicamValueType
5.1.1.1 PicamActiveShutter

Data Type
PicamActiveShutter is defined as enum.

Description
PicamActiveShutter is the shutter that will be controlled during an acquisition.

Enumerator Definitions
Refer to Table 5-2 for enumerator definitions.

Table 5-1: PicamActiveShutter Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamActiveShutter_Internal</td>
<td>The shutter internal to the hardware.</td>
</tr>
<tr>
<td>PicamActiveShutter_External</td>
<td>The shutter external to the hardware.</td>
</tr>
<tr>
<td>PicamActiveShutter_None</td>
<td>There is no shutter installed.</td>
</tr>
</tbody>
</table>

5.1.1.2 PicamAdcAnalogGain

Data Type
PicamAdcAnalogGain is defined as enum.

Description
PicamAdcAnalogGain is the set of electronic gain settings for pixel digitization.

Enumerator Definitions
Refer to Table 5-2 for enumerator definitions.

Table 5-2: PicamAdcAnalogGain Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamAdcAnalogGain_High</td>
<td>Large amplification. Refer to the user manual for the specific hardware being used for complete information.</td>
</tr>
<tr>
<td>PicamAdcAnalogGain_Low</td>
<td>Small amplification. Refer to the user manual for the specific hardware being used for complete information.</td>
</tr>
<tr>
<td>PicamAdcAnalogGain_Medium</td>
<td>Average amplification. Refer to the user manual for the specific hardware being used for complete information.</td>
</tr>
</tbody>
</table>
5.1.1.3 PicamAdcQuality

Data Type
PicamAdcQuality is defined as enum.

Description
PicamAdcQuality is the set of Analog-to-Digital conversion techniques and quality settings for pixel digitization.

Enumerator Definitions
Refer to Table 5-3 for enumerator definitions.

Table 5-3: PicamAdcQuality Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamAdcQuality_ElectronMultiplied</td>
<td>Provides electron multiplication.</td>
</tr>
<tr>
<td>PicamAdcQuality_HighCapacity</td>
<td>Optimized for sensing high levels of radiation.</td>
</tr>
<tr>
<td>PicamAdcQuality_HighSpeed</td>
<td>Provides faster readout speeds.</td>
</tr>
<tr>
<td>PicamAdcQuality_LowNoise</td>
<td>Optimized for the lowest noise.</td>
</tr>
</tbody>
</table>
### 5.1.1.4 PicamCcdCharacteristicsMask

**Data Type**

`PicamCcdCharacteristicsMask` is defined as `enum`.

**Description**

`PicamCcdCharacteristicsMask` is the set of CCD sensor characteristics.

**Enumerator Definitions**

Refer to Table 5-4 for enumerator definitions.

#### Table 5-4: `PicamCcdCharacteristicsMask` Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamCcdCharacteristicsMask_AdvancedInvertedMode</td>
<td>The CCD has reduced dark current.</td>
</tr>
<tr>
<td>PicamCcdCharacteristicsMask_BackIlluminated</td>
<td>Indicates the type of illumination used.</td>
</tr>
<tr>
<td></td>
<td>Valid values are:</td>
</tr>
<tr>
<td></td>
<td>• 1 CCD is back-illuminated</td>
</tr>
<tr>
<td></td>
<td>• 0 CCD is front-illuminated</td>
</tr>
<tr>
<td>PicamCcdCharacteristicsMask_DeepDepleted</td>
<td>The CCD is deep depleted.</td>
</tr>
<tr>
<td>PicamCcdCharacteristicsMask_ExcelonEnabled</td>
<td>The CCD is enhanced with eXcelon technology.</td>
</tr>
<tr>
<td>PicamCcdCharacteristicsMask_HighResistivity</td>
<td>The CCD is enhanced for sensing infrared radiation.</td>
</tr>
<tr>
<td>PicamCcdCharacteristicsMask_Multiport</td>
<td>The CCD has multiple readout ports that can be used simultaneously.</td>
</tr>
<tr>
<td>PicamCcdCharacteristicsMask_None</td>
<td>No additional characteristics.</td>
</tr>
<tr>
<td>PicamCcdCharacteristicsMask_OpenElectrode</td>
<td>The CCD is open electrode.</td>
</tr>
<tr>
<td>PicamCcdCharacteristicsMask_SecondaryMask</td>
<td>The CCD has an additional masked area.</td>
</tr>
<tr>
<td>PicamCcdCharacteristicsMask_UVEnhanced</td>
<td>The CCD is enhanced for sensing ultraviolet radiation.</td>
</tr>
</tbody>
</table>
5.1.1.5 PicamCenterWavelengthStatus

Data Type
PicamCenterWavelengthStatus is defined as enum.

Description
PicamCenterWavelengthStatus is the set of center wavelength statuses.

Enumerator Definitions
Refer to Table 5-5 for enumerator definitions.

Table 5-5: PicamCenterWavelengthStatus Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamCenterWavelengthStatus_Faulted</td>
<td>The grating drive has malfunctioned.</td>
</tr>
<tr>
<td>PicamCenterWavelengthStatus_Moving</td>
<td>The center wavelength is moving.</td>
</tr>
<tr>
<td>PicamCenterWavelengthStatus_Stationary</td>
<td>The center wavelength is stationary.</td>
</tr>
</tbody>
</table>

5.1.1.6 PicamConstraintType

Data Type
PicamConstraintType is defined as enum.

Description
PicamConstraintType is the set of constraints that may be placed on a parameter’s value.

Enumerator Definitions
Refer to Table 5-6 for enumerator definitions.

Table 5-6: PicamConstraintType Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamConstraintType_Collection</td>
<td>The value can be one in a collection of choices.</td>
</tr>
<tr>
<td>PicamConstraintType_Modulations</td>
<td>The value is a custom modulation sequence.</td>
</tr>
<tr>
<td>PicamConstraintType_None</td>
<td>The value is read-only and not constrained.</td>
</tr>
<tr>
<td>PicamConstraintType_Pulse</td>
<td>The value is a gate pulse.</td>
</tr>
<tr>
<td>PicamConstraintType_Range</td>
<td>The value is numeric and naturally constrained within a linear range.</td>
</tr>
<tr>
<td>PicamConstraintType_Rois</td>
<td>The value is a set of regions of interests.</td>
</tr>
</tbody>
</table>
5.1.1.7 PicamCoolingFanStatus

Data Type
PicamCoolingFanStatus is defined as enum.

Description
PicamCoolingFanStatus is the set of cooling fan statuses.

Enumerator Definitions
Refer to Table 5-7 for enumerator definitions.

Table 5-7: PicamCoolingFanStatus Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamCoolingFanStatus_ForcedOn</td>
<td>The cooling fan has been forced on to prevent overheating.</td>
</tr>
<tr>
<td>PicamCoolingFanStatus_Off</td>
<td>The cooling fan is off.</td>
</tr>
<tr>
<td>PicamCoolingFanStatus_On</td>
<td>The cooling fan is on.</td>
</tr>
</tbody>
</table>

5.1.1.8 PicamEMIccdGainControlMode

Data Type
PicamEMIccdGainControlMode is defined as enum.

Description
PicamEMIccdGainControlMode is the set of Control Modes which control intensifier gain and electron multiplication gain for emICCD hardware.

Enumerator Definitions
Refer to Table 5-8 for enumerator definitions.

Table 5-8: PicamEMIccdGainControlMode Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamEMIccdGainControlMode_Manual</td>
<td>Allows each gain to be controlled independently.</td>
</tr>
<tr>
<td>PicamEMIccdGainControlMode_Optimal</td>
<td>Controls both gains simultaneously as a single emICCD gain.</td>
</tr>
</tbody>
</table>
5.1.1.9 PicamGateTrackingMask

Data Type
PicamGateTrackingMask is defined as enum.

Description
PicamGateTrackingMask is the set of sequential gate pulse components that are to be tracked.

Enumerator Definitions
Refer to Table 5-9 for enumerator definitions.

Table 5-9: PicamGateTrackingMask Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamGateTrackingMask_Delay</td>
<td>The delay of the gate pulse is tracked.</td>
</tr>
<tr>
<td>PicamGateTrackingMask_None</td>
<td>No components are tracked.</td>
</tr>
<tr>
<td>PicamGateTrackingMask_Width</td>
<td>The width of the gate pulse is tracked.</td>
</tr>
</tbody>
</table>

5.1.1.10 PicamGatingMode

Data Type
PicamGatingMode is defined as enum.

Description
PicamGatingMode is the set of supported gate pulse timing modes.

Enumerator Definitions
Refer to Table 5-10 for enumerator definitions.

Table 5-10: PicamGatingMode Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamGatingMode_Custom</td>
<td>Custom gate timing.</td>
</tr>
<tr>
<td>PicamGatingMode_Disabled</td>
<td>Gating is disabled.</td>
</tr>
<tr>
<td>PicamGatingMode_Repetitive</td>
<td>The same gate timing is repeated for each frame.</td>
</tr>
<tr>
<td>PicamGatingMode_Sequential</td>
<td>Get timing varies for each frame.</td>
</tr>
</tbody>
</table>
5.1.11 PicamGatingSpeed

Data Type
PicamGatingSpeed is defined as enum.

Description
PicamGatingSpeed is the set of classifications of the narrowest gate pulse.

Enumerator Definitions
Refer to Table 5-11 for enumerator definitions.

Table 5-11: PicamGatingSpeed Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamGatingSpeed_Fast</td>
<td>The gate pulse can be very narrow.</td>
</tr>
<tr>
<td>PicamGatingSpeed_Slow</td>
<td>The gate pulse width is limited by the intensifier.</td>
</tr>
</tbody>
</table>

5.1.12 PicamGratingCoating

Data Type
PicamGratingCoating is defined as enum.

Description
PicamGratingCoating is the coating on the grating.

Enumerator Definitions
Refer to Table 5-12 for enumerator definitions.

Table 5-12: PicamGratingCoating Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamGratingCoating_Al</td>
<td>Aluminum coated.</td>
</tr>
<tr>
<td>PicamGratingCoating_AlMgF2</td>
<td>Aluminum and magnesium fluoride coated.</td>
</tr>
<tr>
<td>PicamGratingCoating_Ag</td>
<td>Silver coated.</td>
</tr>
<tr>
<td>PicamGratingCoating_Au</td>
<td>Gold coated.</td>
</tr>
</tbody>
</table>
5.1.1.13 PicamGratingType

Data Type
PicamGratingType is defined as enum.

Description
PicamGratingType is the type of grating.

Enumerator Definitions
Refer to Table 5-13 for enumerator definitions.

Table 5-13: PicamGratingType Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamGratingType_Ruled</td>
<td>Ruled grating.</td>
</tr>
<tr>
<td>PicamGratingType_HolographicVisible</td>
<td>Holographic grating for the visible range.</td>
</tr>
<tr>
<td>PicamGratingType_HolographicNir</td>
<td>Holographic grating for the near infrared range.</td>
</tr>
<tr>
<td>PicamGratingType_HolographicUV</td>
<td>Holographic grating for the ultraviolet range.</td>
</tr>
<tr>
<td>PicamGratingType_Mirror</td>
<td>Grating is a mirror.</td>
</tr>
</tbody>
</table>

5.1.1.14 PicamIntensifierOptionsMask

Data Type
PicamIntensifierOptionsMask is defined as enum.

Description
PicamIntensifierOptionsMask is the set of intensifier characteristics.

Enumerator Definitions
Refer to Table 5-14 for enumerator definitions.

Table 5-14: PicamIntensifierOptionsMask Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamIntensifierOptionsMask_Modulation</td>
<td>The intensifier can be modulated.</td>
</tr>
<tr>
<td>PicamIntensifierOptionsMask_SubNanosecondGating</td>
<td>The pulse can be gated narrower than a nanosecond.</td>
</tr>
<tr>
<td>PicamIntensifierOptionsMask_McpGating</td>
<td>The microchannel plate is gated instead of the photocathode.</td>
</tr>
<tr>
<td>PicamIntensifierOptionsMask_None</td>
<td>No additional options.</td>
</tr>
</tbody>
</table>
5.1.15 PicamIntensifierStatus

Data Type
PicamIntensifierStatus is defined as enum.

Description
PicamIntensifierStatus is the set of intensifier power statuses.

Enumerator Definitions
Refer to Table 5-15 for enumerator definitions.

Table 5-15: PicamIntensifierStatus Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamIntensifierStatus_PoweredOff</td>
<td>The physical switch is in the off position.</td>
</tr>
<tr>
<td>PicamIntensifierStatus_PoweredOn</td>
<td>The physical switch is in the on position.</td>
</tr>
</tbody>
</table>

5.1.16 PicamLaserOutputMode

Data Type
PicamLaserOutputMode is defined as enum.

Description
PicamLaserOutputMode is the set of laser output statuses.

Enumerator Definitions
Refer to Table 5-16 for enumerator definitions.

Table 5-16: PicamLaserOutputMode Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamLaserOutputMode_Disabled</td>
<td>The laser is off.</td>
</tr>
<tr>
<td>PicamLaserOutputMode_ContinuousWave</td>
<td>Output is constant.</td>
</tr>
<tr>
<td>PicamLaserOutputMode_Pulsed</td>
<td>output is gated on and off.</td>
</tr>
</tbody>
</table>

5.1.17 PicamLaserStatus

Data Type
PicamLaserStatus is defined as enum.

Description
PicamLaserStatus is the laser output status.

Enumerator Definitions
Refer to Table 5-17 for enumerator definitions.

Table 5-17: PicamLaserStatus Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamLaserStatus_Disarmed</td>
<td>The hardware key is missing.</td>
</tr>
<tr>
<td>PicamLaserStatus_Unarmed</td>
<td>The laser is off.</td>
</tr>
<tr>
<td>PicamLaserStatus_Arming</td>
<td>The laser will be enabled momentarily.</td>
</tr>
<tr>
<td>PicamLaserStatus_Armed</td>
<td>The laser is ready.</td>
</tr>
</tbody>
</table>
5.1.1.18 *PicamLightSource*

**Data Type**

*PicamLightSource* is defined as `enum`.

**Description**

*PicamLightSource* is the type of light source.

**Enumerator Definitions**

Refer to Table 5-18 for enumerator definitions.

**Table 5-18: *PicamLightSource* Enumerator Definitions**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamLightSource_Disabled</td>
<td>No Light Source</td>
</tr>
<tr>
<td>PicamLightSource_Hg</td>
<td>Mercury Light Source</td>
</tr>
<tr>
<td>PicamLightSource_NeAr</td>
<td>Neon and Argon mixed light source</td>
</tr>
<tr>
<td>PicamLightSource_Qth</td>
<td>Quartz Tungsten Halogen light source.</td>
</tr>
</tbody>
</table>

5.1.1.19 *PicamLightSourceStatus*

**Data Type**

*PicamLightSourceStatus* is defined as `enum`.

**Description**

*PicamLightSourceStatus* is the light source stability.

**Enumerator Definitions**

Refer to Table 5-19 for enumerator definitions.

**Table 5-19: *PicamLightSourceStatus* Enumerator Definitions**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamLightSourceStatus_Unstable</td>
<td>The light source is unstable.</td>
</tr>
<tr>
<td>PicamLightSourceStatus_Stable</td>
<td>The light source is stable.</td>
</tr>
</tbody>
</table>
5.1.1.20 PicamModulationTrackingMask

Data Type
PicamModulationTrackingMask is defined as enum.

Description
PicamModulationTrackingMask is the set of modulation parameters that are to be tracked.

Enumerator Definitions
Refer to step 5-20 for enumerator definitions.

Table 5-20: PicamModulationTrackingMask Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamModulationTrackingMask_Duration</td>
<td>The modulation duration is tracked.</td>
</tr>
<tr>
<td>PicamModulationTrackingMask_Frequency</td>
<td>The modulation frequency is tracked.</td>
</tr>
<tr>
<td>PicamModulationTrackingMask_None</td>
<td>No components are tracked.</td>
</tr>
<tr>
<td>PicamModulationTrackingMask_OutputSignalFrequency</td>
<td>The modulation output signal frequency is tracked.</td>
</tr>
<tr>
<td>PicamModulationTrackingMask_Phase</td>
<td>The modulation phase is tracked.</td>
</tr>
</tbody>
</table>

5.1.1.21 PicamOrientationMask

Data Type
PicamOrientationMask is defined as enum.

Description
PicamOrientationMask is the set of image orientation descriptors.

Enumerator Definitions
Refer to Table 5-21 for enumerator definitions.

Table 5-21: PicamOrientationMask Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamOrientationMask_FlippedHorizontally</td>
<td>The data is flipped about the centered, vertical axis relative to normal.</td>
</tr>
<tr>
<td>PicamOrientationMask_FlippedVertically</td>
<td>The data is flipped about the centered, horizontal axis relative to normal.</td>
</tr>
<tr>
<td>PicamOrientationMask_Normal</td>
<td>This defines a standard orientation.</td>
</tr>
</tbody>
</table>
5.1.1.22 PicamOutputSignal

Data Type
PicamOutputSignal is defined as enum.

Description
PicamOutputSignal is the set of parameters defining the hardware's MONITOR OUTPUT signal.

Enumerator Definitions
Refer to Table 5-22 for enumerator definitions.

Table 5-22: PicamOutputSignal Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamOutputSignal_Acquiring</td>
<td>The signal is high when the hardware is acquiring or ready to receive the first trigger.</td>
</tr>
<tr>
<td>PicamOutputSignal_AlwaysHigh</td>
<td>The signal is always high.</td>
</tr>
<tr>
<td>PicamOutputSignal_AlwaysLow</td>
<td>The signal is always low.</td>
</tr>
<tr>
<td>PicamOutputSignal_AuxOutput</td>
<td>The signal is high during an AUX output pulse.</td>
</tr>
<tr>
<td>PicamOutputSignal_Busy</td>
<td>The signal is high when the hardware is busy.</td>
</tr>
<tr>
<td>PicamOutputSignal_EffectivelyExposing</td>
<td>The signal is high for the entire duration the sensor is exposed.</td>
</tr>
<tr>
<td>PicamOutputSignal_EffectivelyExposing_Alternation</td>
<td>The signal is high for the entire duration the sensor is exposed; every other frame beginning with the first.</td>
</tr>
<tr>
<td>PicamOutputSignal_Exposing</td>
<td>The signal is high when the sensor is exposed as requested.</td>
</tr>
<tr>
<td>PicamOutputSignal_Gate</td>
<td>The signal is high during a gate pulse.</td>
</tr>
<tr>
<td>PicamOutputSignal_InternalTriggerT0</td>
<td>The signal is high during $t_0$ of the internal trigger.</td>
</tr>
<tr>
<td>PicamOutputSignal_NotReadingOut</td>
<td>The signal is low when the sensor is reading out.</td>
</tr>
<tr>
<td>PicamOutputSignal_ReadingOut</td>
<td>The signal is high when the sensor is reading out.</td>
</tr>
<tr>
<td>PicamOutputSignal_ShiftingUnderMask</td>
<td>The signal is high when the image is shifting under the sensor's mask.</td>
</tr>
<tr>
<td>PicamOutputSignal_ShutterOpen</td>
<td>The signal is high when the shutter is open.</td>
</tr>
<tr>
<td>PicamOutputSignal_WaitingForTrigger</td>
<td>The signal is high when the hardware is waiting for a trigger.</td>
</tr>
</tbody>
</table>
5.1.1.23 PicamParameter

Data Type

PicamParameter is defined as enum.

Description

PicamParameter is the set of user-accessible hardware parameters.

Enumerator Definitions

Refer to Table 5-23 for enumerator definitions.

Table 5-23: PicamParameter Enumerator Definitions (Sheet 1 of 11)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamParameter_Accumulations</td>
<td>Controls the number of on-sensor accumulations</td>
</tr>
<tr>
<td>PicamParameter_ActiveBottomMargin</td>
<td>Controls the inactive number of rows on the bottom.</td>
</tr>
<tr>
<td>PicamParameter_ActiveExtendedHeight</td>
<td>Controls the number of additional active rows that can be used for storage.</td>
</tr>
<tr>
<td></td>
<td>NOTE: These rows cannot be imaged directly.</td>
</tr>
<tr>
<td>PicamParameter_ActiveHeight</td>
<td>Controls the active number of rows.</td>
</tr>
<tr>
<td>PicamParameter_ActiveLeftMargin</td>
<td>Controls the inactive number of columns on the left.</td>
</tr>
<tr>
<td>PicamParameter_ActiveRightMargin</td>
<td>Controls the inactive number of columns on the right.</td>
</tr>
<tr>
<td>PicamParameter_ActiveShutter</td>
<td>Selects the shutter via the PicamActiveShutter data enumeration.</td>
</tr>
<tr>
<td></td>
<td>Refer to Section 5.1.1, PicamActiveShutter, on page 69 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_ActiveTopMargin</td>
<td>Controls the inactive number of rows on the top.</td>
</tr>
<tr>
<td>PicamParameter_ActiveWidth</td>
<td>Controls the active number of columns.</td>
</tr>
<tr>
<td>PicamParameter_AdcAnalogGain</td>
<td>Controls the electronic gain of the pixel digitization via the PicamAdcAnalogGain data enumeration.</td>
</tr>
<tr>
<td></td>
<td>Refer to Section 5.1.1.2, PicamAdcAnalogGain, on page 69 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_AdcBitDepth</td>
<td>Controls the resolution of the pixel digitization in bits-per-pixel.</td>
</tr>
<tr>
<td>PicamParameter_AdcEMGain</td>
<td>Controls the electromagnetic gain in terms of multiples.</td>
</tr>
<tr>
<td>PicamParameter_AdcQuality</td>
<td>Controls the nature of pixel digitization via the PicamAdcQuality data enumeration.</td>
</tr>
<tr>
<td></td>
<td>Refer to Section 5.1.1.3, PicamAdcQuality, on page 70 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_AdcSpeed</td>
<td>Controls the rate pixels are digitized, in MHz.</td>
</tr>
<tr>
<td>PicamParameter_Age</td>
<td>Reports the age measured in minutes.</td>
</tr>
<tr>
<td>PicamParameter_AnticipateTrigger</td>
<td>Uses an external pre-trigger to anticipate an external trigger.</td>
</tr>
<tr>
<td>PicamParameter_AuxOutput</td>
<td>Controls the auxiliary output gate pulse.</td>
</tr>
<tr>
<td>PicamParameter_BracketGating</td>
<td>Enables bracket pulsing for intensified hardware.</td>
</tr>
</tbody>
</table>
Table 5-23: **PicamParameter** Enumerator Definitions (Sheet 2 of 11)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamParameter_CcdCharacteristics</td>
<td>Reports characteristics of a CCD sensor via the PicamCcdCharacteristicsMask data enumeration. Refer to Section 5.1.1.4, PicamCcdCharacteristicsMask, on page 71 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_CenterWavelengthReading</td>
<td>Reports the actual position of the center wavelength in nanometers (nm).</td>
</tr>
<tr>
<td>PicamParameter_CenterWavelengthSet</td>
<td>Controls the target position of the center wavelength in nanometers (nm). Refer to Section 5.1.1.5, PicamCenterWavelengthStatus, on page 72 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_CenterWavelengthStatus</td>
<td>Reports if the center wavelength is moving.</td>
</tr>
<tr>
<td>PicamParameter_CleanBeforeExposure</td>
<td>Controls cleaning before each exposure.</td>
</tr>
<tr>
<td>PicamParameter_CleanCycleCount</td>
<td>Controls the number of clean cycles to run before acquisition begins.</td>
</tr>
<tr>
<td>PicamParameter_CleanCycleHeight</td>
<td>Controls the number of rows in a clean cycle.</td>
</tr>
<tr>
<td>PicamParameter_CleanSectionFinalHeight</td>
<td>Controls the final height rows for exponential decomposition cleaning.</td>
</tr>
<tr>
<td>PicamParameter_CleanSectionFinalHeightCount</td>
<td>Controls the final height iterations for exponential decomposition cleaning.</td>
</tr>
<tr>
<td>PicamParameter_CleanSerialRegister</td>
<td>Controls the cleaning of the serial register itself.</td>
</tr>
<tr>
<td>PicamParameter_CleanUntilTrigger</td>
<td>Controls the nature of cleaning while waiting for an external trigger.</td>
</tr>
<tr>
<td>PicamParameter_CoolingFanStatus</td>
<td>Reports the status of the cooling fan via the PicamCoolingFanStatus data enumeration. Refer to Section 5.1.1.7, PicamCoolingFanStatus, on page 73 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_CorrectPixelBias</td>
<td>Enables pixel bias correction.</td>
</tr>
<tr>
<td>PicamParameter_CustomModulationSequence</td>
<td>Customizes a modulation sequence.</td>
</tr>
<tr>
<td>PicamParameter_DelayFromPreTrigger</td>
<td>Specifies the delay from pre-trigger to trigger in microseconds (µS).</td>
</tr>
<tr>
<td>PicamParameter_DifEndingGate</td>
<td>Controls the second gate pulse in DIF readout in nanoseconds (nS).</td>
</tr>
<tr>
<td>PicamParameter_DifStartingGate</td>
<td>Controls the initial gate pulse in DIF readout in nanoseconds (nS).</td>
</tr>
<tr>
<td>PicamParameter_DisableCoolingFan</td>
<td>Enables/disables the thermoelectric cooling fan.</td>
</tr>
<tr>
<td>PicamParameter_DisableDataFormatting</td>
<td>Controls the basic processing necessary to receive data in the expected format.</td>
</tr>
<tr>
<td>PicamParameter_EMImiccdGain</td>
<td>Optimally controls the intensifier gain and electron multiplication gain in emICCD hardware in terms of multiples.</td>
</tr>
<tr>
<td>Enumerator</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PicamParameter_EMICcdGainControlMode</td>
<td>Determines how the intensifier gain and electron multiplication gain are controlled in emICCD hardware via the PicamEMICcdGainControlMode data enumeration. Refer to Section 5.1.1.8, PicamEMICcdGainControlMode, on page 73, for additional information.</td>
</tr>
<tr>
<td>PicamParameter_EnableAuxOutput</td>
<td>Enables the AUX output pulse.</td>
</tr>
<tr>
<td>PicamParameter_EnableIntensifier</td>
<td>Enables the intensifier. <strong>NOTE:</strong> The intensifier must be enabled and powered on for it to function.</td>
</tr>
<tr>
<td>PicamParameter_EnableModulation</td>
<td>Enables RF modulation for intensified hardware.</td>
</tr>
<tr>
<td>PicamParameter_EnableModulationOutputSignal</td>
<td>Enables an RF output signal from intensified hardware to be used as the user sees fit.</td>
</tr>
<tr>
<td>PicamParameter_EnableNondestructiveReadout</td>
<td>Allows the hardware to periodically readout while exposing.</td>
</tr>
<tr>
<td>PicamParameter_EnableSensorWindowHeater</td>
<td>Enables the sensor window to heat up in an effort to prevent condensation.</td>
</tr>
<tr>
<td>PicamParameter_EnableSyncMaster</td>
<td>Enables SyncMASTER1 and SyncMASTER2 gate pulses.</td>
</tr>
<tr>
<td>PicamParameter_ExactReadoutCountMaximum</td>
<td>Reports the maximum number of readouts the hardware can acquire. <strong>NOTE:</strong> This does not include non-destructive readouts from hardware that supports the feature.</td>
</tr>
<tr>
<td>PicamParameter_ExactReadoutCountMaximum</td>
<td>Reports the maximum number of readouts the hardware can acquire. <strong>NOTE:</strong> This does not include non-destructive readouts from hardware that supports the feature.</td>
</tr>
<tr>
<td>PicamParameter_ExposureTime</td>
<td>Controls the time the sensor is exposed in milliseconds (mS).</td>
</tr>
<tr>
<td>PicamParameter_ExternalShutterStatus</td>
<td>Reports the status of the shutter that is external to the hardware via the PicamShutterStatus data enumeration. Refer to Section 5.1.1.31, PicamShutterStatus, on page 95 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_ExternalShutterType</td>
<td>Reports the type of shutter that is external to, and can be driven by, the hardware via the PicamShutterType data enumeration. Refer to Section 5.1.1.33, PicamShutterType, on page 97 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_FocalLength</td>
<td>Reports the optical path length from the focusing mirror to the sensor in millimeters (mm).</td>
</tr>
<tr>
<td>PicamParameter_FrameRateCalculation</td>
<td>Reports the estimated frame rate in frames-per-second. <strong>NOTE:</strong> If there is more than one frame-per-readout, this represents the burst frame rate within the readout.</td>
</tr>
<tr>
<td>PicamParameter_FrameRateCalculation</td>
<td>Reports the estimated frame rate in frames-per-second. <strong>NOTE:</strong> If there is more than one frame-per-readout, this represents the burst frame rate within the readout.</td>
</tr>
<tr>
<td>PicamParameter_FrameSize</td>
<td>Reports the size, in bytes, of a data frame.</td>
</tr>
<tr>
<td>PicamParameter_FramesPerReadout</td>
<td>Reports the number of frames contained in one readout.</td>
</tr>
<tr>
<td>PicamParameter_FrameStride</td>
<td>Reports the length, in bytes, necessary to traverse to the next frame.</td>
</tr>
</tbody>
</table>
Table 5-23: *PicamParameter* Enumerator Definitions (Sheet 4 of 11)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamParameter_FrameTrackingBitDepth</td>
<td>Controls the frame tracking number size in bits-per-pixel.</td>
</tr>
<tr>
<td>PicamParameter_GateTracking</td>
<td>Controls the tracking of a sequential gate pulse in metadata via the <em>PicamGateTrackingMask</em> data enumeration. Refer to Section 5.1.1.9, <em>PicamGateTrackingMask</em>, on page 74 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_GateTrackingBitDepth</td>
<td>Controls the size of one component in a varying sequential gate pulse. <em>NOTE:</em> This metadata is floating point.</td>
</tr>
<tr>
<td>PicamParameter_GatingMode</td>
<td>Controls the nature of gate pulse timing via the <em>PicamGatingMode</em> data enumeration. Refer to Section 5.1.1.10, <em>PicamGatingMode</em>, on page 74 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_GatingSpeed</td>
<td>Classifies the narrowest gate pulse. Refer to Section 5.1.1.11, <em>PicamGatingSpeed</em>, on page 75 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_GratingBlazing Wavelength</td>
<td>Reports the blaze at a particular wavelength.</td>
</tr>
<tr>
<td>PicamParameter_GratingCoating</td>
<td>Reports the coating on the grating. Refer to Section 5.1.1.12, <em>PicamGratingCoating</em>, on page 75 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_GratingGrooveDensity</td>
<td>Reports the groove density of the grating in grooves per millimeter.</td>
</tr>
<tr>
<td>PicamParameter_GratingType</td>
<td>Reports the type of grating. Refer to Section 5.1.1.13, <em>PicamGratingType</em>, on page 76 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_InactiveShutterTimingModeResult</td>
<td>Reports the state of the inactive shutter via the <em>PicamShutterTimingMode</em> data enumeration. Refer to Section 5.1.1.32, <em>PicamShutterTimingMode</em>, on page 96 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_InclusionAngle</td>
<td>Reports the sum of the incident and diffracted ray angles relative to the grating normal vector in degrees.</td>
</tr>
<tr>
<td>PicamParameter_InputTriggerStatus</td>
<td>Reports if an external trigger source is connected. Refer to Section 5.1.1.39, <em>PicamTriggerStatus</em>, on page 100 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_IntensifierDiameter</td>
<td>Reports the diameter of the intensifier in millimeters (mm).</td>
</tr>
<tr>
<td>PicamParameter_IntensifierGain</td>
<td>Controls the gain of the intensifier in terms of multiples.</td>
</tr>
<tr>
<td>PicamParameter_IntensifierOptions</td>
<td>Reports additional features of intensified hardware via the <em>PicamIntensifierOptionsMask</em> data enumeration. Refer to Section 5.1.1.14, <em>PicamIntensifierOptionsMask</em>, on page 76 for additional information.</td>
</tr>
</tbody>
</table>
Table 5-23: **PicamParameter** Enumerator Definitions (Sheet 5 of 11)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamParameter_IntensifierStatus</td>
<td>Reports the status of the intensifier power via the PicamIntensifierStatus data enumeration. Refer to Section 5.1.15, PicamIntensifierStatus, on page 77 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_InternalShutterStatus</td>
<td>Reports the status of the shutter that is internal to the hardware via the PicamShutterStatus data enumeration. Refer to Section 5.1.1.31, PicamShutterStatus, on page 95 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_InternalShutterType</td>
<td>Reports the type of shutter that is internal to, and can be driven by, the hardware via the PicamShutterType data enumeration. Refer to Section 5.1.1.33, PicamShutterType, on page 97 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_InvertOutputSignal</td>
<td>Controls if the timing signal is inverted when viewed from the hardware monitor.</td>
</tr>
<tr>
<td>PicamParameter_InvertOutputSignal2</td>
<td>Controls if the timing signal is inverted when viewed from the second hardware monitor.</td>
</tr>
<tr>
<td>PicamParameter_KineticsWindowHeight</td>
<td>Controls the number of rows used for the sensing window in a kinetics readout.</td>
</tr>
<tr>
<td>PicamParameter_LaserOutputMode</td>
<td>Controls when the laser produces light. Refer to Section 5.1.16, PicamLaserOutputMode, on page 77 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_LaserPower</td>
<td>Controls the laser power as a multiplier.</td>
</tr>
<tr>
<td>PicamParameter_LaserStatus</td>
<td>Reports if the laser is ready. Refer to Section 5.1.17, PicamLaserStatus, on page 77 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_LifeExpectancy</td>
<td>Reports the expected lifetime measured in minutes.</td>
</tr>
<tr>
<td>PicamParameter_LightSource</td>
<td>Controls the light source on the lamp. Refer to Section 5.1.18, PicamLightSource, on page 78 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_LightSourceStatus</td>
<td>Reports if the light source has stabilized. Refer to Section 5.1.1.19, PicamLightSourceStatus, on page 78 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_MaskedBottomMargin</td>
<td>Controls the number of masked rows akin to active bottom margin.</td>
</tr>
<tr>
<td>PicamParameter_MaskedHeight</td>
<td>Controls the number of masked rows akin to active height.</td>
</tr>
<tr>
<td>PicamParameter_MaskedTopMargin</td>
<td>Controls the number of masked rows akin to active top margin.</td>
</tr>
<tr>
<td>PicamParameter_ModulationDuration</td>
<td>Controls the time the intensifier is modulating in milliseconds (mS).</td>
</tr>
<tr>
<td>PicamParameter_ModulationFrequency</td>
<td>Controls the frequency of the intensifier modulation in MHz.</td>
</tr>
</tbody>
</table>
Table 5-23: PicamParameter Enumerator Definitions (Sheet 6 of 11)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamParameter_ModulationOutputSignal Amplitude</td>
<td>Controls the peak-to-peak amplitude of the user RF output signal in volts (V).</td>
</tr>
<tr>
<td>PicamParameter_ModulationOutputSignal Frequency</td>
<td>Controls the frequency of the user RF output signal in MHz.</td>
</tr>
<tr>
<td>PicamParameter_ModulationTracking</td>
<td>Controls the tracking of a sequential phase or custom modulation sequence in metadata via the PicamModulationTrackingMask data enumeration. Refer to Section 5.1.1.20, PicamModulationTrackingMask, on page 79 for additional information.</td>
</tr>
</tbody>
</table>
| PicamParameter_ModulationTrackingBit Depth       | Controls the size of one component in a varying sequential modulation phase or custom modulation sequence.  

**NOTE:** This metadata is floating point. |
| PicamParameter_NondestructiveReadout Period      | Controls the rate at which the hardware will non-destructively readout during exposure in seconds (S).  

**NOTE:** This duration must be less than exposure time for any non-destructive readouts to occur. |
| PicamParameter_NormalizeOrientation              | Controls automatic orientation correction for data due to readout ports used.                   |
| PicamParameter_OnlineReadoutRate Calculation     | Reports the fastest possible readout rate that could occur given the current setup and accounting for possible changes to online hardware parameters while acquiring. |
| PicamParameter_Orientation                       | Reports the orientation of the data via the PicamOrientationMask data enumeration. Refer to Section 5.1.1.21, PicamOrientationMask, on page 79 for additional information. |
| PicamParameter_OutputSignal                      | Controls what timing signal is issued from the hardware monitor via the PicamOutputSignal data enumeration. Refer to Section 5.1.1.22, PicamOutputSignal, on page 80 for additional information. |
| PicamParameter_OutputSignal2                     | Controls what timing signal is issued from the second hardware monitor via the PicamOutputSignal data enumeration. Refer to Section 5.1.1.22, PicamOutputSignal, on page 80 for additional information. |
| PicamParameter_PhotonDecayDelay                  | Controls the length of time the hardware waits for the phosphor to decay before reading out. The time unit depends on PicamParameter_PhotonDecayDelay Resolution. |
| PicamParameter_PhotonDecayDelay Resolution        | Controls the time unit used for phosphor decay delay. This value is in microseconds (µS). Example: A resolution of:  
- 1 signifies delay is in microseconds (µS);  
- 1000 for milliseconds (mS);  
- 0.01 for tens-of-nanoseconds (nS). |
### Table 5-23: `PicamParameter` Enumerator Definitions (Sheet 7 of 11)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamParameter_PhosphorType</td>
<td>Reports the type of phosphor used in intensified hardware. Refer to Section 5.1.1.24, PicamPhosphorType, on page 92 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_PhotocathodeSensitivity</td>
<td>Classifies the wavelength sensitivity of the photocathode. Refer to Section 5.1.1.25, PicamPhotocathodeSensitivity, on page 92 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_PhotonDetectionMode</td>
<td>Enables/disables photon detection and controls how it is done via the <code>PicamPhotonDetectionMode</code> data enumeration. Refer to Section 5.1.1.26, PicamPhotonDetectionMode, on page 93 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_PhotonDetectionThreshold</td>
<td>The threshold, in counts, used to distinguish photons from background for each pixel.</td>
</tr>
<tr>
<td>PicamParameter_PixelBitDepth</td>
<td>Reports the size of a data pixel in bits-per-pixel.</td>
</tr>
<tr>
<td>PicamParameter_PixelFormat</td>
<td>Controls the format of a data pixel via the <code>PicamPixelFormat</code> data enumeration. Refer to Section 5.1.1.27, PicamPixelFormat, on page 93 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_PixelGapHeight</td>
<td>Reports the vertical distance between pixels, in microns.</td>
</tr>
<tr>
<td>PicamParameter_PixelGapWidth</td>
<td>Reports the horizontal distance between pixels, in microns.</td>
</tr>
<tr>
<td>PicamParameter_PixelHeight</td>
<td>Reports the pixel height, in microns.</td>
</tr>
<tr>
<td>PicamParameter_PixelWidth</td>
<td>Reports the pixel width, in microns.</td>
</tr>
<tr>
<td>PicamParameter_ReadoutControlMode</td>
<td>Controls how the sensor is read out via the <code>PicamReadoutControlMode</code> data enumeration. Refer to Section 5.1.1.28, PicamReadoutControlMode, on page 94 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_ReadoutCount</td>
<td>Controls the number of readouts to acquire before stopping the hardware. NOTE: The hardware may acquire more than the readouts requested for large requests (i.e., more readouts than those specified by <code>PicamParameter_ExactReadoutCount Maximum</code>).</td>
</tr>
<tr>
<td></td>
<td>NOTE: This does not include non-destructive readouts from hardware that supports such a feature.</td>
</tr>
<tr>
<td></td>
<td>NOTE: [Advanced-API Usage Only] The value 0 indicates the hardware will run forever until explicitly stopped or an error occurs.</td>
</tr>
<tr>
<td>PicamParameter_ReadoutOrientation</td>
<td>Reports the orientation of the data due to readout port location via the <code>PicamOrientationMask</code> data enumeration. Refer to Section 5.1.1.21, PicamOrientationMask, on page 79 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_ReadoutPortCount</td>
<td>Controls the number of readout ports from which the hardware should simultaneously read data.</td>
</tr>
</tbody>
</table>
## PicamParameter Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamParameter_ReadoutRateCalculation</td>
<td>Reports the estimated rate of data in readouts-per-second. <strong>NOTE:</strong> If the hardware is being externally triggered, this represents the fastest possible rate.</td>
</tr>
<tr>
<td>PicamParameter_ReadoutStride</td>
<td>Reports the length, in bytes, necessary to traverse to the next readout.</td>
</tr>
<tr>
<td>PicamParameter_ReadoutTimeCalculation</td>
<td>Reports the duration of time it takes for the hardware to read out one readout in milliseconds (mS).</td>
</tr>
<tr>
<td>PicamParameter_RepetitiveGate</td>
<td>Controls the constant gate pulse in nanoseconds (nS).</td>
</tr>
<tr>
<td>PicamParameter_RepetitiveModulation Phase</td>
<td>Controls the constant phase of the intensifier with respect to the modulation output signal in degrees.</td>
</tr>
<tr>
<td>PicamParameter_Rois</td>
<td>Controls the area of the sensor to be digitized via the PicamRois structure. Refer to Section 5.2.1.2, PicamRois, on page 107 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_SecondaryActiveHeight</td>
<td>Controls the number of secondary active rows.</td>
</tr>
<tr>
<td>PicamParameter_SecondaryMaskedHeight</td>
<td>Controls the number of secondary masked rows.</td>
</tr>
<tr>
<td>PicamParameter_SensorActiveBottom Margin</td>
<td>Reports the inactive rows on the bottom.</td>
</tr>
<tr>
<td>PicamParameter_SensorActiveExtended Height</td>
<td>Reports the number of additional active rows that can be used for storage. <strong>NOTE:</strong> These rows cannot be imaged directly.</td>
</tr>
<tr>
<td>PicamParameter_SensorActiveHeight</td>
<td>Reports the active number of rows.</td>
</tr>
<tr>
<td>PicamParameter_SensorActiveLeftMargin</td>
<td>Reports the inactive columns on the left.</td>
</tr>
<tr>
<td>PicamParameter_SensorActiveRightMargin</td>
<td>Reports the inactive columns on the right.</td>
</tr>
<tr>
<td>PicamParameter_SensorActiveTopMargin</td>
<td>Reports the inactive rows on the top.</td>
</tr>
<tr>
<td>PicamParameter_SensorActiveWidth</td>
<td>Reports the active number of columns.</td>
</tr>
<tr>
<td>PicamParameter_SensorAngle</td>
<td>Reports the angle between rays striking the sensor relative to the sensor normal vector in degrees.</td>
</tr>
<tr>
<td>PicamParameter_SensorMaskedBottom Margin</td>
<td>Reports the number of masked rows akin to active bottom margin.</td>
</tr>
<tr>
<td>PicamParameter_SensorMaskedHeight</td>
<td>Reports the number of masked rows akin to active height.</td>
</tr>
<tr>
<td>PicamParameter_SensorMaskedTopMargin</td>
<td>Reports the number of masked rows akin to active top margin.</td>
</tr>
<tr>
<td>PicamParameter_SensorSecondaryActive Height</td>
<td>Reports the number of secondary active rows.</td>
</tr>
<tr>
<td>PicamParameter_SensorSecondaryMasked Height</td>
<td>Reports the number of secondary masked rows.</td>
</tr>
<tr>
<td>PicamParameter_SensorTemperature Reading</td>
<td>Reports the temperature of the sensor in degrees C.</td>
</tr>
</tbody>
</table>
Table 5-23: **PicamParameter** Enumerator Definitions (Sheet 9 of 11)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamParameter_SensorTemperatureSetPoint</td>
<td>Controls the target temperature for the sensor in degrees C.</td>
</tr>
<tr>
<td>PicamParameter_SensorTemperatureStatus</td>
<td>Reports the status of the sensor temperature via the PicamSensorTemperatureStatus data enumeration. Refer to Section 5.1.1.29, PicamSensorTemperatureStatus, on page 94 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_SensorType</td>
<td>Reports the kind of sensor being used via the PicamSensorType data enumeration. Refer to Section 5.1.1.30, PicamSensorType, on page 95 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_SeNsRWindowHeight</td>
<td>Controls the height of the unmasked area used in SeNsR readout.</td>
</tr>
<tr>
<td>PicamParameter_SequentialEndingGate</td>
<td>Controls the last gate pulse in a sequence in nanoseconds (nS).</td>
</tr>
<tr>
<td>PicamParameter_SequentialEndingModulationPhase</td>
<td>Controls the last modulation phase of the intensifier with respect to the modulation output signal in a sequence in degrees.</td>
</tr>
<tr>
<td>PicamParameter_SequentialGateStepCount</td>
<td>Controls the number of gate pulse steps in a sequence.</td>
</tr>
<tr>
<td>PicamParameter_SequentialGateStepIterations</td>
<td>Controls the number of gate pulses at each step in a sequence.</td>
</tr>
<tr>
<td>PicamParameter_SequentialStartingGate</td>
<td>Controls the first gate pulse in a sequence in nanoseconds (nS).</td>
</tr>
<tr>
<td>PicamParameter_SequentialStartingModulationPhase</td>
<td>Controls the first modulation phase of the intensifier with respect to the modulation output signal in a sequence in degrees.</td>
</tr>
<tr>
<td>PicamParameter_ShutterClosingDelay</td>
<td>Controls the duration of time the hardware waits for the shutter to close before reading out. The time unit depends on PicamParameter_ShutterDelayResolution.</td>
</tr>
<tr>
<td>PicamParameter_ShutterDelayResolution</td>
<td>Controls the time unit used for shutter opening/closing delay. This value is in microseconds. Example: A resolution of: • 1 signifies delay is in microseconds (μS); • 1000 for milliseconds (mS); • 0.01 for tens-of-nanoseconds (nS).</td>
</tr>
<tr>
<td>PicamParameter_ShutterOpeningDelay</td>
<td>Controls the duration of time the hardware waits for the shutter to open before exposing. The time unit depends on PicamParameter_ShutterDelayResolution.</td>
</tr>
<tr>
<td>PicamParameter_ShutterTimingMode</td>
<td>Controls the behavior of the shutter during acquisition. Refer to Section 5.1.1.32, PicamShutterTimingMode, on page 96 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_StopCleaningOnPreTrigger</td>
<td>Stops sensor cleaning when an external pre-trigger is acknowledged.</td>
</tr>
</tbody>
</table>
Table 5-23: **PicamParameter** Enumerator Definitions (Sheet 10 of 11)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamParameter_SyncMaster2Delay</td>
<td>Controls the delay of SyncMASTER2 relative to SyncMASTER1 in microseconds (µS).</td>
</tr>
<tr>
<td>PicamParameter_TimeStampBitDepth</td>
<td>Controls the time stamp size in bits-per-pixel. * NOTE: Because time stamps may be negative one bit is reserved for sign.</td>
</tr>
<tr>
<td>PicamParameter_TimeStampResolution</td>
<td>Controls the time stamp resolution in ticks-per-second. * NOTE: This value is computer-dependent when time stamps are software generated.</td>
</tr>
<tr>
<td>PicamParameter_TimeStamps</td>
<td>Controls time stamp metadata via the PicamTimeStampsMask data enumeration. Refer to Section 5.1.1.34, PicamTimeStampsMask, on page 97 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_TrackFrames</td>
<td>Controls frame tracking metadata.</td>
</tr>
<tr>
<td>PicamParameter_TriggerCoupling</td>
<td>Controls the coupling between an external trigger source and the camera input via the PicamTriggerCoupling data enumeration. Refer to Section 5.1.1.35, PicamTriggerCoupling, on page 98 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_TriggerDelay</td>
<td>Controls the delay between an external trigger and the camera’s response in nanoseconds.</td>
</tr>
<tr>
<td>PicamParameter_TriggerDetermination</td>
<td>Controls what the hardware recognizes as an external trigger via the PicamTriggerDetermination data enumeration. Refer to Section 5.1.1.36, PicamTriggerDetermination, on page 98 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_TriggerFrequency</td>
<td>Controls the internal trigger and SyncMASTER frequency in Hz.</td>
</tr>
<tr>
<td>PicamParameter_TriggerResponse</td>
<td>Controls the hardware’s behavior in response to a trigger via the PicamTriggerResponse data enumeration. Refer to Section 5.1.1.37, PicamTriggerResponse, on page 99 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_TriggerResponse</td>
<td>Controls the hardware’s behavior in response to an external trigger via the PicamTriggerResponse data enumeration. Refer to Section 5.1.1.37, PicamTriggerResponse, on page 99 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_TriggerSource</td>
<td>Controls the source of a trigger via the PicamTriggerSource data enumeration. Refer to Section 5.1.1.38, PicamTriggerSource, on page 99 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_TriggerTermination</td>
<td>Controls the termination of an external trigger source at the hardware input via the PicamTriggerTermination data enumeration. Refer to Section 5.1.1.40, PicamTriggerTermination, on page 100 for additional information.</td>
</tr>
<tr>
<td>PicamParameter_TriggerThreshold</td>
<td>Controls the voltage threshold necessary for the hardware to recognize a trigger in volts (V).</td>
</tr>
</tbody>
</table>
### Table 5-23: PicamParameter Enumerator Definitions (Sheet 11 of 11)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>***PicamParameter_VacuumStatus</td>
<td>Reports the status of the vacuum chamber containing the sensor.</td>
</tr>
<tr>
<td>PicamParameter_VerticalShiftRate</td>
<td>Controls the rate to shift one row towards the serial register in a CCD in microseconds (μS).</td>
</tr>
</tbody>
</table>
5.1.1.24 **PicamPhosphorType**

**Data Type**

*PicamPhosphorType* is defined as *enum*.

**Description**

*PicamPhosphorType* is the set of phosphor types within intensified hardware.

**Enumerator Definitions**

Refer to Table 5-24 for enumerator definitions.

**Table 5-24: PicamPhosphorType Enumerator Definitions**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamPhosphorType_P43</td>
<td>The phosphor is P43.</td>
</tr>
<tr>
<td>PicamPhosphorType_P46</td>
<td>The phosphor is P46.</td>
</tr>
</tbody>
</table>

5.1.1.25 **PicamPhotocathodeSensitivity**

**Data Type**

*PicamPhotocathodeSensitivity* is defined as *enum*.

**Description**

*PicamPhotocathodeSensitivity* is the set of parameters used to define the photocathode’s wavelength range.

**Enumerator Definitions**

Refer to Table 5-25 for enumerator definitions.

**Table 5-25: PicamPhotocathodeSensitivity Enumerator Definitions**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamPhotocathodeSensitivity_HighBlueFilmless</td>
<td>Improved quantum efficiency and optimized for blue wavelengths.</td>
</tr>
<tr>
<td>PicamPhotocathodeSensitivity_HighQEFilmless</td>
<td>Improved quantum efficiency.</td>
</tr>
<tr>
<td>PicamPhotocathodeSensitivity_HighRedFilmless</td>
<td>Improved quantum efficiency and optimized for red wavelengths.</td>
</tr>
<tr>
<td>PicamPhotocathodeSensitivity_InGaAsFilmless</td>
<td>Extends into near-infrared wavelengths.</td>
</tr>
<tr>
<td>PicamPhotocathodeSensitivity_RedBlue</td>
<td>Spans red and blue wavelengths.</td>
</tr>
<tr>
<td>PicamPhotocathodeSensitivity_SolarBlind</td>
<td>Optimized only for ultraviolet wavelengths.</td>
</tr>
<tr>
<td>PicamPhotocathodeSensitivity_SuperBlue</td>
<td>Optimized for blue wavelengths.</td>
</tr>
<tr>
<td>PicamPhotocathodeSensitivity_SuperRed</td>
<td>Optimized for red wavelengths.</td>
</tr>
<tr>
<td>PicamPhotocathodeSensitivity_Unigen2Filmless</td>
<td>Coated with UNIGEN2.</td>
</tr>
<tr>
<td>PicamPhotocathodeSensitivity_UV</td>
<td>Optimized for ultraviolet wavelengths.</td>
</tr>
</tbody>
</table>
5.1.1.26 PicamPhotonDetectionMode

Data Type
PicamPhotonDetectionMode is defined as enum.

Description
PicamPhotonDetectionMode is the set of photon detection modes.

Enumerator Definitions
Refer to Table 5-26 for enumerator definitions.

Table 5-26: PicamPhotonDetectionMode Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamPhotonDetectionMode_Clipping</td>
<td>Each pixel whose intensity is greater than or equal to the threshold is a photon and retains its original value. Otherwise the value is 0.</td>
</tr>
<tr>
<td>PicamPhotonDetectionMode_Disabled</td>
<td>Photon detection is disabled.</td>
</tr>
<tr>
<td>PicamPhotonDetectionMode_Thresholding</td>
<td>Each pixel whose intensity is greater than or equal to the threshold is a photon and replaced with a count of 1. Otherwise the value is 0.</td>
</tr>
</tbody>
</table>

5.1.1.27 PicamPixelFormat

Data Type
PicamPixelFormat is defined as enum.

Description
PicamPixelFormat is the set of characteristics that defines the format of a data pixel.

Enumerator Definitions
Refer to Table 5-27 for enumerator definitions.

Table 5-27: PicamPixelFormat Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamPixelFormat_Monochrome16Bit</td>
<td>16 bits of monochrome data</td>
</tr>
<tr>
<td>PicamPixelFormat_Monochrome32Bit</td>
<td>32 bits of monochrome data</td>
</tr>
</tbody>
</table>
5.1.1.28 PicamReadoutControlMode

Data Type
PicamReadoutControlMode is defined as enum.

Description
PicamReadoutControlMode is the set of sensor readout modes.

Enumerator Definitions
Refer to Table 5-28 for enumerator definitions.

Table 5-28: PicamReadoutControlMode Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamReadoutControlMode_Dif</td>
<td>The Dual Imaging Feature where the sensor acquires two frames rapidly and then reads them both out.</td>
</tr>
<tr>
<td>PicamReadoutControlMode_FrameTransfer</td>
<td>The sensor is reading out a frame while exposing the next frame.</td>
</tr>
<tr>
<td>PicamReadoutControlMode_FullFrame</td>
<td>The sensor is read one frame at a time.</td>
</tr>
<tr>
<td>PicamReadoutControlMode_Interline</td>
<td>The sensor is reading out a frame while exposing the next frame.</td>
</tr>
<tr>
<td>PicamReadoutControlMode_Kinetics</td>
<td>The sensor rapidly stores multiple frames and then reads those out.</td>
</tr>
<tr>
<td>PicamReadoutControlMode_RollingShutter</td>
<td>The sensor is reading out a row while exposing the next row.</td>
</tr>
<tr>
<td>PicamReadoutControlMode_SeNsR</td>
<td>The sensor accumulates frames by alternating between two different phases</td>
</tr>
<tr>
<td>PicamReadoutControlMode_SpectraKinetics</td>
<td>Same as kinetics, but optimized to capture a larger burst of spectral frames.</td>
</tr>
</tbody>
</table>

5.1.1.29 PicamSensorTemperatureStatus

Data Type
PicamSensorTemperatureStatus is defined as enum.

Description
PicamSensorTemperatureStatus is the set of sensor temperature statuses.

Enumerator Definitions
Refer to Table 5-29 for enumerator definitions.

Table 5-29: PicamSensorTemperatureStatus Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamSensorTemperatureStatus_Faulted</td>
<td>Sensor cooling has malfunctioned.</td>
</tr>
<tr>
<td>PicamSensorTemperatureStatus_Locked</td>
<td>The temperature has stabilized at the set point.</td>
</tr>
<tr>
<td>PicamSensorTemperatureStatus_Unlocked</td>
<td>The temperature has not stabilized at the set point.</td>
</tr>
</tbody>
</table>
5.1.1.30  PicamSensorType

Data Type

PicamSensorType is defined as enum.

Description

PicamSensorType is the set of sensor types.

Enumerator Definitions

Refer to Table 5-30 for enumerator definitions.

Table 5-30:  PicamSensorType Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamSensorType_Ccd</td>
<td>The sensor is a CCD.</td>
</tr>
<tr>
<td>PicamSensorType_Cmos</td>
<td>The sensor is a CMOS.</td>
</tr>
<tr>
<td>PicamSensorType_InGaAs</td>
<td>The sensor is an InGaAs.</td>
</tr>
</tbody>
</table>

5.1.1.31  PicamShutterStatus

Data Type

PicamShutterStatus is defined as enum.

Description

PicamShutterStatus is the set of shutter statuses.

Enumerator Definitions

Refer to Table 5-31 for enumerator definitions.

Table 5-31:  PicamShutterStatus Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamShutterStatus_Connected</td>
<td>A shutter is connected.</td>
</tr>
<tr>
<td>PicamShutterStatus_NotConnected</td>
<td>No shutter is connected.</td>
</tr>
<tr>
<td>PicamShutterStatus_Overheated</td>
<td>A connected shutter has overheated and is temporarily disabled.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> If a shutter becomes overheated, data acquisition will stop and cannot be started again until the shutter is no longer overheated.</td>
</tr>
</tbody>
</table>
5.1.1.32 PicamShutterTimingMode

Data Type
PicamShutterTimingMode is defined as enum.

Description
PicamShutterTimingMode is the set of shutter behaviors during data acquisition.

Enumerator Definitions
Refer to Table 5-32 for enumerator definitions.

Table 5-32: PicamShutterTimingMode Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamShutterTimingMode_Normal</td>
<td>The shutter only opens during exposure time. <strong>NOTE:</strong> During PicamReadoutControlMode_Kinetics readout, the shutter stays open while storing frames.</td>
</tr>
<tr>
<td>PicamShutterTimingMode_AlwaysClosed</td>
<td>The shutter is always closed. <strong>NOTE:</strong> This mode is also valid when not acquiring data.</td>
</tr>
<tr>
<td>PicamShutterTimingMode_AlwaysOpen</td>
<td>The shutter is always open. <strong>NOTE:</strong> This mode is also valid when not acquiring data.</td>
</tr>
<tr>
<td>PicamShutterTimingMode_OpenBeforeTrigger</td>
<td>The shutter opens ahead of time while waiting for a trigger. This is different from PicamShutterTimingMode_Normal where the shutter opens in reaction to a trigger.</td>
</tr>
</tbody>
</table>
5.1.1.33 PicamShutterType

Data Type
PicamShutterType is defined as enum.

Description
PicamShutterType is the set of shutter types.

NOTE:
This does not indicate the presence of a shutter, only the kind of shutter that could be driven. PicamShutterStatus indicates the presence of a shutter.

Enumerator Definitions
Refer to Table 5-33 for enumerator definitions.

Table 5-33: PicamShutterType Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamShutterType_None</td>
<td>No shutter.</td>
</tr>
<tr>
<td>PicamShutterType_ProntorMagnetic0</td>
<td>PRONTOR magnetic 0 shutter.</td>
</tr>
<tr>
<td>PicamShutterType_ProntorMagneticE40</td>
<td>PRONTOR magnetic E/40 shutter.</td>
</tr>
<tr>
<td>PicamShutterType_VincentCS25</td>
<td>Vincent CS25 shutter</td>
</tr>
<tr>
<td>PicamShutterType_VincentCS45</td>
<td>Vincent CS45 shutter</td>
</tr>
<tr>
<td>PicamShutterType_VincentCS90</td>
<td>Vincent CS90 shutter</td>
</tr>
<tr>
<td>PicamShutterType_VincentDSS10</td>
<td>Vincent DSS10 shutter</td>
</tr>
<tr>
<td>PicamShutterType_VincentVS25</td>
<td>Vincent VS25 shutter</td>
</tr>
<tr>
<td>PicamShutterType_VincentVS35</td>
<td>Vincent VS35 shutter</td>
</tr>
</tbody>
</table>

5.1.1.34 PicamTimeStampsMask

Data Type
PicamTimeStampsMask is defined as enum.

Description
PicamTimeStampsMask is the set of time stamp metadata.

Enumerator Definitions
Refer to Table 5-34 for enumerator definitions.

Table 5-34: PicamTimeStampsMask Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamTimeStampsMask_None</td>
<td>No time stamps are generated during acquisition.</td>
</tr>
<tr>
<td>PicamTimeStampsMask_ExposureStarted</td>
<td>The time will be stamped when exposure starts.</td>
</tr>
<tr>
<td>PicamTimeStampsMask_ExposureEnded</td>
<td>The time will be stamped when exposure ends.</td>
</tr>
</tbody>
</table>
5.1.1.35 PicamTriggerCoupling

Data Type
PicamTriggerCoupling is defined as enum.

Description
PicamTriggerCoupling is the set of coupling modes between an external trigger and the hardware's input.

Enumerator Definitions
Refer to Table 5-35 for enumerator definitions.

Table 5-35: PicamTriggerCoupling Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamTriggerCoupling_AC</td>
<td>The components are AC-coupled.</td>
</tr>
<tr>
<td>PicamTriggerCoupling_DC</td>
<td>The components are DC-coupled.</td>
</tr>
</tbody>
</table>

5.1.1.36 PicamTriggerDetermination

Data Type
PicamTriggerDetermination is defined as enum.

Description
PicamTriggerDetermination is the set external trigger styles that are recognized by hardware.

Enumerator Definitions
Refer to Table 5-36 for enumerator definitions.

Table 5-36: PicamTriggerDetermination Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamTriggerDetermination_AlternatingEdgeRising</td>
<td>The first trigger is a signal's rising edge and subsequent triggers alternate direction.</td>
</tr>
<tr>
<td>PicamTriggerDetermination_AlternatingEdgeFalling</td>
<td>The first trigger is a signal's falling edge and subsequent triggers alternate direction.</td>
</tr>
<tr>
<td>PicamTriggerDetermination_FallingEdge</td>
<td>The trigger is a signal's falling edge.</td>
</tr>
<tr>
<td>PicamTriggerDetermination_NegativePolarity</td>
<td>The trigger is initially a signal's falling edge and then level-sensitive to a low signal for the rest of the acquisition.</td>
</tr>
<tr>
<td>PicamTriggerDetermination_PositivePolarity</td>
<td>The trigger is initially a signal's rising edge and then level-sensitive to a high signal for the rest of the acquisition.</td>
</tr>
<tr>
<td>PicamTriggerDetermination_RisingEdge</td>
<td>The trigger is a signal's rising edge.</td>
</tr>
</tbody>
</table>
5.1.1.37 PicamTriggerResponse

Data Type
PicamTriggerResponse is defined as enum.

Description
PicamTriggerResponse is the set of the hardware's responses to an external trigger.

Enumerator Definitions
Refer to Table 5-37 for enumerator definitions.

Table 5-37: PicamTriggerResponse Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamTriggerResponse_ExposeDuringTriggerPulse</td>
<td>Each trigger controls when exposure begins and ends.</td>
</tr>
<tr>
<td>PicamTriggerResponse_GatePerTrigger</td>
<td>The hardware generates a gate pulse after the trigger.</td>
</tr>
<tr>
<td>PicamTriggerResponse_NoResponse</td>
<td>The hardware does not respond to triggering.</td>
</tr>
<tr>
<td>PicamTriggerResponse_ReadoutPerTrigger</td>
<td>The hardware reads out the sensor after each trigger.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> For hardware that can non-destructively readout, all non-destructive readouts associated with the normal readout will occur on the same trigger as the normal readout.</td>
</tr>
<tr>
<td>PicamTriggerResponse_ShiftPerTrigger</td>
<td>Each trigger moves to the next frame on the sensor.</td>
</tr>
<tr>
<td>PicamTriggerResponse_StartOnSingleTrigger</td>
<td>The hardware begins acquisition after a single trigger.</td>
</tr>
</tbody>
</table>

5.1.1.38 PicamTriggerSource

Data Type
PicamTriggerSource is defined as enum.

Description
PicamTriggerSource is the set of trigger sources.

Enumerator Definitions
Refer to Table 5-38 for enumerator definitions.

Table 5-38: PicamTriggerSource Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamTriggerSource_External</td>
<td>Triggers originate from an external source.</td>
</tr>
<tr>
<td>PicamTriggerSource_Internal</td>
<td>Triggers originate from the hardware itself.</td>
</tr>
<tr>
<td>PicamTriggerSource_None</td>
<td>No triggers.</td>
</tr>
</tbody>
</table>
5.1.1.39 PicamTriggerStatus

Data Type

PicamTriggerStatus is defined as enum.

Description

PicamTriggerStatus is the presence of an external trigger source.

Enumerator Definitions

Refer to Table 5-39 for enumerator definitions.

Table 5-39: PicamTriggerStatus Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamTriggerStatus_NotConnected</td>
<td>No trigger source is connected.</td>
</tr>
<tr>
<td>PicamTriggerStatus_Connected</td>
<td>A trigger source is connected.</td>
</tr>
</tbody>
</table>

5.1.1.40 PicamTriggerTermination

Data Type

PicamTriggerTermination is defined as enum.

Description

PicamTriggerTermination is the set of input terminations provided by the hardware for an external trigger source.

Enumerator Definitions

Refer to Table 5-40 for enumerator definitions.

Table 5-40: PicamTriggerTermination Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamTriggerTermination_FiftyOhms</td>
<td>The trigger terminates into 50 ohms.</td>
</tr>
<tr>
<td>PicamTriggerTermination_HighImpedance</td>
<td>The trigger terminates into very high impedance.</td>
</tr>
</tbody>
</table>

5.1.1.41 PicamVacuumStatus

Data Type

PicamVacuumStatus is defined as enum.

Description

PicamVacuumStatus reports the status of the vacuum chamber containing the sensor.

Enumerator Definitions

Refer to Table 5-41 for enumerator definitions.

Table 5-41: PicamVacuumStatus Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamVacuumStatus_Sufficient</td>
<td>The chamber has sufficient vacuum.</td>
</tr>
<tr>
<td>PicamVacuumStatus_Low</td>
<td>The chamber has low or no vacuum.</td>
</tr>
</tbody>
</table>
5.1.1.42 PicamValueType

Data Type

PicamValueType is defined as enum.

Description

PicamValueType is the set of parameter value data types.

Enumerator Definitions

Refer to Table 5-42 for enumerator definitions.

Table 5-42: PicamValueType Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamValueType_Boolean</td>
<td>Accessed as piint.</td>
</tr>
<tr>
<td></td>
<td>• FALSE = 0</td>
</tr>
<tr>
<td></td>
<td>• TRUE = 1</td>
</tr>
<tr>
<td>PicamValueType_Enumeration</td>
<td>Any enum accessed as piint.</td>
</tr>
<tr>
<td>PicamValueType_FloatingPoint</td>
<td>Accessed as piflt.</td>
</tr>
<tr>
<td>PicamValueType_Integer</td>
<td>Accessed as piint.</td>
</tr>
<tr>
<td>PicamValueType_LargeInteger</td>
<td>Accessed as pi64s.</td>
</tr>
<tr>
<td>PicamValueType_Modulations</td>
<td>Accessed as PicamModulations</td>
</tr>
<tr>
<td>PicamValueType_Pulse</td>
<td>Accessed as PicamPulse.</td>
</tr>
<tr>
<td>PicamValueType_Rois</td>
<td>Accessed as PicamRois.</td>
</tr>
</tbody>
</table>
5.1.2 Parameter Access Enumerations

This section provides detailed information about the following parameter access enumerations:

- PicamValueAccess

5.1.2.1 PicamValueAccess

Data Type

PicamValueAccess is defined as enum.

Description

PicamValueAccess is the set of permitted parameter access.

Enumerator Definitions

Refer to Table 5-43 for enumerator definitions.

Table 5-43: PicamValueAccess Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamValueAccess_ReadOnly</td>
<td>The stored parameter value can only be read.</td>
</tr>
<tr>
<td>PicamValueAccess_ReadWriteTrivial</td>
<td>The stored parameter value can be read and/or overwritten, but there is only one value for this parameter.</td>
</tr>
<tr>
<td>PicamValueAccess_ReadWrite</td>
<td>The stored parameter value can be read and/or overwritten.</td>
</tr>
</tbody>
</table>
5.1.3 Parameter Constraint Enumerations

This section provides detailed information about the following parameter constraint enumerations:

- PicamConstraintScope
- PicamConstraintSeverity
- PicamConstraintCategory
- PicamRoisConstraintRulesMask

5.1.3.1 PicamConstraintScope

Data Type

PicamConstraintScope is defined as enum.

Description

PicamConstraintScope is the set of constraint dependencies.

Enumerator Definitions

Refer to Table 5-44 for enumerator definitions.

Table 5-44: PicamConstraintScope Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamConstraintScope_Independent</td>
<td>The constraint has no dependencies and is therefore constant.</td>
</tr>
<tr>
<td>PicamConstraintScope_Dependent</td>
<td>The constraint has dependencies and therefore is variable.</td>
</tr>
</tbody>
</table>

5.1.3.2 PicamConstraintSeverity

Data Type

PicamConstraintSeverity is defined as enum.

Description

PicamConstraintSeverity is the set of severities when failing a constraint.

Enumerator Definitions

Refer to Table 5-45 for enumerator definitions.

Table 5-45: PicamConstraintSeverity Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamConstraintSeverity_Error</td>
<td>Failure indicates the value is in error.</td>
</tr>
<tr>
<td>PicamConstraintSeverity_Warning</td>
<td>Failure indicates the value is in warning and notice should be taken.</td>
</tr>
</tbody>
</table>
5.1.3.3 PicamConstraintCategory

Data Type
PicamConstraintCategory is defined as enum.

Description
PicamConstraintCategory is the set of constraint categories.

Enumerator Definitions
Refer to Table 5-46 for enumerator definitions.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamConstraintCategory_Capable</td>
<td>Which set of values are ultimately possible.</td>
</tr>
<tr>
<td>PicamConstraintCategory_Required</td>
<td>Which set of values are currently permissible.</td>
</tr>
<tr>
<td>PicamConstraintCategory_Recommended</td>
<td>Which set of values fall within a recommended range for most scenarios.</td>
</tr>
</tbody>
</table>
5.1.3.4 PicamRoisConstraintRulesMask

Data Type
PicamRoisConstraintRulesMask is defined as enum.

Description
PicamRoisConstraintRulesMask is the set of complex rules that defines a valid set of regions of interest.

Enumerator Definitions
Refer to Table 5-47 for enumerator definitions.

Table 5-47: PicamRoisConstraintRulesMask Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamRoisConstraintRulesMask_None</td>
<td>No additional rules.</td>
</tr>
<tr>
<td>PicamRoisConstraintRulesMask_X BinningAlignment</td>
<td>Regions sharing columns must bin those columns equally. This means not only must they contain equal x-binning values, the regions must also begin on x-binning boundaries.</td>
</tr>
<tr>
<td>PicamRoisConstraintRulesMask_Y BinningAlignment</td>
<td>Regions sharing rows must bin those rows equally. This means not only must they contain equal y-binning values, the regions must also begin on y-binning boundaries.</td>
</tr>
<tr>
<td>PicamRoisConstraintRulesMask_HorizontalSymmetry</td>
<td>Regions must be symmetrical about the line between the two center-most columns. Either one region must bisect this line or two regions must be reflective to each other about this line.</td>
</tr>
<tr>
<td>PicamRoisConstraintRulesMask_VerticalSymmetry</td>
<td>Regions must be symmetrical about the line between the two center-most rows. Either one region must bisect this line or two regions must be reflective to each other about this line.</td>
</tr>
<tr>
<td>PicamRoisConstraintRulesMask_SymmetryBoundsBinning</td>
<td>A region required to bisect a line of symmetry may not bin pixels together that fall on both sides of the line.</td>
</tr>
</tbody>
</table>
5.2 Data Structure Definitions

This section provides programming information for the following PICam data structure definitions:

- Camera-Specific Parameter Data Structures
  - PicamRoi
  - PicamRois
  - PicamPulse
  - PicamModulation
  - PicamModulations
- Shared Camera/Accessory Parameter Data Structures
  - PicamStatusPurview

5.2.1 Camera-Specific Parameter Data Structures

This section provides detailed programming information about camera-specific parameter data structures.

5.2.1.1 PicamRoi

Description

PicamRoi defines a single Region of Interest (ROI.)

Structure Definition

The structure definition for PicamRoi is:

```c
typedef struct PicamRoi
{
    piint x;
    piint width;
    piint x_binning;
    piint y;
    piint height;
    piint y_binning;
} PicamRoi;
```

Variable Definitions

The variables required by PicamRoi are:

- x: The left-most column coordinate (zero-based).
- width: The number of columns.
- x_binning: The number of columns to group into a sum.
- y: The top-most row coordinate (zero-based).
- height: The number of rows.
- y_binning: The number of rows to group into a sum.
5.2.1.2 PicamRois

Description
PicamRois defines a set of non-overlapping Regions of Interest (ROIs.)

Structure Definition
The structure definition for PicamRois is:

```c
typedef struct PicamRois {
    PicamRoi* roi_array;
    pint roi_count;
} PicamRois;
```

Variable Definitions
The variables required by PicamRois are:

- `roi_array`: An array of one or more regions.
- `roi_count`: The number of regions.

5.2.1.3 PicamPulse

Description
PicamPulse defines a gate pulse.

Structure Definition
The structure definition for PicamPulse is:

```c
typedef struct PicamPulse {
    piflt delay;
    piflt width;
} PicamPulse;
```

Variable Definitions
The variables required by PicamPulse are:

- `delay`: The delay until a gate pulse begins.
- `width`: The width of the gate pulse.
5.2.1.4 PicamModulation

Description
PicamModulation defines a custom intensifier modulation sequence point.

Structure Definition
The structure definition for PicamModulation is:

```c
typedef struct PicamModulation
{
    piflt duration;
    piflt frequency;
    piflt phase;
    piflt output_signal_frequency;
} PicamModulation;
```

Variable Definitions
The variables required by PicamModulation are:

- duration: The time, in mS, the intensifier is modulating.
- frequency: The frequency, in MHz, of the intensifier modulation.
- phase: The phase, in degrees, of the intensifier with respect to the modulation output signal.
- output_signal_frequency: The frequency, in MHz, of the user RF output signal.

5.2.1.5 PicamModulations

Description
PicamModulations defines a sequence of intensifier modulation sequence points.

Structure Definition
The structure definition for PicamModulations is:

```c
typedef struct PicamModulations
{
    PicamModulation* modulation_array;
    piint modulation_count;
} PicamModulations;
```

Variable Definitions
The variables required by PicamModulations are:

- modulation_array: An array of one or more sequence points.
- modulation_count: The number of sequence points.
5.2.2 Shared Camera/Accessory Parameter Data Structures

This section provides detailed programming information about shared camera/accessory parameter data structures.

5.2.2.1 PicamStatusPurview

Description
PicamStatusPurview defines the scope of a status.

Structure Definition
The structure definition for PicamStatusPurview is:

```c
typedef struct PicamStatusPurview {
    const uint* values_array;
    uint values_count;
} PicamStatusPurview;
```

Variable Definitions
The variables required by PicamStatusPurview are:

- `values_array`: The allowable status values.
- `values_count`: The number of allowable status values.
5.3 Parameter Constraints

This section provides programming information for the following PICam parameter constraints:

- Camera-Specific Parameter Constraints
  - PicamRoisConstraint
  - PicamPulseConstraint
  - PicamModulationsConstraint
- Shared Camera/Accesory Parameter Constraints
  - PicamCollectionConstraint
  - PicamRangeConstraint

5.3.1 Camera-Specific Parameter Constraints

This section provides detailed programming information about the following camera-specific parameter constraint data structures:

- PicamRoisConstraint
- PicamPulseConstraint
- PicamModulationsConstraint

5.3.1.1 PicamRoisConstraint

Description

PicamRoisConstraint defines the constraints placed on a set of Regions of Interest (ROIs).

**NOTE:**

Regions of Interest may not overlap.

Structure Definition

The structure definition for PicamRoisConstraint is:

```c
typedef struct PicamRoisConstraint
{
    PicamConstraintScope scope;
    PicamConstraintSeverity severity;
    pibln empty_set;
    PicamRoisConstraintRulesMask rules;
    piint maximum_roi_count;
    PicamRangeConstraint x_constraint;
    PicamRangeConstraint width_constraint;
    const piint* x_binning_limits_array;
    piint x_binning_limits_count;
    PicamRangeConstraint y_constraint;
    PicamRangeConstraint height_constraint;
    const piint* y_binning_limits_array;
    piint y_binning_limits_count;
} PicamRoisConstraint;
```

*continued on next page*
Variable Definitions
The variables required by PicamRoisConstraint are:

- **scope**: The scope of the constraint.
- **severity**: The severity of the constraint.
- **empty_set**: Indicates when there are no valid Regions of Interest defined.
  Valid values are:
  - **TRUE**: There are no valid ROIs defined.
    When TRUE, only scope and severity are relevant.
  - **FALSE**: There is at least one valid ROI defined.
- **rules**: Complex set of rules to which a parameter of this type must adhere.
- **maximum_roi_count**: The maximum number of ROIs permitted.
- **x_constraint**: The constraint governing the value of PicamRoi.x.
- **width_constraint**: The constraint governing the value of PicamRoi.width.
- **x_binning_limits_array**: The list of valid values for PicamRoi.x_binning.
  **NOTE**: An additional requirement is that PicamRoi.x_binning must always divide evenly into PicamRoi.width.
  This is null when no additional limits are required.
- **x_binning_limits_count**: The number of items in x_binning_limits_array.
  This is 0 when no additional limits are required.
- **y_constraint**: The constraint governing the value of PicamRoi.y.
- **height_constraint**: The constraint governing the value of PicamRoi.height.
- **y_binning_limits_array**: The list of valid values for PicamRoi.y_binning.
  **NOTE**: An additional requirement is that PicamRoi.y_binning must always divide evenly into PicamRoi.height.
  This is null when no additional limits are required.
- **y_binning_limits_count**: The number of items in y_binning_limits_array.
  This is 0 when no additional limits are required.
5.3.1.2 **PicamPulseConstraint**

**Description**

`PicamPulseConstraint` defines the constraints placed on a valid gate pulse.

**Structure Definition**

The structure definition for `PicamPulseConstraint` is:

```c
typedef struct PicamPulseConstraint
{
    PicamConstraintScope scope;
    PicamConstraintSeverity severity;
    pibln empty_set;
    PicamRangeConstraint delay_constraint;
    PicamRangeConstraint width_constraint;
    piflt minimum_duration;
    piflt maximum_duration;
} PicamPulseConstraint;
```

**Variable Definitions**

The variables required by `PicamPulseConstraint` are:

- **scope**: The scope of the constraint.
- **severity**: The severity of the constraint.
- **empty_set**: Indicates when there are no valid Pulses defined. Valid values are:
  - TRUE
    - There are no valid Pulses defined.
    - When TRUE, only `scope` and `severity` are relevant.
  - FALSE
    - There is at least one valid Pulse defined.
- **delay_constraint**: The constraint governing the value of `PicamPulse.delay`.
- **width_constraint**: The constraint governing the value of `PicamPulse.width`.
- **minimum_duration**: The minimum numeric value for:
  - `PicamPulse.delay + PicamPulse.width`
- **maximum_duration**: The maximum numeric value for:
  - `PicamPulse.delay + PicamPulse.width`
5.3.1.3 `PicamModulationsConstraint`

**Description**

`PicamModulationsConstraint` defines the constraints placed on custom intensifier modulation sequence points.

**Structure Definition**

The structure definition for `PicamModulationsConstraint` is:

```c
typedef struct PicamModulationsConstraint
{
    PicamConstraintScope scope;
    PicamConstraintSeverity severity;
    pibln empty_set;
    pint maximum_modulation_count;
    PicamRangeConstraint duration_constraint;
    PicamRangeConstraint frequency_constraint;
    PicamRangeConstraint phase_constraint;
    PicamRangeConstraint output_signal_frequency_constraint;
} PicamModulationsConstraint;
```

**Variable Definitions**

The variables required by `PicamModulationsConstraint` are:

- **scope**: The scope of the constraint.
- **severity**: The severity of the constraint.
- **empty_set**: Indicates when there are no valid modulation points defined.
  
  Valid values are:
  
  - **TRUE**
    
    There are no valid modulation points defined.
    
    When TRUE, only `scope` and `severity` are relevant.
  
  - **FALSE**
    
    There is at least one valid modulation point defined.

- **maximum_modulation_count**: The maximum number of modulation sequence points.

- **duration_constraint**: The constraint governing the value of `PicamModulation.duration`.

- **frequency_constraint**: The constraint governing the value of `PicamModulation.frequency`.

- **phase_constraint**: The constraint governing the value of `PicamModulation.phase`.

- **output_signal_frequency_constraint**: The constraint governing the value of `PicamModulation.output_signal_frequency`. 
5.3.2 Shared Camera/Accessory Parameter Constraints

This section provides detailed programming information about the following shared camera and accessory parameter constraint data structures:

- PicamCollectionConstraint
- PicamRangeConstraint

5.3.2.1 PicamCollectionConstraint

Description

PicamCollectionConstraint defines the constraints placed on a variable whose value is selected from a list of predefined values.

Structure Definition

The structure definition for PicamCollectionConstraint is:

```c
typedef struct PicamCollectionConstraint
{
    PicamConstraintScope scope;
    PicamConstraintSeverity severity;
    const piflt* values_array;
    piint values_count;
} PicamCollectionConstraint;
```

Variable Definitions

The variables required by PicamCollectionConstraint are:

- **scope**: The scope of the constraint.
- **severity**: The severity of the constraint.
- **values_array**: The allowable values.
- **values_count**: The number of allowable values.
5.3.2.2 PicamRangeConstraint

Description
PicamRangeConstraint defines the constraints placed a numeric variable whose value lies within a linear range of numeric values.

Structure Definition
The structure definition for PicamRangeConstraint is:

typedef struct PicamRangeConstraint
{
    PicamConstraintScope scope;
    PicamConstraintSeverity severity;
    pbln empty_set;
    piflt minimum;
    piflt maximum;
    piflt increment;
    const piflt* excluded_values_array;
    piint excluded_values_count;
    const piflt* outlying_values_array;
    piint outlying_values_count;
} PicamRangeConstraint;

Variable Definitions
The variables required by PicamRangeConstraint are:

    scope:  The scope of the constraint.
    severity:  The severity of the constraint.
    empty_set:  Indicates when there are no valid values within the range. Valid values are:
        • TRUE
        There are no valid values within the range.
        When TRUE, only scope and severity are relevant.
        • FALSE
        There is at least one valid value within the range.
    minimum:  The smallest value within the range.
        NOTE:  outlying_values_array may include a smaller value.
    maximum:  The largest value within the range.
        NOTE:  outlying_values_array may include a larger value.
    increment:  The numeric gap between consecutive values within the range.
    excluded_values_array:  The set of values within the range (excluding minimum and maximum) that is not valid.
        This is null when all values within the range are valid.
    excluded_values_count:  The number of items within excluded_values_array.
        This is 0 when there are no excluded values.
    outlying_values_array:  The set of valid values that lie outside of the range of values.
        This is null when no valid values fall outside of the range.
    outlying_values_count:  The number of items within outlying_values_array.
        This is 0 when there are no outlying values.
5.4 Programmers' Reference for Configuration APIs

This section provides a detailed programmers' reference guide for the following configuration APIs:

- **Camera-Specific Parameter Value APIs**
  - Picam_GetParameterLargeIntegerValue()
  - Picam_CanSetParameterLargeIntegerValue()
  - Picam_SetParameterLargeIntegerValue()
  - Picam_DestroyRois()
  - Picam_GetParameterRoisValue()
  - Picam_CanSetParameterRoisValue()
  - Picam_SetParameterRoisValue()
  - Picam_DestroyPulses()
  - Picam_GetParameterPulseValue()
  - Picam_CanSetParameterPulseValue()
  - Picam_SetParameterPulseValue()
  - Picam_DestroyModulations()
  - Picam_GetParameterModulationsValue()
  - Picam_CanSetParameterModulationsValue()
  - Picam_SetParameterModulationsValue()
  - Picam_GetParameterLargeIntegerDefaultValue()
  - Picam_GetParameterRoisPointDefaultValue()
  - Picam_GetParameterPulseDefaultValue()
  - Picam_GetParameterModulationsDefaultValue()
  - Picam_SetParameterIntegerValueOnline()
  - Picam_SetParameterFloatingPointValueOnline()

- **Shared Camera/Accessory Parameter Value APIs**
  - Picam_GetParameterIntegerValue()
  - Picam_CanSetParameterIntegerValue()
  - Picam_SetParameterIntegerValue()
  - Picam_GetParameterFloatingPointValue()
  - Picam_CanSetParameterFloatingPointValue()
  - Picam_SetParameterFloatingPointValue()
  - Picam_GetParameterIntegerDefaultValue()
  - Picam_GetParameterFloatingPointDefaultValue()
  - Picam_RestoreParametersToDefaultValues()
  - Picam_CanSetParameterOnline()
  - Picam_CanReadParameter()
  - Picam_ReadParameterIntegerValue()
  - Picam_ReadParameterFloatingPointValue()
  - Picam_CanWaitForStatusParameter()
  - Picam_GetStatusParameterPurview()
  - Picam_DestroyStatusPurviews()
  - Picam_EstimateTimeToStatusParameterValue()
  - Picam_WaitForStatusParameterValue()

- **Shared Camera/Accessory Parameter Information APIs**
  - Picam_DestroyParameters()
  - Picam_GetParameters()
  - Picam.DoesParameterExist()
  - Picam_IsParameterRelevant()
  - Picam_GetParameterValueType()
  - Picam_GetParameterEnumeratedType()
  - Picam_GetParameterValueAccess()
  - Picam.GetParameterConstraintType()
• Camera-Specific Parameter Constraints APIs
  - `Picam_DestroyRoisConstraints()`
  - `Picam_GetParameterRoisConstraint()`
  - `Picam_DestroyPulseConstraints()`
  - `Picam_GetParameterPulseConstraint()`
  - `Picam_DestroyModulationsConstraints()`
  - `Picam_GetParameterModulationsConstraint()`

• Shared Camera/Accessory Parameter Constraints APIs
  - `Picam_DestroyCollectionConstraints()`
  - `Picam_GetParameterRangeConstraint()`
  - `Picam_GetParameterCollectionConstraint()`
  - `Picam_DestroyRangeConstraints()`

• Shared Camera/Accessory Parameter Commitment APIs
  - `Picam_AreParametersCommitted()`
  - `Picam_CommitParameters()`
5.4.1 Camera-Specific Parameter Value APIs

This section provides programming information for APIs used when working with camera-specific parameter values.

5.4.1.1 Picam_GetParameterLargeIntegerValue()

Description
Picam_GetParameterLargeIntegerValue() returns the current large integer value for a specified parameter.

Syntax
The syntax for Picam_GetParameterLargeIntegerValue() is:

```c
PICAM_API Picam_GetParameterLargeIntegerValue(
    PicamHandle camera,
    PicamParameter parameter,
    pi64s* value);
```

Input Parameters
Input parameters for Picam_GetParameterLargeIntegerValue() are:

- `camera`: Handle for the camera for which the large integer value is being requested.
- `parameter`: Specifies the parameter that is to be queried. Valid parameters are those of type PicamValueType_LargeInteger.

Output Parameters
Output parameters for Picam_GetParameterLargeIntegerValue() are:

- `value`: Pointer to the large integer value of the specified parameter.

Advanced API Usage
When used in conjunction with Advanced APIs, `camera` may be a handle to either the:

- model, or
- device.

Stored values for any specific parameter are not necessarily the same for the device and model instances.
5.4.1.2  Picam_CanSetParameterLargeIntegerValue()

Description
Picam_CanSetParameterLargeIntegerValue() determines if a large integer value is valid for a specified parameter.

Syntax
The syntax for Picam_CanSetParameterLargeIntegerValue() is:

```
PICAM_API Picam_CanSetParameterLargeIntegerValue(
    PicamHandle camera,
    PicamParameter parameter,
    pi64s value,
    pibln* settable);
```

Input Parameters
Input parameters for Picam_CanSetParameterLargeIntegerValue() are:

- camera: Handle for the camera for which the value/parameter combination is being tested.
- parameter: Specifies the parameter which is to be tested.
- value: The large integer value that is to be tested.

Output Parameters
Output parameters for Picam_CanSetParameterLargeIntegerValue() are:

- settable: Pointer to the test results. Indicates if the large integer value is a valid value for the specified parameter. Valid values are:
  - TRUE: Indicates that the large integer value is a valid value for the specified parameter.
  - FALSE: Indicates that the large integer value is an invalid value for the specified parameter.

Advanced API Usage
When used in conjunction with Advanced APIs, camera may be a handle to either the:

- model,
- device.
**5.4.1.3 Picam_SetParameterLargeIntegerValue()**

**Description**

`Picam_SetParameterLargeIntegerValue()` sets a parameter to a specified large integer value during camera setup.

**Syntax**

The syntax for `Picam_SetParameterLargeIntegerValue()` is:

```c
PICAM_API Picam_SetParameterLargeIntegerValue(
    PicamHandle camera,
    PicamParameter parameter,
    pi64s value);
```

**Input Parameters**

Input parameters for `Picam_SetParameterLargeIntegerValue()` are:

- `camera`: Handle for the camera being configured.
- `parameter`: Specifies the parameter that is to be set with a large integer value. Valid parameters are those of type `PicamValueType_LargeInteger`.
- `value`: The large integer value to which the parameter is to be set.

**Output Parameters**

There are no output parameters associated with `Picam_SetParameterLargeIntegerValue()`.

**Advanced API Usage**

When used in conjunction with Advanced APIs, `camera` may be a handle to either the:

- **model**;
  - The `model` parameter may be set independently from the corresponding `device` parameter. However, doing so requires that all parameters be committed to the `device` prior to starting any data acquisition by calling `Picam_CommitParameters()`.
- **device**.
  - Setting a `device` parameter automatically sets the corresponding `model` parameter to the same value.

**Related APIs**

For additional information, refer to the following related APIs:

- `Picam_CommitParameters()`
5.4.1.4 Picam_DestroyRois()

Description
Picam_DestroyRois() releases memory that has been allocated by PICam for use by the array rois.

If rois is null, calling Picam_DestroyRois() has no effect.

Syntax
The syntax for Picam_DestroyRois() is:

```
PICAM_API Picam_DestroyRois(
    const PicamRois* rois);
```

Input Parameters
Input parameters for Picam_DestroyRois() are:

- rois: Pointer to array memory that is to be released.

Output Parameters
There are no output parameters associated with Picam_DestroyRois().
### 5.4.1.5 Picam_GetParameterRoisValue()

**Description**

`Picam_GetParameterRoisValue()` returns the current value for a specified Rois parameter.

**Syntax**

The syntax for `Picam_GetParameterRoisValue()` is:

```c
PICAM_API Picam_GetParameterRoisValue(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamRois** value);
```

**Input Parameters**

Input parameters for `Picam_GetParameterRoisValue()` are:

- `camera`: Handle for the camera for which the Rois parameter value is being requested.
- `parameter`: Specifies the Rois parameter for which the current value is to be returned.
  Valid parameters are those of type `PicamValueType_Rois`.

**Output Parameters**

Output parameters for `Picam_GetParameterRoisValue()` are:

- `value`: Pointer to the memory location in which the value of the specified Rois parameter has been stored.
  
  **NOTE:** This memory is allocated by PICam and must be released by calling `Picam_DestroyRois()`.

**Advanced API Usage**

When used in conjunction with Advanced APIs, `camera` may be a handle to either the:

- model, or
- device.

Stored values for any specific parameter are not necessarily the same for the device and model instances.

**Related Structures**

For additional information, refer to the following ROI structures:

- `PicamRoi`
- `PicamRois`

**Related APIs**

For additional information, refer to the following related APIs:

- `Picam_DestroyRois()`
5.4.16  Picam_CanSetParameterRoisValue()

Description
Picam_CanSetParameterRoisValue() determines if a value is valid for a specified Rois parameter.

Syntax
The syntax for Picam_CanSetParameterRoisValue() is:

```c
PICAM_API Picam_CanSetParameterRoisValue(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamRois* value,
    pibln* settable);
```

Input Parameters
Input parameters for Picam_CanSetParameterRoisValue() are:

- `camera`: Handle for the camera for which the value/parameter combination is being validated.
- `parameter`: Specifies the Rois parameter.
- `value`: The value that is to be tested.

Output Parameters
Output parameters for Picam_CanSetParameterRoisValue() are:

- `settable`: Pointer to the test results. Indicates if the value is valid for the specified Rois parameter. Valid values are:
  - `TRUE`: Indicates that the value is valid for the specified Rois parameter.
  - `FALSE`: Indicates that the value is not valid for the specified Rois parameter.

Advanced API Usage
When used in conjunction with Advanced APIs, `camera` may be a handle to either the:

- `model`, or
- `device`.

Related Structures
For additional information, refer to the following ROI constraint structures:

- `PicamRoisConstraint`. 
5.4.1.7 Picam_SetParameterRoisValue()

Description
Picam_SetParameterRoisValue() configures an Rois parameter to a specified value during camera setup.

Syntax
The syntax for Picam_SetParameterRoisValue() is:

```c
PICAM_API Picam_SetParameterRoisValue(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamRois* value);
```

Input Parameters
Input parameters for Picam_SetParameterRoisValue() are:

- camera: Handle for the camera being configured.
- parameter: Specifies the Rois parameter that is to be configured. Valid parameters are those of type PicamValueType_Rois.
- value: The value to which the Rois parameter is to be set.

Output Parameters
There are no output parameters associated with Picam_SetParameterRoisValue().

Advanced API Usage
When used in conjunction with Advanced APIs, camera may be a handle to either the:

- model;
  - The model parameter may be set independently from the corresponding device parameter. However, doing so requires that all parameters be committed to the device prior to starting any data acquisition by calling Picam_CommitParameters().
- device.
  - Setting a device parameter automatically sets the corresponding model parameter to the same value.

Related Structures
For additional information, refer to the following ROI structures:

- PicamRoi;
- PicamRois.

Related APIs
For additional information, refer to the following related APIs:

- Picam_CommitParameters()
5.4.18  *Picam_DestroyPulses()*  

**Description**  
*Picam_DestroyPulses()* releases memory that has been allocated by PI Cam for use by *pulses*.  
If *pulses* is null, calling *Picam_DestroyPulses()* has no effect.  

**Syntax**  
The syntax for *Picam_DestroyPulses()* is:  

```c
PICAM_API Picam_DestroyPulses(
        const PicamPulse* pulses);
```

**Input Parameters**  
Input parameters for *Picam_DestroyPulses()* are:  

```c
        pulses:  Pointer to array memory that is to be released.
```

**Output Parameters**  
There are no output parameters associated with *Picam_DestroyPulses()*.
5.4.1.9 Picam_GetParameterPulseValue()

Description
Picam_GetParameterPulseValue() returns the current value for a specified Pulse parameter.

Syntax
The syntax for Picam_GetParameterPulseValue() is:

```c
PICAM_API Picam_GetParameterPulseValue(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamPulse** value);
```

Input Parameters
Input parameters for Picam_GetParameterPulseValue() are:

- `camera`: Handle for the camera for which the specified pulse parameter value is being requested.
- `parameter`: Specifies the Pulse parameter for which the current value is to be returned.
  Valid parameters are those of type PicamValueType_Pulse.

Output Parameters
Output parameters for Picam_GetParameterPulseValue() are:

- `value`: Pointer to the memory location where the value of the specified Pulse parameter has been stored.

  **NOTE:** This memory is allocated by PICam and must be released by calling Picam_DestroyPulses()

Advanced API Usage
When used in conjunction with Advanced APIs, camera may be a handle to either the:

- device, or
- model.

Stored values for any specific parameter are not necessarily the same for the device and model instances.

Related Structures
For additional information, refer to the following Pulse structure:

- PicamPulse

Related APIs
For additional information, refer to the following related APIs:

- Picam_DestroyPulses()
5.4.10 Picam\_CanSetParameterPulseValue()

**Description**

`Picam\_CanSetParameterPulseValue()` determines if a value is valid for a specified Pulse parameter.

**Syntax**

The syntax for `Picam\_CanSetParameterPulseValue()` is:

```c
PICAM\_API Picam\_CanSetParameterPulseValue(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamPulse* value,
    pibln* settable);
```

**Input Parameters**

Input parameters for `Picam\_CanSetParameterPulseValue()` are:

- `camera`: Handle for the camera for which the value/parameter combination is being validated.
- `parameter`: Specifies the Pulse parameter.
- `value`: The value that is to be tested.

**Output Parameters**

Output parameters for `Picam\_CanSetParameterPulseValue()` are:

- `settable`: Pointer to the test results. Indicates if the value is valid for the specified Pulse parameter. Valid values are:
  - `TRUE`: Indicates that the value is valid for the specified Pulse parameter.
  - `FALSE`: Indicates that the value is not valid for the specified Pulse parameter.

**Advanced API Usage**

When used in conjunction with Advanced APIs, `camera` may be a handle to either:

- `device`, or
- `model`.

**Related Structures**

For additional information, refer to the following Pulse constraint structure:

- `PicamPulseConstraint`. 
5.4.1.11 Picam_SetParameterPulseValue()

Description
Picam_SetParameterPulseValue() configures a Pulse parameter to a specified value during camera setup.

Syntax
The syntax for Picam_SetParameterPulseValue() is:

```c
PICAM_API Picam_SetParameterPulseValue(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamPulse* value);
```

Input Parameters
Input parameters for Picam_SetParameterPulseValue() are:

- **camera**: Handle for the camera being configured.
- **parameter**: Specifies the Pulse parameter that is to be configured. Valid parameters are those of type PicamValueType_Pulse.
- **value**: The value to which the Pulse parameter is to be set.

Output Parameters
There are no output parameters associated with Picam_SetParameterPulseValue().

Advanced API Usage
When used in conjunction with Advanced APIs, camera may be a handle to either the:

- **model**;
  - The model parameter may be set independently from the corresponding device parameter. However, doing so requires that all parameters be committed to the device prior to starting any data acquisition by calling Picam_CommitParameters().
- **device**.
  - Setting a device parameter automatically sets the corresponding model parameter to the same value.

Related Structures
For additional information, refer to the following Pulse structure:

- PicamPulse.

Related APIs
For additional information, refer to the following related APIs:

- Picam_CommitParameters()
5.4.1.12 Picam_DestroyModulations()

Description
Picam_DestroyModulations() releases memory that has been allocated by PICam for use by modulations.

If modulations is null, calling Picam_DestroyModulations() has no effect.

Syntax
The syntax for Picam_DestroyModulations() is:

    PICAM_API Picam_DestroyModulations(
        const PicamModulations* modulations);

Input Parameters
Input parameters for Picam_DestroyModulations() are:

    modulations: Pointer to array memory that is to be released.

Output Parameters
There are no output parameters associated with Picam_DestroyModulations().
5.4.1.13 Picam_GetParameterModulationsValue()

Description
Picam_GetParameterModulationsValue() returns the current value for a specified
intensifier modulation sequence parameter.

Syntax
The syntax for Picam_GetParameterModulationsValue() is:

    PICAM_API Picam_GetParameterModulationsValue(
        PicamHandle camera,
        PicamParameter parameter,
        const PicamModulations** value);

Input Parameters
Input parameters for Picam_GetParameterModulationsValue() are:

    camera: Handle for the camera for which the intensifier modulation sequence
            parameter value is being requested.
    parameter: Specifies the intensifier modulation sequence parameter for which the
                current value is to be returned.
                Valid parameters are those of type PicamValueType_Modulations.

Output Parameters
Output parameters for Picam_GetParameterModulationsValue() are:

    value: Pointer to the memory location in which the value of the specified
           intensifier modulation sequence parameter is stored.
    NOTE: This memory is allocated by PICam and must be released by
          calling Picam_DestroyModulations()

Advanced API Usage
When used in conjunction with Advanced APIs, camera may be a handle to either the:

    • device, or
    • model.

Stored values for any specific parameter are not necessarily the same for the device
and model instances.

Related Structures
For additional information, refer to the following intensifier modulation sequence
structures:

    • PicamModulation;
    • PicamModulations.

Related APIs
For additional information, refer to the following related APIs:

    • Picam_DestroyModulations().
5.4.1.14 Picam_CanSetParameterModulationsValue()

Description
Picam_CanSetParameterModulationsValue() determines if a value is valid for a specified intensifier modulation sequence parameter.

Syntax
The syntax for Picam_CanSetParameterModulationsValue() is:

```c
PICAM_API Picam_CanSetParameterModulationsValue(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamModulations* value,
    pibln* settable);
```

Input Parameters
Input parameters for Picam_CanSetParameterModulationsValue() are:
- **camera**: Handle for the camera for which the value/parameter combination is being validated.
- **parameter**: Specifies the intensifier modulation sequence parameter.
- **value**: The value that is to be tested.

Output Parameters
Output parameters for Picam_CanSetParameterModulationsValue() are:
- **settable**: Pointer to the test results. Indicates if the value is valid for the specified intensifier modulation sequence parameter. Valid values are:
  - **TRUE**: Indicates that the value is valid for the specified intensifier modulation sequence parameter.
  - **FALSE**: Indicates that the value is not valid for the specified intensifier modulation sequence parameter.

Advanced API Usage
When used in conjunction with Advanced APIs, **camera** may be a handle to either the:
- **device**, or
- **model**.

Related Structures
For additional information, refer to the following intensifier modulation sequence structures:
- **PicamModulation**;
- **PicamModulations**.
5.4.1.15 Picam_SetParameterModulationsValue()

Description
Picam_SetParameterModulationsValue() configures an intensifier modulation sequence parameter to a specified value during camera setup.

Syntax
The syntax for Picam_SetParameterModulationsValue() is:

```c
PICAM_API Picam_SetParameterModulationsValue(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamModulations* value);
```

Input Parameters
Input parameters for Picam_SetParameterModulationsValue() are:

- camera: Handle for the camera being configured.
- parameter: Specifies the Pulse parameter that is to be configured. Valid parameters are those of type PicamValueType_Pulse.
- value: The value to which the intensifier modulation sequence parameter is to be set.

Output Parameters
There are no output parameters associated with Picam_SetParameterModulationsValue().

Advanced API Usage
When used in conjunction with Advanced APIs, camera may be a handle to either the:

- model;
  The model parameter may be set independently from the corresponding device parameter. However, doing so requires that all parameters be committed to the device prior to starting any data acquisition by calling Picam_CommitParameters().
- device.
  Setting a device parameter automatically sets the corresponding model parameter to the same value.

Related Structures
For additional information, refer to the following intensifier modulation sequence structures:

- PicamModulation;
- PicamModulations.

Related APIs
For additional information, refer to the following related APIs:

- Picam_CommitParameters()
5.4.1.16 Picam_GetParameterLargeIntegerDefaultValue()

Description
Picam_GetParameterLargeIntegerDefaultValue() returns the large integer default value for a specified parameter.

Syntax
The syntax for Picam_GetParameterLargeIntegerDefaultValue() is:

```
PICAM_API Picam_GetParameterLargeIntegerDefaultValue(
    PicamHandle camera,
    PicamParameter parameter,
    pi64s* value);
```

Input Parameters
Input parameters for Picam_GetParameterLargeIntegerDefaultValue() are:

- `camera`: Handle for the camera for which the default parameter value is being requested.
- `parameter`: Specifies the parameter for which the default value is to be returned. Valid parameter are those of type PicamValueType_LargeInteger.

Output Parameters
Output parameters for Picam_GetParameterLargeIntegerDefaultValue() are:

- `value`: Pointer to the memory location in which the large integer default value for the specified parameter has been stored.

Advanced API Usage
When used in conjunction with Advanced APIs, `camera` may be a handle to either the:

- device, or
- model.

Both the device and model share the same default value.
5.4.1.17 Picam_GetParameterRoisPointDefaultValue()

Description
Picam_GetParameterRoisPointDefaultValue() returns the default value for a specified Rois parameter.

Syntax
The syntax for Picam_GetParameterRoisPointDefaultValue() is:

```c
PICAM_API Picam_GetParameterRoisDefaultValue(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamRois** value);
```

Input Parameters
Input parameters for Picam_GetParameterRoisPointDefaultValue() are:

- **camera**: Handle for the camera for which the default parameter value is being requested.
- **parameter**: Specifies the Rois parameter for which the default value is to be returned.
  Valid parameters are those of type PicamValueType_Rois.

Output Parameters
Output parameters for Picam_GetParameterRoisPointDefaultValue() are:

- **value**: Pointer to the memory location in which the default value for the specified Rois parameter has been stored.

  NOTE: This memory is allocated by PICam and must be released by calling Picam_DestroyRois().

Advanced API Usage
When used in conjunction with Advanced APIs, camera may be a handle to either the:
- device, or
- model.

Both the device and model share the same default value.

Related Structures
For additional information, refer to the following ROI structures:

- PicamRoi;
- PicamRois.

Related APIs
For additional information, refer to the following related APIs:

- Picam_DestroyRois().
5.4.1.18 Picam_GetParameterPulseDefaultValue()

Description
Picam_GetParameterPulseDefaultValue() returns the default value for a specified Pulse parameter.

Syntax
The syntax for Picam_GetParameterPulseDefaultValue() is:

```c
PICAM_API Picam_GetParameterPulseDefaultValue(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamPulse** value);
```

Input Parameters
Input parameters for Picam_GetParameterPulseDefaultValue() are:

- **camera**: Handle for the camera for which the default parameter value is being requested.
- **parameter**: Specifies the Pulse parameter for which the default value is to be returned. Valid parameters are those of type PicamValueType_Pulse.

Output Parameters
Output parameters for Picam_GetParameterPulseDefaultValue() are:

- **value**: Pointer to the memory location in which the default value for the specified Pulse parameter has been stored.

**NOTE**: This memory is allocated by PICam and must be released by calling Picam_DestroyPulses().

Advanced API Usage
When used in conjunction with Advanced APIs, `camera` may be a handle to either the:

- device, or
- model.

Both the `device` and `model` share the same default value.

Related Structures
For additional information, refer to the following Pulse structure:

- PicamPulse.

Related APIs
For additional information, refer to the following related APIs:

- Picam_DestroyPulses().
5.4.1.19 Picam_GetParameterModulationsDefaultValue()

Description
Picam_GetParameterModulationsDefaultValue() returns the default value for a specified intensifier modulation sequence parameter.

Syntax
The syntax for Picam_GetParameterModulationsDefaultValue() is:

```c
PICAM_API Picam_GetParameterModulationsDefaultValue(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamModulations** value);
```

Input Parameters
Input parameters for Picam_GetParameterModulationsDefaultValue() are:

- **camera**: Handle for the camera for which the default parameter value is being requested.
- **parameter**: Specifies the intensifier modulation sequence parameter for which the default value is to be returned. Valid parameters are those of type PicamValueType_Modulations.

Output Parameters
Output parameters for Picam_GetParameterModulationsDefaultValue() are:

- **value**: Pointer to the memory location in which the default value for the specified intensifier modulation sequence parameter has been stored. **NOTE**: This memory is allocated by PICam and must be released by calling Picam_DestroyModulations().

Advanced API Usage
When used in conjunction with Advanced APIs, camera may be a handle to either the:

- device, or
- model.

Both the device and model share the same default value.

Related Structures
For additional information, refer to the following intensifier modulation sequence structures:

- PicamModulation;
- PicamModulations.

Related APIs
For additional information, refer to the following related APIs:

- Picam_DestroyModulations().
5.4.1.20 Picam_SetParameterIntegerValueOnline()

Description
Picam_SetParameterIntegerValueOnline() configures the specified parameter with an integer value during data acquisition.

NOTE: The specified parameter must be capable of being configured during data acquisition. Refer to Picam_CanSetParameterOnline() for additional information.

Syntax
The syntax for Picam_SetParameterIntegerValueOnline() is:

```c
PICAM_API Picam_SetParameterIntegerValueOnline(
    PicamHandle camera,
    PicamParameter parameter,
    piint value);
```

Input Parameters
Input parameters for Picam_SetParameterIntegerValueOnline() are:

- `camera`: Handle for the camera being configured.
- `parameter`: Specifies the parameter that is to be configured. Valid parameters are those of type PicamValueType_Integer.
- `value`: The integer value with which the specified parameter is to be configured.

Advanced API Usage
When used in conjunction with Advanced APIs, camera may be a handle to either the:

- device, or
- model.

Picam_SetParameterIntegerValueOnline() effectively sets parameter on the camera device.

Output Parameters
There are no output parameters associated with Picam_SetParameterIntegerValueOnline().

Related APIs
For additional information, refer to the following related APIs:

- Picam_CommitParameters().
5.4.1.21 Picam_SetParameterFloatingPointValueOnline()

Description
Picam_SetParameterFloatingPointValueOnline() configures the specified parameter with a floating point value during data acquisition.

NOTE: The specified parameter must be capable of being configured during data acquisition. Refer to Picam_CanSetParameterOnline() for additional information.

Syntax
The syntax for Picam_SetParameterFloatingPointValueOnline() is:

```c
PICAM_API Picam_SetParameterFloatingPointValueOnline(
    PicamHandle camera,
    PicamParameter parameter,
    piflt value);
```

Input Parameters
Input parameters for Picam_SetParameterFloatingPointValueOnline() are:

- camera: Handle for the camera being configured.
- parameter: Specifies the parameter that is to be configured during data acquisition. Valid parameters are those of type PicamValueType_FloatingPoint.
- value: The floating point value with which the specified parameter is to be configured.

Advanced API Usage
When used in conjunction with Advanced APIs, camera may be a handle to either the:

- device, or
- model.

Picam_SetParameterFloatingPointValueOnline() effectively sets parameter on the camera device.

Output Parameters
There are no output parameters associated with Picam_SetParameterFloatingPointValueOnline().

Related APIs
For additional information, refer to the following related APIs:
- Picam_CommitParameters().
5.4.1.22 Picam_SetParameterPulseValueOnline()

Description
Picam_SetParameterPulseValueOnline() configures the specified Pulse parameter during data acquisition.

NOTE:
The specified parameter must be capable of being configured during data acquisition.
Refer to Picam_CanSetParameterOnline() for additional information.

Syntax
The syntax for Picam_SetParameterPulseValueOnline() is:

PICAM_API Picam_SetParameterPulseValueOnline(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamPulse* value);

Input Parameters
Input parameters for Picam_SetParameterPulseValueOnline() are:

camera: Handle for the camera being configured.
parameter: Specifies the Pulse parameter that is to be configured during data acquisition.
Value: Pointer to the memory location in which the desired configuration value is stored.

Output Parameters
There are no output parameters associated with Picam_SetParameterPulseValueOnline().

Advanced API Usage
When used in conjunction with Advanced APIs, camera may be a handle to either the:

• device, or
• model.

Picam_SetParameterPulseValueOnline() effectively sets parameter on the camera device.
5.4.2 Shared Camera/Accessory Parameter Value APIs

This section provides programming information for APIs used when working with shared camera/accessory parameter values.

5.4.2.1 Picam_GetParameterIntegerValue()

Description
Picam_GetParameterIntegerValue() returns the integer value for a specified parameter.

Syntax
The syntax for Picam_GetParameterIntegerValue() is:

```
PICAM_API Picam_GetParameterIntegerValue(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    piint* value);
```

Input Parameters
Input parameters for Picam_GetParameterIntegerValue() are:

- **camera_or_accessory**: Handle for the hardware for which the integer value is being requested.
- **parameter**: Specifies the parameter that is to be queried. Valid parameters are those of type:
  - PicamValueType_Integer;
  - PicamValueType_Boolean;
  - PicamValueType_Enumeration.

Output Parameters
Output parameters for Picam_GetParameterIntegerValue() are:

- **value**: Pointer to the integer value of the specified parameter.

Advanced API Usage
When used in conjunction with Advanced APIs, if camera_or_accessory is a camera handle, it may be a handle to either the:

- model, or
- device.

Stored values for any specific parameter are not necessarily the same for the device and model instances.
5.4.2.2 Picam_CanSetParameterIntegerValue()

Description
Picam_CanSetParameterIntegerValue() determines if an integer value is valid for a specified parameter.

Syntax
The syntax for Picam_CanSetParameterIntegerValue() is:

```c
PICAM_API Picam_CanSetParameterIntegerValue(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    printf value,
    pibln* settable);
```

Input Parameters
Input parameters for Picam_CanSetParameterIntegerValue() are:

- `camera_or_accessory`: Handle for the hardware for which the value/parameter combination is being validated.
- `parameter`: Specifies the parameter which is to be tested.
- `value`: The integer value that is to be tested.

Output Parameters
Output parameters for Picam_CanSetParameterIntegerValue() are:

- `settable`: Pointer to the test results. Indicates if the integer value is a valid value for the specified parameter.
  Valid values are:
  - `TRUE`: Indicates that the integer value is a valid value for the specified parameter.
  - `FALSE`: Indicates that the integer value is an invalid value for the specified parameter.

Advanced API Usage
When used in conjunction with Advanced APIs, if `camera_or_accessory` is a camera handle, it may be a handle to either the:

- `model`, or
- `device`. 
5.4.2.3 Picam_SetParameterIntegerValue()

Description

Picam_SetParameterIntegerValue() sets a parameter to a specified integer value during hardware setup.

Syntax

The syntax for Picam_SetParameterIntegerValue() is:

```c
PICAM_API Picam_SetParameterIntegerValue(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    piint value);
```

Input Parameters

Input parameters for Picam_SetParameterIntegerValue() are:

- `camera_or_accessory`: Handle for the hardware being configured.
- `parameter`: Specifies the parameter that is to be set with an integer value.
  Valid parameters are those of type:
  - PicamValueType_Integer;
  - PicamValueType_Boolean;
  - PicamValueType_Enumeration.
- `value`: The integer value to which the parameter is to be set.

Output Parameters

There are no output parameters associated with Picam_SetParameterIntegerValue().

Advanced API Usage

When used in conjunction with Advanced APIs, if `camera_or_accessory` is a camera handle, it may be a handle to either the:

- model;
  The model parameter may be set independently from the corresponding device parameter. However, doing so requires that all parameters be committed to the device prior to starting any data acquisition by calling Picam_CommitParameters().
- device.
  Setting a device parameter automatically sets the corresponding model parameter to the same value.

Related APIs

For additional information, refer to the following related APIs:

- Picam_CommitParameters().
5.4.2.4 Picam_GetParameterFloatingPointValue()

Description
Picam_GetParameterFloatingPointValue() returns the current floating point value for a specified parameter.

Syntax
The syntax for Picam_GetParameterFloatingPointValue() is:

```c
PICAM_API Picam_GetParameterFloatingValue(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    piflt* value);
```

Input Parameters
Input parameters for Picam_GetParameterFloatingPointValue() are:

- `camera_or_accessory`: Handle for the hardware for which the floating point value is being requested.
- `parameter`: Specifies the parameter that is to be queried. Valid parameters are those of type PicamValueType_FloatingPoint.

Output Parameters
Output parameters for Picam_GetParameterFloatingPointValue() are:

- `value`: Pointer to the floating point value of the specified parameter.

Advanced API Usage
When used in conjunction with Advanced APIs, if `camera_or_accessory` is a camera handle, it may be a handle to either the:

- model, or
- device.

Stored values for any specific parameter are not necessarily the same for the device and model instances.
5.4.2.5 *Picam_CanSetParameterFloatingPointValue()*

**Description**

*Picam_CanSetParameterFloatingPointValue()* determines if a floating point value is valid for a specified parameter.

**Syntax**

The syntax for *Picam_CanSetParameterFloatingPointValue()* is:

```c
PICAM_API Picam_CanSetParameterFloatingValue(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    piflt value,
    pibln* settable);
```

**Input Parameters**

Input parameters for *Picam_CanSetParameterFloatingPointValue()* are:

- `camera_or_accessory`: Handle for the hardware for which the value/parameter combination is being validated.
- `parameter`: Specifies the parameter which is to be tested.
- `value`: The floating point value that is to be tested.

**Output Parameters**

Output parameters for *Picam_CanSetParameterFloatingPointValue()* are:

- `settable`: Pointer to the test results. Indicates if the floating point value is a valid value for the specified parameter. Valid values are:
  - `TRUE`: Indicates that the floating point value is a valid value for the specified parameter.
  - `FALSE`: Indicates that the floating point value is an invalid value for the specified parameter.

**Advanced API Usage**

When used in conjunction with Advanced APIs, if `camera_or_accessory` is a camera handle, it may be a handle to either the:

- `model`, or
- `device`. 
5.4.2.6 Picam_SetParameterFloatingPointValue()

Description
Picam_SetParameterFloatingPointValue() sets a parameter to a specified floating point value during hardware setup.

Syntax
The syntax for Picam_SetParameterFloatingPointValue() is:

```c
PICAM_API Picam_SetParameterFloatingValue(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    piflt value);
```

Input Parameters
Input parameters for Picam_SetParameterFloatingPointValue() are:

- camera_or_accessory: Handle for the hardware being configured.
- parameter: Specifies the parameter that is to be set with a floating point value. Valid parameters are those of type PicamValueType_FloatingPoint.
- value: The floating point value to which the parameter is to be set.

Output Parameters
There are no output parameters associated with Picam_SetParameterFloatingPointValue().

Advanced API Usage
When used in conjunction with Advanced APIs, if camera_or_accessory is a camera handle, it may be a handle to either the:

- model;
  The model parameter may be set independently from the corresponding device parameter. However, doing so requires that all parameters be committed to the device prior to starting any data acquisition by calling Picam_CommitParameters().
- device.
  Setting a device parameter automatically sets the corresponding model parameter to the same value.

Related APIs
For additional information, refer to the following related APIs:

- Picam_CommitParameters()
5.4.2.7 Picam_GetParameterIntegerDefaultValue()

Description
Picam_GetParameterIntegerDefaultValue() returns the integer default value for a specified parameter.

Syntax
The syntax for Picam_GetParameterIntegerDefaultValue() is:

```c
PICAM_API Picam_GetParameterIntegerDefaultValue(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    piint* value);
```

Input Parameters
Input parameters for Picam_GetParameterIntegerDefaultValue() are:

- `camera_or_accessory`: Handle for the hardware for which the default parameter value is being requested.
- `parameter`: Specifies the parameter for which the integer default value is to be returned.
  Valid parameters are those of type:
  - `PicamValueType_Integer`;
  - `PicamValueType_Boolean`;
  - `PicamValueType_Enumeration`.

Output Parameters
Output parameters for Picam_GetParameterIntegerDefaultValue() are:

- `value`: Pointer to the memory location in which the integer default value for the specified parameter has been stored.

Advanced API Usage
When used in conjunction with Advanced APIs, if `camera_or_accessory` is a camera handle, it may be a handle to either the:

- device, or
- model.

Both the device and model share the same default value.
5.4.2.8 Picam_GetParameterFloatingPointDefaultValue()

Description
Picam_GetParameterFloatingPointDefaultValue() returns the floating point default value for a specified parameter.

Syntax
The syntax for Picam_GetParameterFloatingPointDefaultValue() is:

```c
PICAM_API Picam_GetParameterFloatingPointDefaultValue(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    piflt* value);
```

Input Parameters
Input parameters for Picam_GetParameterFloatingPointDefaultValue() are:

- `camera_or_accessory`: Handle for the camera for which the default parameter value is being requested.
- `parameter`: Specifies the parameter for which the default value is to be returned. Valid parameters are those of type `PicamValue type_FloatingPoint`.

Output Parameters
Output parameters for Picam_GetParameterFloatingPointDefaultValue() are:

- `value`: Pointer to the memory location in which the floating point default value for the specified parameter has been stored.

Advanced API Usage
When used in conjunction with Advanced APIs, if `camera_or_accessory` is a camera handle, it may be a handle to either the:

- device, or
- model.

Both the device and model share the same default value.
5.4.2.9 Picam_RestoreParametersToDefaultValues()

Description
Picam_RestoreParametersToDefaultValues() will set all read/write parameters to default values.

Syntax
The syntax for Picam_RestoreParametersToDefaultValues() is:

```c
PICAM_API Picam_RestoreParametersToDefaultValues(
    PicamHandle camera_or_accessory);
```

Input Parameters
Input parameters for Picam_RestoreParametersToDefaultValues() are:

- camera_or_accessory: Handle for the hardware for which parameters are to be restored.

Output Parameters
There are no output parameters associated with Picam_RestoreParametersToDefaultValues()

Advanced API Usage
When used in conjunction with Advanced APIs, if camera_or_accessory is a camera handle, it may be a handle to either the:

- device,
- model.
5.4.2.10 Picam_CanSetParameterOnline()

Description
Picam_CanSetParameterOnline() determines if the specified parameter can be configured during data acquisition.

Syntax
The syntax for Picam_CanSetParameterOnline() is:

```c
PICAM_API Picam_CanSetParameterOnline(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    pibln* onlineable);
```

Input Parameters
Input parameters for Picam_CanSetParameterOnline() are:

- `camera_or_accessory`: Handle for the hardware under test.
- `parameter`: Specifies the parameter for which the ability to be configured during data acquisition is to be determined.

Output Parameters
Output parameters for Picam_CanSetParameterOnline() are:

- `onlineable`: Pointer to the test results. Indicates if the specified parameter value can be set during data acquisition with this hardware. Valid values are:
  - `TRUE`: Indicates that the specified parameter can be configured during data acquisition.
  - `FALSE`: Indicates that the specified parameter cannot be configured during data acquisition.

Advanced API Usage
When used in conjunction with Advanced APIs, if `camera_or_accessory` is a camera handle, it may be a handle to either the:

- device, or
- model.
5.4.2.11 Picam_CanReadParameter()

Description
Picam_CanReadParameter() determines if a parameter value can be read directly from hardware connected to the system.

Syntax
The syntax for Picam_CanReadParameter() is:

```c
PICAM_API Picam_CanReadParameter(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    pibln* readable);
```

Input Parameters
Input parameters for Picam_CanReadParameter() are:

- `camera_or_accessory`: Handle for the hardware under test.
- `parameter`: Specifies the parameter for which the ability to read its value directly from the hardware is to be determined.

Output Parameters
Output parameters for Picam_CanReadParameter() are:

- `readable`: Pointer to the test results. Indicates if the specified parameter value can be read directly from the hardware. Valid values are:
  - `TRUE`: Indicates that the value for the specified parameter can be read from the hardware.
  - `FALSE`: Indicates that the value for the specified parameter cannot be read from the hardware.

Advanced API Usage
When used in conjunction with Advanced APIs, if `camera_or_accessory` is a camera handle, it may be a handle to either the:

- device, or
- model.
5.4.2.12 Picam_ReadParameterIntegerValue()

Description
Picam_ReadParameterIntegerValue() returns the integer value for a specified parameter as read directly from hardware connected to the system.

**NOTE:**
The specified parameter must be capable of being read directly from the hardware.
Refer to Picam_CanReadParameter() for additional information.

Syntax
The syntax for Picam_ReadParameterIntegerValue() is:

```c
PICAM_API Picam_ReadParameterIntegerValue(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    int* value);
```

Input Parameters
Input parameters for Picam_ReadParameterIntegerValue() are:

- **camera_or_accessory:** Handle for the camera under test.
- **parameter:** Specifies the parameter that is to have its value read from hardware.

**NOTE:** The specified parameter must be capable of being read directly from hardware. Refer to Picam_CanReadParameter() for additional information.

Valid parameters are those of type:
- PicamValueType_Integer;
- PicamValueType_Boolean;
- PicamValueType_Enumeration.

Output Parameters
Output parameters for Picam_ReadParameterIntegerValue() are:

- **value:** Pointer to the memory location in which the parameter value is stored.

Advanced API Usage
When used in conjunction with Advanced APIs, if camera_or_accessory is a camera handle, it may be a handle to either the:
- device, or
- model.
Picam_ReadParameterIntegerValue() effectively gets parameter from the hardware.

Related APIs
For additional information, refer to the following related APIs:

- Picam_CanReadParameter().
5.4.2.13 Picam_ReadParameterFloatingPointValue()

Description
Picam_ReadParameterFloatingPointValue() returns the floating point value for a specified parameter as read directly from hardware connected to the system.

NOTE: The specified parameter must be capable of being read directly from the hardware. Refer to Picam_CanReadParameter() for additional information.

Syntax
The syntax for Picam_ReadParameterFloatingPointValue() is:

```
PICAM_API Picam_ReadParameterFloatingPointValue(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    piflt* value);
```

Input Parameters
Input parameters for Picam_ReadParameterFloatingPointValue() are:

- `camera_or_accessory`: Handle for the hardware under test.
- `parameter`: Specifies the parameter that is to have its value read from hardware.

NOTE: The specified parameter must be capable of being read directly from hardware. Refer to Picam_CanReadParameter() for additional information.

Valid parameter are those of type:
- `PicamValueType_FloatingPoint`.

Output Parameters
Output parameters for Picam_ReadParameterFloatingPointValue() are:

- `value`: Pointer to the memory location in which the parameter value is stored.

Advanced API Usage
When used in conjunction with Advanced APIs, if `camera_or_accessory` is a camera handle, it may be a handle to either the:

- device, or
- model.

Picam_ReadParameterFloatingPointValue() effectively gets parameter from the hardware.

Related APIs
For additional information, refer to the following related APIs:

- Picam_CanReadParameter().
5.4.2.14 Picam_CanWaitForStatusParameter()

Description
Picam_CanWaitForStatusParameter() determines if a parameter is a waitable status.

Syntax
The syntax for Picam_CanWaitForStatusParameter() is:

```c
PICAM_API Picam_CanWaitForStatusParameter(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    pibln* waitable);
```

Input Parameters
Input parameters for Picam_CanWaitForStatusParameter() are:

- `camera_or_accessory`: Handle for the hardware under test.
- `parameter`: Specifies the parameter to check as a waitable status.

Output Parameters
Output parameters for Picam_CanWaitForStatusParameter() are:

- `waitable`: Pointer to the test results. Indicates if the specified parameter is a waitable status.
  Valid values are:
  - TRUE: Indicates that the parameter is a waitable status.
  - FALSE: Indicates that the parameter is not a waitable status.

Advanced API Usage
When used in conjunction with Advanced APIs, if `camera_or_accessory` is a camera handle, it may be a handle to either the:

- device, or
- model.
5.4.2.15 Picam_GetStatusParameterPurview()

**Description**

Picam_GetStatusParameterPurview() returns the scope of a waitable status.

**Syntax**

The syntax for Picam_GetStatusParameterPurview() is:

```c
PICAM_API Picam_GetStatusParameterPurview(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    const PicamStatusPurview** purview);
```

**Input Parameters**

Input parameters for Picam_GetStatusParameterPurview() are:

- `camera_or_accessory`: Handle for the hardware for which the status purview is being requested.
- `parameter`: Specifies the parameter whose status purview is being requested.

**NOTE:** The specified parameter must be a waitable status. Refer to Picam_CanWaitForStatusParameter() for additional information.

**Output Parameters**

Output parameters for Picam_GetStatusParameterPurview() are:

- `purview`: Pointer to the allocated status purview.

**NOTE:** This memory is allocated by PICam and must be released by calling Picam_DestroyStatusPurviews().

**Advanced API Usage**

When used in conjunction with Advanced APIs, if `camera_or_accessory` is a camera handle, it may be a handle to either the:

- device, or
- model.

**Related APIs**

For additional information, refer to the following related APIs:

- Picam_CanWaitForStatusParameter()
- Picam_DestroyStatusPurviews()
5.4.2.16 Picam_DestroyStatusPurviews()

Description
Picam_DestroyStatusPurviews() releases memory that has been allocated by PICam for use by the purviews_array.

If the purviews_array is null, calling Picam_DestroyStatusPurviews() has no effect.

Syntax
The syntax for Picam_DestroyStatusPurviews() is:

PICAM_API Picam_DestroyStatusPurviews(
    const PicamStatusPurview* purviews_array);

Input Parameters
Input parameters for Picam_DestroyStatusPurviews() are:

    purviews_array: Pointer to array memory that is to be released.

Output Parameters
There are no output parameters associated with Picam_DestroyStatusPurviews().
5.4.2.17  Picam_EstimateTimeToStatusParameterValue()

Description
Picam_EstimateTimeToStatusParameterValue() returns the estimated time, in milliseconds, for a particular status to be reached.

Syntax
The syntax for Picam_EstimateTimeToStatusParameterValue() is:

```c
PICAM_API Picam_EstimateTimeToStatusParameterValue(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    IntPtr value,
    IntPtr* estimated_time);
```

Input Parameters
Input parameters for Picam_EstimateTimeToStatusParameterValue() are:

- camera_or_accessory: Handle for the hardware whose time to status will be estimated.
- parameter: Specifies the parameter whose time to status will be estimated.
  
  **NOTE:** The specified parameter must be a waitable status.
  Refer to Picam_CanWaitForStatusParameter() for additional information.
- value: Specifies the status for which the time is to be estimated.
  
  **NOTE:** The specified value must be in the status purview.
  Refer to Picam_GetStatusParameterPurview() for additional information.

Output Parameters
Output parameters for Picam_EstimateTimeToStatusParameterValue() are:

- estimated_time: Pointer to the estimated time in milliseconds.
  
  **NOTE:** If the time cannot be estimated, -1 is returned.

Advanced API Usage
When used in conjunction with Advanced APIs, if camera_or_accessory is a camera handle, it may be a handle to either the:

- device,
- or
- model.

Related APIs
For additional information, refer to the following related APIs:

- Picam_CanWaitForStatusParameter()
- Picam_GetStatusParameterPurview()
5.4.2.18 Picam_WaitForStatusParameterValue()

Description
Picam_WaitForStatusParameterValue() waits for a particular status to be reached or until time_out milliseconds has elapsed. PicamError_TimeOutOccurred is returned if time_out has elapsed.

Syntax
The syntax for Picam_WaitForStatusParameterValue() is:

```
PICAM_API Picam_WaitForStatusParameterValue(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    puint value,
    puint time_out);
```

Input Parameters
Input parameters for Picam_WaitForStatusParameterValue() are:

- camera_or_accessory: Handle for the hardware whose status will be awaited.
- parameter: Specifies the parameter whose status will be awaited.
  
  **NOTE:** The specified parameter must be a waitable status. Refer to Picam_CanWaitForStatusParameter() for additional information.

- value: Specifies the status to await.
  
  **NOTE:** The specified value must be in the status purview. Refer to Picam_GetStatusParameterPurview() for additional information.

- time_out: Specifies the time to wait, in milliseconds.
  
  **NOTE:** Use –1 to wait indefinitely.

Output Parameters
There are no output parameters associated with Picam_WaitForStatusParameterValue().

Related APIs
For additional information, refer to the following related APIs:

- Picam_CanWaitForStatusParameter()
- Picam_GetStatusParameterPurview()
5.4.3 Shared Camera/Accessory Parameter Information APIs

This section provides programming information for APIs used to configure and retrieve shared camera and accessory parameter information.

5.4.3.1 Picam_DestroyParameters()

Description
Picam_DestroyParameters() releases memory that has been allocated by PICam for use by parameter_array.

If parameter_array is null, calling Picam_DestroyParameters() has no effect.

NOTE: parameter_array may be a single PicamParameter allocated by PICam.

Syntax
The syntax for Picam_DestroyParameters() is:

```
PICAM_API Picam_DestroyParameters(
    const PicamParameter* parameter_array);
```

Input Parameters
Input parameters for Picam_DestroyParameters() are:

parameter_array: Pointer to array memory that is to be released.

Output Parameters
There are no output parameters associated with Picam_DestroyParameters().

Related Structures
For additional information, refer to the following parameter structure:

- PicamParameter.
5.4.3.2 *Picam_GetParameters()*

**Description**

*Picam_GetParameters()* returns a list of parameters that are available for the specified hardware. The number of parameters is also returned.

**Syntax**

The syntax for *Picam_GetParameters()* is:

```c
PICAM_API Picam_GetParameters(
    PicamHandle camera_or_accessory,
    const PicamParameter** parameter_array,
    piint* parameter_count);
```

**Input Parameters**

Input parameters for *Picam_GetParameters()* are:

- `camera_or_accessory`: Handle for the hardware under test.

**Output Parameters**

Output parameters for *Picam_GetParameters()* are:

- `parameter_array`: Pointer to the allocated array in which the list of parameters associated with the specified hardware is stored.
  
  **NOTE:** This memory is allocated by PlCam and must be released by calling *Picam_DestroyParameters()*.

- `parameter_count`: Pointer to the memory location in which the number of available parameters associated with the specified hardware is stored.

**Advanced API Usage**

When used in conjunction with Advanced APIs, if `camera_or_accessory` is a camera handle, it may be a handle to either the:

- device, or
- model.

**Related APIs**

For additional information, refer to the following related APIs:

- *Picam_DestroyParameters()*.

**Related Structures**

For additional information, refer to the following parameter structure:

- *PicamParameter*.
5.4.3.3 Picam_DoesParameterExist()

Description
Picam_DoesParameterExist() determines if a specified parameter is available for the specified hardware.

Syntax
The syntax for Picam_DoesParameterExist() is:

```c
PICAM_API Picam_DoesParameterExist(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    pibln* exists);
```

Input Parameters
Input parameters for Picam_DoesParameterExist() are:
- camera_or_accessory: Handle for the hardware under test.
- parameter: Specifies the parameter for which availability is being determined.

Output Parameters
Output parameters for Picam_DoesParameterExist() are:
- exists: Pointer to the test results. Indicates if the specified parameter is available on the specified hardware. Valid values are:
  - TRUE
    Indicates that the specified parameter is available on the specified hardware.
  - FALSE
    Indicates that the specified parameter is not available on the specified hardware.

Advanced API Usage
When used in conjunction with Advanced APIs, if camera_or_accessory is a camera handle, it may be a handle to either the:
- device, or
- model.

Related Structures
For additional information, refer to the following parameter structure:
- PicamParameter.
5.4.3.4 Picam_IsParameterRelevant()

Description
Picam_IsParameterRelevant() determines if the value of a specified parameter is currently applicable for the specified hardware.

Syntax
The syntax for Picam_IsParameterRelevant() is:

```c
PICAM_API Picam_IsParameterRelevant(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    pibln* relevant);
```

Input Parameters
Input parameters for Picam_IsParameterRelevant() are:

- `camera_or_accessory`: Handle for the hardware under test.
- `parameter`: Specifies the parameter for which value applicability is being determined.

Output Parameters
Output parameters for Picam_IsParameterRelevant() are:

- `relevant`: Pointer to the test results. Indicates if the specified parameter value is currently applicable for the specified hardware. 
  Valid values are:
  - `TRUE`: Indicates that the specified parameter value is currently applicable for the specified hardware.
  - `FALSE`: Indicates that the specified parameter value is not currently applicable for the specified hardware.

Advanced API Usage
When used in conjunction with Advanced APIs, if `camera_or_accessory` is a camera handle, it may be a handle to either the:

- device,
- model.

Related Structures
For additional information, refer to the following parameter structure:

- PicamParameter.
5.4.3.5 Picam_GetParameterValueType()

Description
Picam_GetParameterValueType() returns the data type for a value stored within a specified parameter.

Syntax
The syntax for Picam_GetParameterValueType() is:

PICAM_API Picam_GetParameterValueType(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamValueType* type);

Input Parameters
Input parameters for Picam_GetParameterValueType() are:

camera_or_accessory: Handle for the hardware under test.
parameter: Specifies the parameter for which the data type of the stored value is being requested.

type: Pointer to the memory location in which the data type of the specified parameter’s value is stored.

Output Parameters
Output parameters for Picam_GetParameterValueType() are:

Advanced API Usage
When used in conjunction with Advanced APIs, if camera_or_accessory is a camera handle, it may be a handle to either the:

- device, or
- model.

Related Structures
For additional information, refer to the following parameter structure:

- PicamParameter;
- PicamValueType.
5.4.3.6  Picam_GetParameterEnumeratedType()

**Description**

`Picam_GetParameterEnumeratedType()` returns the enumeration type for a specified parameter.

**Syntax**

The syntax for `Picam_GetParameterEnumeratedType()` is:

```c
PICAM_API Picam_GetParameterEnumeratedType(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamEnumeratedType* type);
```

**Input Parameters**

Input parameters for `Picam_GetParameterEnumeratedType()` are:

- `camera_or_accessory`: Handle for the hardware under test.
- `parameter`: Specifies the parameter for which the enumeration type is being requested.

Valid parameters are those of type `PicamValueType_Enumeration`.

**Output Parameters**

Output parameters for `Picam_GetParameterEnumeratedType()` are:

- `type`: Pointer to the memory location in which the enumeration type of the specified parameter is stored.

**Advanced API Usage**

When used in conjunction with Advanced APIs, if `camera_or_accessory` is a camera handle, it may be a handle to either the:

- device, or
- model.

**Related Structures**

For additional information, refer to the following parameter structure:

- `PicamParameter`. 
5.4.3.7 Picam_GetParameterValueAccess()

Description
Picam_GetParameterValueAccess() returns the read/write permissions for the specified parameter.

Syntax
The syntax for Picam_GetParameterValueAccess() is:

```c
PICAM_API Picam_GetParameterValueAccess(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamValueAccess* access);
```

Input Parameters
Input parameters for Picam_GetParameterValueAccess() are:

- `camera_or_accessory`: Handle for the hardware under test.
- `parameter`: Specifies the parameter for which read/write permission is being requested.

Output Parameters
Output parameters for Picam_GetParameterValueAccess() are:

- `access`: Pointer to the memory location in which the read/write permission for the specified parameter is stored.

Advanced API Usage
When used in conjunction with Advanced APIs, if `camera_or_accessory` is a camera handle, it may be a handle to either the:

- device, or
- model.

Related Structures
For additional information, refer to the following parameter structure:

- PicamParameter.
5.4.3.8  *Picam_GetParameterConstraintType()*

**Description**

*Picam_GetParameterConstraintType()* returns the type of constraint placed on a specified parameter.

**Syntax**

The syntax for *Picam_GetParameterConstraintType()* is:

```c
PICAM_API Picam_GetParameterConstraintType(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamConstraintType* type);
```

**Input Parameters**

Input parameters for *Picam_GetParameterConstraintType()* are:

- `camera_or_accessory`: Handle for the hardware under test.
- `parameter`: Specifies the parameter for which constraint information is being requested.

**Output Parameters**

Output parameters for *Picam_GetParameterConstraintType()* are:

- `type`: Pointer to the memory location in which constraint information for the specified parameter is stored.

**Advanced API Usage**

When used in conjunction with Advanced APIs, if `camera_or_accessory` is a camera handle, may be a handle to either the:

- `device`, or
- `model`.

**Related Structures**

For additional information, refer to the following parameter structure:

- `PicamParameter`. 
5.4.4 Camera-Specific Parameter Constraints APIs

This section provides programming information for APIs used to configure camera-specific parameter constraints.

5.4.4.1 Picam_DestroyRoisConstraints()

Description
Picam_DestroyRoisConstraints() releases memory that has been allocated by PICam for use by constraint_array. If constraint_array is null, calling Picam_DestroyRoisConstraints() has no effect.

NOTE: constraint_array may be a single PicamRoisConstraint allocated by PICam.

Syntax
The syntax for Picam_DestroyRoisConstraints() is:

```
PICAM_API Picam_DestroyRoisConstraints(
    const PicamRoisConstraint* constraint_array);
```

Input Parameters
Input parameters for Picam_DestroyRoisConstraints() are:

- constraint_array: Pointer to array memory that is to be released.

Output Parameters
There are no output parameters associated with Picam_DestroyRoisConstraints().

Related Structures
For additional information, refer to the following parameter structure:

- PicamRoisConstraint.
5.4.4.2 Picam_GetParameterRoisConstraint()

Description
Picam_GetParameterRoisConstraint() returns Roi constraints for a specified constraint category and parameter combination.

Syntax
The syntax for Picam_GetParameterRoisConstraint() is:

```c
PICAM_API Picam_GetParameterRoisConstraint(
    PicamHandle camera,
    PicamParameter parameter,
    PicamConstraintCategory category,
    const PicamRoisConstraint** constraint);
```

Input Parameters
Input parameters for Picam_GetParameterRoisConstraint() are:

- camera: Handle for the camera for which constraint information is being returned.
- parameter: Specifies the parameter for which Roi constraint information is being requested.
- category: Specifies the constraint category for which Roi constraint information is being requested.

Output Parameters
Output parameters for Picam_GetParameterRoisConstraint() are:

- constraint: Pointer to the allocated array in which the Roi constraints for the specified constraint category and parameter combination are stored.

**NOTE:** This memory is allocated by PICam and must be released by calling Picam_DestroyRoisConstraints().

Advanced API Usage
When used in conjunction with Advanced APIs, camera may be a handle to either the:

- device, or
- model.

Related APIs
For additional information, refer to the following related APIs:

- Picam_DestroyRoisConstraints().

Related Structures
For additional information, refer to the following parameter structures:

- PicamParameter;
- PicamRoisConstraint.
5.4.4.3 Picam_DestroyPulseConstraints()

Description
Picam_DestroyPulseConstraints() releases memory that has been allocated by PICam for use by constraint_array.

If constraint_array is null, calling Picam_DestroyPulseConstraints() has no effect.

NOTE: constraint_array may be a single PicamPulseConstraint allocated by PICam.

Syntax
The syntax for Picam_DestroyPulseConstraints() is:

```c
PICAM_API Picam_DestroyPulseConstraints(
    const PicamPulseConstraint* constraint_array);
```

Input Parameters
Input parameters for Picam_DestroyPulseConstraints() are:

- constraint_array: Pointer to array memory that is to be released.

Output Parameters
There are no output parameters associated with Picam_DestroyPulseConstraints().

Related Structures
For additional information, refer to the following parameter structure:

- PicamPulseConstraint.
5.4.4.4 Picam_GetParameterPulseConstraint()

Description
Picam_GetParameterPulseConstraint() returns Pulse constraints for a specified constraint category and parameter combination.

Syntax
The syntax for Picam_GetParameterPulseConstraint() is:

```c
PICAM_API Picam_GetParameterPulseConstraint(
    PicamHandle camera,
    PicamParameter parameter,
    PicamConstraintCategory category,
    const PicamPulseConstraint** constraint);
```

Input Parameters
Input parameters for Picam_GetParameterPulseConstraint() are:

- `camera`: Handle for the camera for which constraint information is being returned.
- `parameter`: Specifies the parameter for which Pulse constraint information is being requested.
- `category`: Specifies the constraint category for which Pulse constraint information is being requested.

Output Parameters
Output parameters for Picam_GetParameterPulseConstraint() are:

- `constraint`: Pointer to the allocated array in which the Pulse constraints for the specified constraint category and parameter combination are stored.

**NOTE:** This memory is allocated by PICam and must be released by calling Picam_DestroyPulseConstraints().

Advanced API Usage
When used in conjunction with Advanced APIs, camera may be a handle to either the:

- device, or
- model.

Related APIs
For additional information, refer to the following related APIs:

- Picam_DestroyPulseConstraints().

Related Structures
For additional information, refer to the following parameter structures:

- PicamParameter;
- PicamPulseConstraint.
5.4.4.5 Picam_DestroyModulationsConstraints()

Description
Picam_DestroyModulationsConstraints() releases memory that has been allocated by PICam for use by constraint_array.

If constraint_array is null, calling Picam_DestroyModulationsConstraints() has no effect.

NOTE: constraint_array may be a single PicamModulationsConstraint allocated by PICam.

Syntax
The syntax for Picam_DestroyModulationsConstraints() is:

```c
PICAM_API Picam_DestroyModulationConstraints(
    const PicamModulationsConstraint* constraint_array);
```

Input Parameters
Input parameters for Picam_DestroyModulationsConstraints() are:

constraint_array: Pointer to array memory that is to be released.

Output Parameters
There are no output parameters associated with Picam_DestroyModulationsConstraints().

Related Structures
For additional information, refer to the following parameter structure:

- PicamModulationsConstraint.
5.4.4.6 Picam_GetParameterModulationsConstraint()

Description
Picam_GetParameterModulationsConstraint() returns intensifier modulation sequence constraints for a specified constraint category and parameter combination.

Syntax
The syntax for Picam_GetParameterModulationsConstraint() is:

```c
PICAM_API Picam_GetParameterModulationsConstraint(
    PicamHandle camera,
    PicamParameter parameter,
    PicamConstraintCategory category,
    const PicamModulationsConstraint** constraint);
```

Input Parameters
Input parameters for Picam_GetParameterModulationsConstraint() are:

- camera: Handle for the camera for which constraint information is being returned.
- parameter: Specifies the parameter for which intensifier modulation sequence constraint information is being requested.
- category: Specifies the constraint category for which intensifier modulation sequence constraint information is being requested.
- constraint: Pointer to the allocated array in which the intensifier modulation sequence constraints for the specified constraint category and parameter combination are stored.

**NOTE:** This memory is allocated by PICam and must be released by calling Picam_DestroyModulationsConstraints().

Output Parameters
Output parameters for Picam_GetParameterModulationsConstraint() are:

- constraint: Pointer to the allocated array in which the intensifier modulation sequence constraints for the specified constraint category and parameter combination are stored.

Advanced API Usage
When used in conjunction with Advanced APIs, camera may be a handle to either the:

- device, or
- model.

Related APIs
For additional information, refer to the following related APIs:

- Picam_DestroyModulationsConstraints().

Related Structures
For additional information, refer to the following parameter structures:

- PicamParameter;
- PicamModulationsConstraint.
5.4.5 Shared Camera/Accessory Parameter Constraints APIs

This section provides programming information for APIs used to configure shared camera and accessory parameter constraints.

5.4.5.1 Picam_DestroyCollectionConstraints()

Description

Picam_DestroyCollectionConstraints() releases memory that has been allocated by PICam for use by constraint_array.

If constraint_array is null, calling Picam_DestroyCollectionConstraints() has no effect.

**NOTE:**

constraint_array may be a single
PicamCollectionConstraint allocated by PICam.

Syntax

The syntax for Picam_DestroyCollectionConstraints() is:

```
PICAM_API Picam_DestroyCollectionConstraints(
    const PicamCollectionConstraint* constraint_array);
```

Input Parameters

Input parameters for Picam_DestroyCollectionConstraints() are:

- constraint_array: Pointer to array memory that is to be released.

Output Parameters

There are no output parameters associated with Picam_DestroyCollectionConstraints().

Related Structures

For additional information, refer to the following parameter structure:

- PicamCollectionConstraint.
5.4.5.2 Picam_GetParameterRangeConstraint()

Description
Picam_GetParameterRangeConstraint() returns range constraints for a specified constraint category and parameter combination.

Syntax
The syntax for Picam_GetParameterRangeConstraint() is:

   PICAM_API Picam_GetParameterRangeConstraint(
       PicamHandle camera,
       PicamParameter parameter,
       PicamConstraintCategory category,
       const PicamRangeConstraint** constraint);

Input Parameters
Input parameters for Picam_GetParameterRangeConstraint() are:

   camera: Handle for the camera for which range constraints are being returned.
   parameter: Specifies the parameter for which range constraint information is being requested.
   category: Specifies the constraint category for which range constraint information is being requested.

Output Parameters
Output parameters for Picam_GetParameterRangeConstraint() are:

   constraint: Pointer to the allocated array in which the range constraints for the specified constraint category and parameter combination are stored.

NOTE: This memory is allocated by PICam and must be released by calling Picam_DestroyRangeConstraints().

Advanced API Usage
When used in conjunction with Advanced APIs, camera may be a handle to either the:

   • device, or
   • model.

Related APIs
For additional information, refer to the following related APIs:

   • Picam_DestroyRangeConstraints().

Related Structures
For additional information, refer to the following parameter structures:

   • PicamParameter;
   • PicamRangeConstraint.
5.4.5.3 Picam_GetParameterCollectionConstraint()

Description

Picam_GetParameterCollectionConstraint() returns constraint information for a specified constraint category and parameter combination.

Syntax

The syntax for Picam_GetParameterCollectionConstraint() is:

```c
PICAM_API Picam_GetParameterCollectionConstraint(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamConstraintCategory category,
    const PicamCollectionConstraint** constraint);
```

Input Parameters

Input parameters for Picam_GetParameterCollectionConstraint() are:

- `camera_or_accessory`: Handle for the hardware for which constraint information is being returned.
- `parameter`: Specifies the parameter for which constraint information is being requested.
- `category`: Specifies the constraint category for which the list of constraints is being requested.

Output Parameters

Output parameters for Picam_GetParameterCollectionConstraint() are:

- `constraint`: Pointer to the allocated array in which the list of constraints available for the specified constraint category and parameter combination is stored.

  **NOTE:** This memory is allocated by PICam and must be released by calling Picam_DestroyCollectionConstraints().

Advanced API Usage

When used in conjunction with Advanced APIs, if `camera_or_accessory` is a camera handle, it may be a handle to either the:

- device, or
- model.

Related APIs

For additional information, refer to the following related APIs:

- Picam_DestroyCollectionConstraints().

Related Structures

For additional information, refer to the following parameter structures:

- PicamParameter;
- PicamCollectionConstraint.
5.4.5.4 **Picam_DestroyRangeConstraints()**

**Description**

`Picam_DestroyRangeConstraints()` releases memory that has been allocated by PICam for use by `constraint_array`. If `constraint_array` is null, calling `Picam_DestroyRangeConstraints()` has no effect.

**NOTE:** `constraint_array` may be a single `PicamRangeConstraint` allocated by PICam.

**Syntax**

The syntax for `Picam_DestroyRangeConstraints()` is:

```c
PICAM_API Picam_DestroyRangeConstraints(
    const PicamRangeConstraint* constraint_array);
```

**Input Parameters**

Input parameters for `Picam_DestroyRangeConstraints()` are:

- `constraint_array`: Pointer to array memory that is to be released.

**Output Parameters**

There are no output parameters associated with `Picam_DestroyRangeConstraints()`.

**Related Structures**

For additional information, refer to the following parameter structure:

- `PicamRangeConstraint`. 

5.4.6 Shared Camera/Accessory Parameter Commitment APIs

This section provides programming information about APIs used to commit parameter values.

NOTE: 
Accessories are always considered committed since any changes to their parameters are applied directly to the hardware.

5.4.6.1 Picam_AreParametersCommitted()

Description
Picam_AreParametersCommitted() determines if the parameter configuration changes have been applied to the specified hardware.

Syntax
The syntax for Picam_AreParametersCommitted() is:

```c
PICAM_API Picam_AreParametersCommitted(
    PicamHandle camera_or_accessory,
    pibln* committed);
```

Input Parameters
Input parameters for Picam_AreParametersCommitted() are:

- `camera_or_accessory`: Handle for the hardware for which parameter configuration status information is being determined.

Output Parameters
Output parameters for Picam_AreParametersCommitted() are:

- `committed`: Pointer to the test results. Indicates if parameter configuration changes have been committed to the specified hardware. Valid values are:
  - `TRUE`: Indicates that parameter configuration changes have been committed to the specified hardware.
  - `FALSE`: Indicates that parameter configuration changes have not been committed to the specified hardware.

Advanced API Usage
When used in conjunction with Advanced APIs, if `camera_or_accessory` is a camera handle, it may be a handle to either the:

- `device`, or
- `model`. 
5.4.6.2 *Picam_CommitParameters()*

**Description**

*Picam_CommitParameters()* validates parameter values and applies these valid values to the specified hardware during system setup and configuration.

- Any parameter that fails to satisfy its required constraint(s) is flagged as invalid and is stored within *failed_parameter_array*.
- The number of invalid parameters is stored in *failed_parameter_count*. If no invalid parameters are detected, this value is 0.

**Syntax**

The syntax for *Picam_CommitParameters()* is:

```c
PICAM_API Picam_CommitParameters(
    PicamHandle camera_or_accessory,
    const PicamParameter** failed_parameter_array,
    puint* failed_parameter_count);
```

**Input Parameters**

Input parameters for *Picam_CommitParameters()* are:

- camera_or_accessory: Handle for the hardware for which parameter values are being configured.

**Output Parameters**

Output parameters for *Picam_CommitParameters()* are:

- failed_parameter_array: Pointer to the allocated array in which the list of failed/invalid parameters is stored. If no invalid parameters are detected, this is a null object.
  
  **NOTE:** This memory is allocated by PICam and must be released by calling *Picam_DestroyParameters()*.

- failed_parameter_count: Pointer to the memory location in which the number of failed/invalid parameters is stored.

**Advanced API Usage**

When used in conjunction with Advanced APIs, if *camera_or_accessory* is a camera handle, it may be a handle to either the:

- device, or
- model.

*Picam_CommitParameters()* systematically configures device with (valid) parameter values that have been stored in model.
This page is intentionally blank.
Chapter 6: Data Acquisition APIs

Once system hardware has been configured and the parameters are committed, the system is ready to acquire data. Data can be acquired either synchronously or asynchronously.

By default, memory is allocated automatically to accommodate the data. This automatic memory is valid until the next acquisition or until the hardware is closed.

By default, the data are returned as follows:

- One frame of sensor data containing each region of interest (in the order defined);
- Followed by any metadata for that frame (timestamps followed by frame tracking, gate tracking delay, gate tracking width, and modulation tracking);
- Repeated for each frame in one readout;
- Possibly followed by any padding between readouts.

Configuring the hardware such that the total number of readouts is indeterminate will disable automatic data memory management:

- Basic:
  - Instruct the hardware to acquire more data than it can exactly acquire;
  - This is achieved by setting `PicamParameter_ReadoutCount` to a value greater than the value of `PicamParameter_ExactReadoutCount Maximum`.
  - Instruct the hardware to readout data non-destructively (for hardware that supports this feature)

- Advanced:
  - Instruct the hardware to acquire data indefinitely
  - This is achieved by setting `PicamParameter_ReadoutCount` to 0.
  - Also, setting a user-allocated buffer with `PicamAdvanced_SetAcquisitionBuffer()` will disable automatic data memory management.
6.1 Data Format

Acquired data is structured as shown in Figure 6-1.

**NOTE:**

All partitions are specified in bytes.

Figure 6-1: Data Format Diagram

The hardware acquires \( N \) Readouts, each separated by one Readout Stride. One Readout Stride is comprised of \( M \) frames, each separated by one Frame Stride followed by padding.

One Frame Stride is divided into:

- Frame Pixel Data
  Frame pixel data contains data for \( X \) Regions of Interest stored in the order in which each ROI was defined.
- Frame Metadata by a frame size.
  Frame Metadata contains any time stamps followed by:
  - Frame Tracking Number;
  - Gate Tracking Delay and Width;
  - Modulation Tracking Duration, Frequency, Phase, and Output Signal Frequency).

All formatting information is available as read-only data acquisition parameters.
6.2 Data Type Definitions

This section provides programming information about PICam data definitions.

6.2.1 Data Acquisition Enumerations

This section provides detailed information about the following data acquisition enumerations:

- PicamAcquisitionErrorsMask

6.2.1.1 PicamAcquisitionErrorsMask

Data Type

PicamAcquisitionErrorsMask is defined as enum.

Description

PicamAcquisitionErrorsMask is the set of acquisition error messages.

Enumerator Definitions

Refer to Table 6-1 for enumerator definitions.

Table 6-1: PicamAcquisitionErrorsMask Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamAcquisitionErrorsMask_CameraFaulted</td>
<td>The hardware has critically malfunctioned and is in need of service. Further acquisitions are not possible until the hardware has been serviced.</td>
</tr>
<tr>
<td>PicamAcquisitionErrorsMask_ConnectionLost</td>
<td>The hardware was disconnected.</td>
</tr>
<tr>
<td>PicamAcquisitionErrorsMask_DataLost</td>
<td>Data has been lost.</td>
</tr>
<tr>
<td>PicamAcquisitionErrorsMask_DataNotArriving</td>
<td>Data is no longer arriving from the hardware.</td>
</tr>
<tr>
<td>PicamAcquisitionErrorsMask_None</td>
<td>No errors have occurred.</td>
</tr>
<tr>
<td>PicamAcquisitionErrorsMask_ShutterOverheated</td>
<td>A connected shutter has overheated and is temporarily disabled. Further acquisitions are not possible until the shutter is no longer overheated.</td>
</tr>
</tbody>
</table>
6.3 Data Acquisition Data Structures

This section provides programming information about the following PICam data acquisition data structures:

- `PicamAvailableData`;
- `PicamAcquisitionStatus`.

6.3.1 PicamAvailableData

Description

`PicamAvailableData` represents newly acquired data.

Structure Definition

The structure definition for `PicamAvailableData` is:

```c
typedef struct PicamAvailableData
{
    void* initial_readout;
    pi64s readout_count;
} PicamAvailableData;
```

Variable Definitions

The variables required by `PicamAvailableData` are:

- `initial_readout`: Pointer to the start of the first available readout.
- `readout_count`: Indicates how many contiguous readouts are currently available.

6.3.2 PicamAcquisitionStatus

Description

`PicamAcquisitionStatus` reports various status information during data acquisition by the hardware.

Structure Definition

The structure definition for `PicamAcquisitionStatus` is:

```c
typedef struct PicamAcquisitionStatus
{
    pibln running;
    PicamAcquisitionErrorsMask errors;
    piflt readout_rate;
} PicamAcquisitionStatus;
```

Variable Definitions

The variables required by `PicamAcquisitionStatus` are:

- `running`: Indicates the data acquisition status. Valid values are:
  - `TRUE`: Indicates an acquisition is in progress.
  - `FALSE`: Indicates there is no current data acquisition in progress.
- `errors`: Contains any errors that have occurred.
- `readout_rate`: The rate of capture in readouts-per-second when acquiring more than one readout.
6.4 Programmers’ Reference for Acquisition Control APIs

This section provides programming information for the following acquisition control APIs:

- Picam_Acquire();
- Picam_StartAcquisition();
- Picam_StopAcquisition();
- Picam_IsAcquisitionRunning();
- Picam_WaitForAcquisitionUpdate().

6.4.1 Picam_Acquire()

Description

Picam_Acquire() performs a specified number of data readouts (specified by readout_count) and returns once the acquisition has been completed.

NOTE: This function cannot be called when hardware is configured for non-destructive readout. This is because the number of readouts acquired is no longer guaranteed to be fixed. As an example, changing the exposure time online will change the number of non-destructive readouts and therefore the total number of readouts acquired.

NOTE: Parameters must be committed prior to initiating data acquisition. Refer to Section 5.4.6.2, Picam_CommitParameters(), on page 177 for additional information.

Data acquisition is successful when:

- The delay between successive readouts does not exceed readout_time_out, and
- No errors have occurred.

Data acquisition is immediately halted when:

- The delay between successive readouts exceeds that specified by readout_time_out.
  The error message PicamError_TimeOutOccurred is returned.
- Any other error conditions are detected.
  Associated error messages are stored in the errors parameter.

Syntax

The syntax for Picam_Acquire() is:

```
PICAM_API Picam_Acquire(
    PicamHandle camera,
    pi64s readout_count,
    piint readout_time_out,
    PicamAvailableData* available,
    PicamAcquisitionErrorsMask* errors);
```

continued on next page
Input Parameters
Input parameters for Picam_Acquire() are:

- **camera**: Handle for the hardware from which data are to be acquired.
- **readout_count**: The number of readouts desired.
  Valid values are in the range: 
  \[ 1 \ldots \text{PicamParameter\_ExactReadoutCount\_Maximum} \]
  If this value becomes excessively large, this function may fail due to 
  a lack of sufficient memory.
- **readout_time_out**: The time, in mS, to wait between each successive readout.
  When specifying an infinite length of time, configure this parameter 
  to -1.

Output Parameters
Output parameters for Picam_Acquire() are:

- **available**: The output buffer used to store data that has been 
  successfully read out from the specified hardware.
  In the event of a data acquisition failure, this buffer may 
  contain little to no data.
  Data stored in this buffer is valid until:
  - The next acquisition cycle is initiated; or
  - The hardware is closed.
- **errors**: The parameter used to store any error messages that were 
  raised during data acquisition.

Advanced API Usage
When used in conjunction with Advanced APIs, data in the output buffer available is 
also invalidated when PicamAdvanced_SetAcquisitionBuffer() is called.
Picam_Acquire() is mutually exclusive with the use of an acquisition-updated callback.

Related APIs
For additional information, refer to the following related APIs:

- Picam_CommitParameters();
- PicamAdvanced_SetAcquisitionBuffer().
6.4.2 Picam_StartAcquisition()

Description
Picam_StartAcquisition() asynchronously initiates data acquisition and returns immediately.

NOTE: Parameters must be committed prior to initiating data acquisition. Refer to Section 5.4.6.2, Picam_CommitParameters(), on page 177 for information.

On success, data acquisition is running and continues until:

- The number of readouts specified by PicamParameter_ReadoutCount have been acquired;
- An error occurs which immediately halts data acquisition (refer to Section 6.4.1, Picam_Acquire(), on page 183 for additional information); or
- Picam_StopAcquisition() is called, in which case the acquisition will stop asynchronously (i.e., a short time later.)

NOTE: To determine the current data acquisition status, call Picam_WaitForAcquisitionUpdate().

Syntax
The syntax for Picam_StartAcquisition() is:

PICAM_API Picam_StartAcquisition(
    PicamHandle camera);

Input Parameters
Input parameters for Picam_StartAcquisition() are:

    camera: Handle for the hardware for which data acquisition is to be initiated.

Output Parameters
There are no output parameters associated with Picam_StartAcquisition().

Advanced API Usage
When used in conjunction with Advanced APIs, if PicamParameter_ReadoutCount = 0, the hardware will run continuously until Picam_StopAcquisition() is called to stop asynchronously.

Related APIs
For additional information, refer to the following related APIs:

- Picam_CommitParameters();
- Picam_Acquire();
- Picam_StopAcquisition();
- Picam_WaitForAcquisitionUpdate().
6.4.3 Picam_StopAcquisition()

Description
Picam_StopAcquisition() asynchronously halts an in-progress data acquisition and returns immediately.
On success, the data acquisition will stop running a short time later.

NOTE: [Advanced API Usage ONLY]
If PicamParameter_ReadoutCount = 0, the hardware will run continuously until Picam_StopAcquisition() has been called.

Syntax
The syntax for Picam_StopAcquisition() is:

PICAM_API Picam_StopAcquisition(
    PicamHandle camera);

Input Parameters
Input parameters for Picam_StopAcquisition() are:

camera: Handle for the hardware for which data acquisition is to be halted.

Output Parameters
There are no output parameters associated with Picam_StopAcquisition().

Related APIs
For additional information, refer to the following related APIs:

- Picam_StartAcquisition().
6.4.4 Picam_IsAcquisitionRunning()

Description
Picam_IsAcquisitionRunning() determines if there is an active data acquisition in process.

NOTE: This function should not be used to wait for data to arrive. Use Picam_WaitForAcquisitionUpdate() instead.

Syntax
The syntax for Picam_IsAcquisitionRunning() is:

```c
PICAM_API Picam_IsAcquisitionRunning(
    PicamHandle camera,
    plbln* running);
```

Input Parameters
Input parameters for Picam_IsAcquisitionRunning() are:

- **camera**: Handle for the hardware for which the data acquisition status is being determined.

Output Parameters
Output parameters for Picam_IsAcquisitionRunning() are:

- **running**: Indicates if there is an active data acquisition in progress. Valid values are:
  - **TRUE**: Indicates that there is an active data acquisition in process.
  - **FALSE**: Indicates that there is no active data acquisition in process.
6.4.5 Picam_WaitForAcquisitionUpdate()

Description

Picam_WaitForAcquisitionUpdate() is used in combination with Picam_StartAcquisition() and indicates when:

- New data are available; or
- The hardware’s status has changed.

Usage

Picam_WaitForAcquisitionUpdate() must be continuously called until PicamAcquisitionStatus.running returns FALSE. This is true regardless of any acquisition errors that may be returned or if Picam_StopAcquisition() has been called.

Any errors returned during data acquisition are stored in PicamAcquisitionStatuserrors and acquisition is immediately halted.

However, if new data is not available within the time specified by readout_time_out:

- The PicamError_TimeOutOccurred error is returned;
- Data acquisition will continue; and
- The contents of both the data buffer available as well as the status data structure are invalid.

Syntax

The syntax for Picam_WaitForAcquisitionUpdate() is:

```c
PICAM_API Picam_WaitForAcquisitionUpdate(
    PicamHandle camera,
    piint readout_time_out,
    PicamAvailableData* available,
    PicamAcquisitionStatus* status);
```

Input Parameters

Input parameters for Picam_WaitForAcquisitionUpdate() are:

- camera: Handle for the hardware from which data is being acquired.
- readout_time_out: The time, in mS, to wait between each successive readout. To configure this function to wait forever until a change occurs, set this parameter to -1.

Output Parameters

Output parameters for Picam_WaitForAcquisitionUpdate() are:

- available: The output buffer used to store newly acquired data from the specified hardware. Data stored in this buffer is valid until the next Picam_WaitForAcquisitionUpdate() call.
- status: Pointer to the PicamAcquisitionStatus data structure in which acquisition status information is stored.

continued on next page
**Advanced API Usage**

When used in conjunction with Advanced APIs, data in the output buffer available is also invalidated when `PicamAdvanced_SetAcquisitionBuffer()` is called (in the case of the last `Picam_WaitForAcquisitionUpdate()` call.)

`Picam_WaitForAcquisitionUpdate()` is mutually exclusive with the usage of an acquisition-updated callback.

**Related APIs**

For additional information, refer to the following related APIs:

- `Picam_StartAcquisition();`
- `PicamAdvanced_SetAcquisitionBuffer()`

**Related Structures**

For additional information, refer to the following related structure definition:

- `PicamAvailableData;`
- `PicamAcquisitionStatus.`
This page is intentionally blank.
Chapter 7: Advanced Function APIs

This chapter provides programming information about PICam advanced function APIs, including related data definitions and structures which are included in the picam_advanced.h file.

Figure 7-1 illustrates a block diagram of the PICam Advanced Function structure.

REFERENCES:

Refer to Section 2.6.1, Handles, for information about the handles used within PICam.

Figure 7-1: PICam Structure - Advanced

* Value and Constraint Types are parameter dependent.

** User State, Calibration, and Defect Map information are only available when the hardware is opened.
7.1 Data Type Definitions

This section provides programming information about the following PICam advanced data definitions:

- Shared Camera/Accessory Plug and Play Discovery Data Enumerations
  - PicamDiscoveryAction
- Shared Camera/Accessory Access Enumerations
  - PicamHandleType
- Shared Camera/Accessory Parameter Information Enumerations
  - PicamDynamicsMask
- Camera-Specific Data Acquisition Enumerations
  - PicamAcquisitionState
  - PicamAcquisitionStateErrorsMask

7.1.1 Shared Camera/Accessory Plug and Play Discovery Data Enumerations

This section provides programming information about shared camera/accessory plug and play discovery data enumerations.

7.1.1.1 PicamDiscoveryAction

Data Type

PicamDiscoveryAction is defined as enum.

Description

PicamDiscoveryAction is the set of discovery states available for hardware.

Enumerator Definitions

Refer to Table 7-1 for enumerator definitions.

Table 7-1: PicamDiscoveryAction Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamDiscoveryAction_Faulted</td>
<td>A hardware item has critically malfunctioned and is in need of service. Any acquisition in progress with this hardware will be stopped and further acquisition are not possible until the hardware has been serviced.</td>
</tr>
<tr>
<td>PicamDiscoveryAction_Found</td>
<td>Hardware is now available for use.</td>
</tr>
<tr>
<td>PicamDiscoveryAction_Lost</td>
<td>Hardware is no longer available for use.</td>
</tr>
</tbody>
</table>
7.1.2 Shared Camera/Accessory Access Enumerations

This section provides programming information about shared camera and accessory access data enumerations.

7.1.2.1 PicamHandleType

Data Type
PicamHandleType is defined as enum.

Description
PicamHandleType is the set of handle types.

Enumerator Definitions
Refer to Table 7-2 for enumerator definitions.

Table 7-2: PicamHandleType Enumerator Definitions

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamHandleType_Accessory</td>
<td>This handle refers to a hardware accessory.</td>
</tr>
<tr>
<td>PicamHandleType_CameraDevice</td>
<td>The handle refers to a camera device.</td>
</tr>
<tr>
<td>PicamHandleType_CameraModel</td>
<td>The handle refers to a camera model.</td>
</tr>
<tr>
<td>PicamHandleType_EMCalibration</td>
<td>The handle refers to a camera opened for EM calibration.</td>
</tr>
</tbody>
</table>
7.1.3  **Shared Camera/Accessory Parameter Information Enumerations**

This section provides programming information about shared camera and accessory parameter information data enumerations.

7.1.3.1  **PicamDynamicsMask**

**Data Type**

*PicamDynamicsMask* is defined as enum.

**Description**

*PicamDynamicsMask* is the set of descriptors for how parameters and their various attributes may or may not change.

**Enumerator Definitions**

Refer to Table 7-3 for enumerator definitions.

**Table 7-3:  **PicamDynamicsMask** Enumerator Definitions**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicamDynamicsMask_None</td>
<td>No parameter attributes may change.</td>
</tr>
<tr>
<td>PicamDynamicsMask_Value</td>
<td>The parameter value may change.</td>
</tr>
<tr>
<td>PicamDynamicsMask_ValueAccess</td>
<td>The parameter value access may change.</td>
</tr>
<tr>
<td>PicamDynamicsMask_IsRelevant</td>
<td>The parameter relevance may change.</td>
</tr>
<tr>
<td>PicamDynamicsMask_Constraint</td>
<td>The parameter dependent constraints may change.</td>
</tr>
</tbody>
</table>
7.1.4 Camera-Specific Data Acquisition Enumerations

This section provides programming information about camera-specific data acquisition enumerations.

7.1.4.1 \textit{PicamAcquisitionState}

\textbf{Data Type}

\textit{PicamAcquisitionState} is defined as \texttt{enum}.

\textbf{Description}

\textit{PicamAcquisitionState} is the set of camera states that can be detected during an acquisition.

\textbf{Enumerator Definitions}

Refer to Table 7-4 for enumerator definitions.

\begin{table}
\centering
\begin{tabular}{|c|p{10cm}|}
\hline
\textbf{Enumerator} & \textbf{Description} \\
\hline
\texttt{PicamAcquisitionState\_Readout\_Started} & The camera has begun to readout data. \\
\hline
\texttt{PicamAcquisitionState\_Readout\_Ended} & The camera has finished reading out data. \\
\hline
\end{tabular}
\caption{\textit{PicamAcquisitionState} Enumerator Definitions}
\end{table}

7.1.4.2 \textit{PicamAcquisitionStateErrorsMask}

\textbf{Data Type}

\textit{PicamAcquisitionStateErrorsMask} is defined as \texttt{enum}.

\textbf{Description}

\textit{PicamAcquisitionStateErrorsMask} is the set of errors that can occur while detecting acquisition states.

\textbf{Enumerator Definitions}

Refer to Table 7-5 for enumerator definitions.

\begin{table}
\centering
\begin{tabular}{|c|p{10cm}|}
\hline
\textbf{Enumerator} & \textbf{Description} \\
\hline
\texttt{PicamAcquisitionStateErrorsMask\_None} & No error has occurred. \\
\hline
\texttt{PicamAcquisitionStateErrorsMask\_LostCount} & One or more state transitions have been missed. \\
\hline
\end{tabular}
\caption{\textit{PicamAcquisitionStateErrorsMask} Enumerator Definitions}
\end{table}
7.2 Data Structures

This section provides programming information about the following PICam data structures:

- Camera-Specific Information Data Structures;
  - PicamPixelLocation
  - PicamColumnDefect
  - PicamRowDefect
  - PicamPixelDefectMap
- Camera-Specific Parameter Validation Data Structures;
  - PicamValidationResult
  - PicamValidationResults
  - PicamFailedDependentParameter
  - PicamDependentValidationResult
- Camera-Specific Data Acquisition Data Structures.
  - PicamAcquisitionBuffer
  - PicamAcquisitionStateCounters

7.2.1 Camera-Specific Information Data Structures

This section provides programming information about structures used to define and describe a camera.

7.2.1.1 PicamPixelLocation

Description

PicamPixelLocation specifies the location of a pixel within the sensor array. A standard zero-based X-Y coordinate system is used where:

- X represents the column number;
- Y represents the row number.

Structure Definition

The structure definition for PicamPixelLocation is:

```c
typedef struct PicamPixelLocation
{
    pil16s x;
    pil16s y;
} PicamPixelLocation;
```

Variable Definitions

The variables required by PicamPixelLocation are:

- x: The column coordinate.
- y: The row coordinate.
### 7.2.1.2 PicamColumnDefect

**Description**

*PicamColumnDefect* specifies the location and size of a single defective column on the sensor.

**Structure Definition**

The structure definition for *PicamPixelLocation* is:

```c
typedef struct PicamColumnDefect {
    PicamPixelLocation start;
    uint height;
} PicamColumnDefect;
```

**Variable Definitions**

The variables required by *PicamPixelLocation* are:

- **start**: The top-most defective pixel.
- **height**: The number of rows this column defect spans.

### 7.2.1.3 PicamRowDefect

**Description**

*PicamRowDefect* specifies the location and size of a single defective row on the sensor.

**Structure Definition**

The structure definition for *PicamRowDefect* is:

```c
typedef struct PicamRowDefect {
    PicamPixelLocation start;
    uint width;
} PicamRowDefect;
```

**Variable Definitions**

The variables required by *PicamRowDefect* are:

- **start**: The left-most defective pixel.
- **height**: The number of columns this row defect spans.
7.2.1.4 PicamPixelDefectMap

Description
PicamPixelDefectMap is an array in which all defects for a specified sensor are stored.

Structure Definition
The structure definition for PicamPixelDefectMap is:

typedef struct PicamPixelDefectMap
{
    const PicamColumnDefect* column_defect_array;
    piint column_defect_count;
    const PicamRowDefect* row_defect_array;
    piint row_defect_count;
    const PicamPixelLocation* point_defect_array;
    piint point_defect_count;
} PicamPixelDefectMap;

Variable Definitions
The variables required by PicamPixelDefectMap are:

column_defect_array: A set of all column defects.
This is null where there are no column defects.
column_defect_count: The number of items in column_defect_array.
This is 0 when there are no defective columns.
row_defect_array: A set of all row defects.
This is null when there are no row defects.
row_defect_count: The number of items in row_defect_array.
This is 0 when there are no defective rows.
point_defect_array: A set of all single-point defects.
This is null when there are no single-point defects.
point_defect_count: The number of items in point_defect_array.
This is 0 when there are no single-point defects.
7.2.2 Camera-Specific Parameter Validation Data Structures

This section provides programming information about camera-specific parameter validation structures.

7.2.2.1 PicamValidationResult

Description

PicamValidationResult provides information about the validation status for a single parameter.

Structure Definition

The structure definition for PicamValidationResult is:

```c
typedef struct PicamValidationResult
{
    pibln is_valid;
    const PicamParameter* failed_parameter;
    const PicamConstraintScope* failed_error_constraint_scope;
    const PicamConstraintScope* failed_warning_constraint_scope;
    const PicamParameter* error_constraining_parameter_array;
    piint error_constraining_parameter_count;
    const PicamParameter* warning_constraining_parameter_array;
    piint warning_constraining_parameter_count;
} PicamValidationResult;
```

Variable Definitions

The variables required by PicamValidationResult are:

- **is_valid**: Indicates the validation status for a single parameter. Valid values are:
  - TRUE: Indicates the parameter validation has succeeded.
  - FALSE: Indicates the parameter validation has failed.

- **failed_parameter**: The parameter that has failed validation. This is null when validation has succeeded.

- **failed_error_constraint_scope**: The scope of the error constraint that has failed. This is null when:
  - Validation has succeeded, or
  - Only a warning constraint has failed validation.

- **failed_warning_constraint_scope**: The scope of the warning constraint that has failed. This is null when:
  - Validation has succeeded, or
  - Only an error constraint has failed validation.

- **error_constraining_parameter_array**: An array of parameters involved in constraining the failed parameter when a dependent error failed. This is null otherwise.

- **error_constraining_parameter_count**: The number of items in the array of parameters involved in constraining the failed parameter if a dependent error failed. This is 0 otherwise.

- **warning_constraining_parameter_array**: An array of parameters involved in constraining the failed parameter if a dependent warning failed. This is null otherwise.

- **warning_constraining_parameter_count**: The number of items in the array of parameters involved in constraining the failed parameter if a dependent warning failed. This is 0 otherwise.
7.2.2 PicamValidationResults

Description
PicamValidationResults provides information about the validation status for multiple parameters.

Structure Definition
The structure definition for PicamValidationResults is:

```c
typedef struct PicamValidationResults
{
    pibln is_valid;
    const PicamValidationResult* validation_result_array;
    piint validation_result_count;
} PicamValidationResults;
```

Variable Definitions
The variables required by PicamValidationResults are:

- **is_valid**: Indicates the validation status for multiple tested parameters. Valid values are:
  - TRUE: Indicates all parameter validations have succeeded.
  - FALSE: Indicates one or more parameter validations has failed.

- **validation_result_array**: An array containing a result for each parameter that failed validation; null if validation succeeded.

- **validation_result_count**: The number of failed parameter results; 0 if validation succeeded.

7.2.3 PicamFailedDependentParameter

Description
PicamFailedDependentParameter provides information about a parameter that has failed validation and is itself constrained by a second parameter.

Structure Definition
The structure definition for PicamFailedDependentParameter is:

```c
typedef struct PicamFailedDependentParameter
{
    PicamParameter failed_parameter;
    const PicamConstraintScope* failed_error_constraint_scope;
    const PicamConstraintScope* failed_warning_constraint_scope;
} PicamFailedDependentParameter;
```

Variable Definitions
The variables required by PicamFailedDependentParameter are:

- **failed_parameter**: The parameter whose validation failed and is constrained by another.

- **failed_error_constraint_scope**: The scope of the error constraint that failed. This is null when only a warning constraint failed.

- **failed_warning_constraint_scope**: The scope of the warning constraint that failed. This is null when only an error constraint failed.
### 7.2.2.4 PicamDependentValidationResult

#### Description

PicamDependentValidationResult provides information about the failed validation of a parameter that is constrained by a second parameter.

#### Structure Definition

The structure definition for PicamDependentValidationResult is:

```c
typedef struct PicamDependentValidationResult {
    pibln is_valid;
    PicamParameter constraining_parameter;
    const PicamFailedDependentParameter* failed_dependent_parameter_array;
    piint failed_dependent_parameter_count;
} PicamDependentValidationResult;
```

#### Variable Definitions

The variables required by PicamDependentValidationResult are:

- **is_valid**: Indicates the validation status for a parameter that is constrained by a second parameter. Valid values are:
  - TRUE: Indicates the parameter validation has succeeded.
  - FALSE: Indicates the parameter validation has failed.

- **constraining_parameter**: The parameter whose value impacts the constraints of another.

- **failed_dependent_parameter_array**: An array containing all parameters whose constraints are dependent on constraining_parameter and that have failed validation. This is null when the validation has succeeded.

- **failed_dependent_parameter_count**: The number of items in an array containing all parameters whose constraints are dependent on constraining_parameter and that have failed validation. This is 0 when the validation has succeeded.
7.2.3 Camera-Specific Data Acquisition Data Structures

This section provides programming information for camera-specific data acquisition structures.

7.2.3.1 PicamAcquisitionBuffer

Description
PicamAcquisitionBuffer is a user-allocated buffer into which acquired data is stored.

Structure Definition
The structure definition for PicamAcquisitionBuffer is:

```c
typedef struct PicamAcquisitionBuffer {
    void* memory;
    pi64s memory_size;
} PicamAcquisitionBuffer;
```

Variable Definitions
The variables required by PicamAcquisitionBuffer are:

- **memory**: Pointer to the top of the user-allocated memory location.
- **memory_size**: Number of bytes allocated for use by the user-allocated memory.

7.2.3.2 PicamAcquisitionStateCounters

Description
PicamAcquisitionStateCounters counts all acquisition state transitions registered for detection while acquiring.

Structure Definition
The structure definition for PicamAcquisitionStateCounters is:

```c
typedef struct PicamAcquisitionStateCounters {
    pi64s readout_started_count;
    pi64s readout_ended_count;
} PicamAcquisitionStateCounters;
```

Variable Definitions
The variables required by PicamAcquisitionStateCounters are:

- **readout_started_count**: The number of occurrences where the camera has begun to readout.
- **readout_ended_count**: The number of occurrences where the camera has finished readout.
7.3 Callback Functions

This section provides programming information about the following callbacks used within PICam:

- Camera-Specific Discovery Callbacks
  - PicamDiscoveryCallback()

- Accessory-Specific Discovery Callbacks
  - PicamAccessoryDiscoveryCallback()

- Camera-Specific Parameter Value Callbacks
  - PicamLargeIntegerValueChangedCallback()
  - PicamRoisValueChangedCallback()
  - PicamPulseValueChangedCallback()
  - PicamModulationsValueChangedCallback()

- Camera-Specific Parameter Constraints Callbacks
  - PicamDependentRoisConstraintChangedCallback()
  - PicamDependentPulseConstraintChangedCallback()
  - PicamDependentModulationsConstraintChangedCallback()

- Shared Camera/Accessory Parameter Value Callbacks
  - PicamIntegerValueChangedCallback()
  - PicamFloatingPointValueChangedCallback()
  - PicamWhenStatusParameterValueCallback()
  - PicamIsRelevantChangedCallback()
  - PicamValueAccessChangedCallback()

- Shared Camera/Accessory Parameter Constraints Callbacks
  - PicamDependentCollectionConstraintChangedCallback()
  - PicamDependentRangeConstraintChangedCallback()

- Camera-Specific Data Acquisition Callbacks
  - PicamAcquisitionUpdatedCallback()
  - PicamAcquisitionStateUpdatedCallback()
7.3.1 **Camera-Specific Discovery Callbacks**
This section provides programming information about camera-specific discovery callbacks.

7.3.1.1 *PicamDiscoveryCallback()*

**Description**
*PicamDiscoveryCallback()* is the callback function for camera discovery.

**Syntax**
The syntax for *PicamDiscoveryCallback()* is:

```c
typedef PicamError (PIL_CALL* PicamDiscoveryCallback)(
    const PicamCameraID* id,
    PicamHandle device,
    PicamDiscoveryAction action);
```

**Input Parameters**
The input parameters for *PicamDiscoveryCallback()* are:

- **id**: Pointer to the camera that has been discovered.
- **device**: The handle for an open camera device if id is open within this process. This is null otherwise.
- **action**: The type of discovery.

7.3.2 **Accessory-Specific Discovery Callbacks**
This section provides programming information about accessory-specific discovery callbacks.

7.3.2.1 *PicamAccessoryDiscoveryCallback()*

**Description**
*PicamAccessoryDiscoveryCallback()* is the callback function for accessory discovery.

**Syntax**
The syntax for *PicamAccessoryDiscoveryCallback()* is:

```c
typedef PicamError (PIL_CALL* PicamAccessoryDiscoveryCallback)(
    const PicamAccessoryID* id,
    PicamHandle accessory,
    PicamDiscoveryAction action);
```

**Input Parameters**
The input parameters for *PicamAccessoryDiscoveryCallback()* are:

- **id**: Pointer to the accessory that has been discovered.
- **accessory**: The handle for an open accessory device if id is open within this process. This is null otherwise.
- **action**: The type of discovery.
7.3.3 Camera-Specific Parameter Value Callbacks

This section provides programming information about camera-specific parameter value callbacks.

7.3.3.1 PicamLargeIntegerValueChangedCallback()

Description

PicamLargeIntegerValueChangedCallback() is the change notification callback function called when a parameter’s large integer value has been changed.

Syntax

The syntax for PicamLargeIntegerValueChangedCallback() is:

```c
typedef PicamError (PIL_CALL* PicamLargeIntegerValueChangedCallback)(
    PicamHandle camera,
    PicamParameter parameter,
    pi64s value);
```

Input Parameters

Input parameters for PicamLargeIntegerValueChangedCallback() are:

- **camera**: Handle for the camera for which a parameter’s large integer value has been changed.
- **parameter**: The parameter which has had its large integer value changed.
- **value**: The new large integer value.

7.3.3.2 PicamRoisValueChangedCallback()

Description

PicamRoisValueChangedCallback() is the change notification callback function called when a parameter’s Rois value has been changed.

Syntax

The syntax for PicamRoisValueChangedCallback() is:

```c
typedef PicamError (PIL_CALL* PicamRoisValueChangedCallback)(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamRois* value);
```

Input Parameters

Input parameters for PicamRoisValueChangedCallback() are:

- **camera**: Handle for the camera for which a parameter’s Rois value has been changed.
- **parameter**: The parameter which has had its Rois value changed.
- **value**: Pointer the array location in which the new Rois value is stored.
7.3.3 **PicamPulseValueChangedCallback()**

**Description**

PicamPulseValueChangedCallback() is the change notification callback function called when a parameter’s gate pulse value has been changed.

**Syntax**

The syntax for PicamPulseValueChangedCallback() is:

```c
typedef PicamError (PIL_CALL* PicamPulseValueChangedCallback)(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamPulse* value);
```

**Input Parameters**

Input parameters for PicamPulseValueChangedCallback() are:

- **camera**: Handle for the camera for which a parameter’s gate pulse value has been changed.
- **parameter**: The parameter which has had its gate pulse value changed.
- **value**: Pointer the array in which the new gate pulse value is stored.

7.3.4 **PicamModulationsValueChangedCallback()**

**Description**

PicamModulationsValueChangedCallback() is the change notification callback function called when a parameter’s intensifier modulation sequence value has been changed.

**Syntax**

The syntax for PicamModulationsValueChangedCallback() is:

```c
typedef PicamError (PIL_CALL* PicamModulationsValueChangedCallback)(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamModulations* value);
```

**Input Parameters**

Input parameters for PicamModulationsValueChangedCallback() are:

- **camera**: Handle for the camera for which a parameter’s intensifier modulation sequence value has been changed.
- **parameter**: The parameter which has had its intensifier modulation sequence value changed.
- **value**: Pointer the array in which the new intensifier modulation sequence value is stored.
7.3.4 Shared Camera/Accessory Parameter Value Callbacks

This section provides programming information about shared camera and accessory parameter value callbacks.

7.3.4.1 PicamIntegerValueChangedCallback()

Description

PicamIntegerValueChangedCallback() is the change notification callback function called when a parameter’s integer value has been changed.

Syntax

The syntax for PicamIntegerValueChangedCallback() is:

```c
typedef PicamError (PIL_CALL* PicamIntegerValueChangedCallback)(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    pint value);
```

Input Parameters

Input parameters for PicamIntegerValueChangedCallback() are:

- `camera_or_accessory`: Handle for the hardware for which a parameter’s integer value has been changed.
- `parameter`: The parameter which has had its integer value changed.
- `value`: The new integer value.

7.3.4.2 PicamFloatingPointValueChangedCallback()

Description

PicamFloatingPointValueChangedCallback() is the change notification callback function called when a parameter’s floating point value has been changed.

Syntax

The syntax for PicamFloatingPointValueChangedCallback() is:

```c
typedef PicamError (PIL_CALL* PicamFloatingPointValueChangedCallback)(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    piflt value);
```

Input Parameters

Input parameters for PicamFloatingPointValueChangedCallback() are:

- `camera_or_accessory`: Handle for the hardware for which a parameter’s floating point value has been changed.
- `parameter`: The parameter which has had its floating point value changed.
- `value`: The new floating point value.
7.3.4.3 PicamWhenStatusParameterValueCallback()

Description
PicamWhenStatusParameterValueCallback() is the notification callback function called when a waitable status value has been met or an error has occurred.

Syntax
The syntax for PicamWhenStatusParameterValueCallback() is:

```c
typedef PicamError (PIL_CALL* PicamWhenStatusParameterValueCallback)(
    PicamHandle device_or_accessory,
    PicamParameter parameter,
    int value,
    PicamError error);
```

Input Parameters
Input parameters for PicamWhenStatusParameterValueCallback() are:

- **device_or_accessory**: Handle for the hardware device for which a parameter’s status value has been met.
- **parameter**: The parameter whose status value has been met.
- **value**: The status value that has been met.
- **error**: Any error that occurred to prevent the status value from being met.

7.3.4.4 PicamIsRelevantChangedCallback()

Description
PicamIsRelevantChangedCallback() is the change notification callback function called when a parameter’s relevance has been changed.

Syntax
The syntax for PicamIsRelevantChangedCallback() is:

```c
typedef PicamError (PIL_CALL* PicamIsRelevantChangedCallback)(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    pibln relevant);
```

Input Parameters
Input parameters for PicamIsRelevantChangedCallback() are:

- **camera_or_accessory**: Handle for the hardware for which a parameter’s relevance has been changed.
- **parameter**: The parameter which has had its relevance changed.
- **relevant**: The new relevance.
7.3.4.5 PicamValueAccessChangedCallback()

Description

PicamValueAccessChangedCallback() is the change notification callback function called when a parameter's value access has been changed.

Syntax

The syntax for PicamValueAccessChangedCallback() is:

```c
typedef PicamError (PIL_CALL* PicamValueAccessChangedCallback)(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamValueAccess access);
```

Input Parameters

Input parameters for PicamValueAccessChangedCallback() are:

- camera_or_accessory: Handle for the hardware for which a parameter’s value access has been changed.
- parameter: The parameter which has had its value access changed.
- access: The new value access.
7.3.5 Camera-Specific Parameter Constraints Callbacks

This section provides programming information about camera-specific parameter constraints callbacks.

7.3.5.1 PicamDependentRoisConstraintChangedCallback()

Description
PicamDependentRoisConstraintChangedCallback() is the change notification callback function called when a parameter’s dependent Rois constraints have been changed.

Syntax
The syntax for PicamDependentRoisConstraintChangedCallback() is:

```c
typedef PicamError (PIL_CALL*
    PicamDependentRoisConstraintChangedCallback)
{
    PicamHandle   camera,
    PicamParameter parameter,
    const PicamRoisConstraint*  constraint);
```

Input Parameters
Input parameters for PicamDependentRoisConstraintChangedCallback() are:

- camera: Handle for the camera for which a parameter’s dependent Rois constraints have been changed.
- parameter: The parameter which has had its dependent Rois constraints changed.
- constraint: Pointer to the array in which the new dependent Rois constraints are stored.

7.3.5.2 PicamDependentPulseConstraintChangedCallback()

Description
PicamDependentPulseConstraintChangedCallback() is the change notification callback function called when a parameter’s dependent gate pulse constraints have been changed.

Syntax
The syntax for PicamDependentPulseConstraintChangedCallback() is:

```c
typedef PicamError (PIL_CALL*
    PicamDependentPulseConstraintChangedCallback)
{
    PicamHandle   camera,
    PicamParameter parameter,
    const PicamPulseConstraint*  constraint);
```

Input Parameters
Input parameters for PicamDependentPulseConstraintChangedCallback() are:

- camera: Handle for the camera for which a parameter’s dependent gate pulse constraints have been changed.
- parameter: The parameter which has had its dependent gate pulse constraints changed.
- constraint: Pointer to the array in which the new dependent gate pulse constraints are stored.
7.3.5.3 PicamDependentModulationsConstraintChangedCallback()

Description
PicamDependentModulationsConstraintChangedCallback() is the change notification callback function called when a parameter's dependent intensifier modulations sequence constraints have been changed.

Syntax
The syntax for PicamDependentModulationsConstraintChangedCallback() is:

```c
typedef PicamError (PIL_CALL* PicamDependentModulationsConstraintChangedCallback)(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamModulationsConstraint* constraint);
```

Input Parameters
Input parameters for PicamDependentModulationsConstraintChangedCallback() are:

- `camera`: Handle for the camera for which a parameter's dependent intensifier modulations sequence constraints have been changed.
- `parameter`: The parameter which has had its dependent intensifier modulations sequence constraints changed.
- `constraint`: Pointer to the array in which the new dependent intensifier modulations sequence constraints are stored.
7.3.6 Shared Camera/Accessory Parameter Constraints Callbacks

This section provides programming information about shared camera and accessory parameter constraints callbacks.

7.3.6.1 PicamDependentCollectionConstraintChangedCallback()

Description
PicamDependentCollectionConstraintChangedCallback() is the change notification callback function called when a parameter’s dependent collection constraints have been changed.

Syntax
The syntax for PicamDependentCollectionConstraintChangedCallback() is:

```c
typedef PicamError (PIL_CALL*PicamDependentCollectionConstraintChangedCallback)(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    const PicamCollectionConstraint* constraint);
```

Input Parameters
Input parameters for PicamDependentCollectionConstraintChangedCallback() are:

- camera_or_accessory: Handle for the hardware for which a parameter’s dependent collection constraints have been changed.
- parameter: The parameter which has had its dependent collection constraints changed.
- constraint: Pointer to the array in which the new dependent collection constraints are stored.

7.3.6.2 PicamDependentRangeConstraintChangedCallback()

Description
PicamDependentRangeConstraintChangedCallback() is the change notification callback function called when a parameter’s dependent range constraints have been changed.

Syntax
The syntax for PicamDependentRangeConstraintChangedCallback() is:

```c
typedef PicamError (PIL_CALL*PicamDependentRangeConstraintChangedCallback)(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    const PicamRangeConstraint* constraint);
```

Input Parameters
Input parameters for PicamDependentRangeConstraintChangedCallback() are:

- camera_or_accessory: Handle for the hardware for which a parameter’s dependent range constraints have been changed.
- parameter: The parameter which has had its dependent range constraints changed.
- constraint: Pointer to the array in which the new dependent range constraints are stored.
7.3.7 Camera-Specific Data Acquisition Callbacks

This section provides programming information about camera-specific data acquisition callbacks.

7.3.7.1 PicamAcquisitionUpdatedCallback()

Description
PicamAcquisitionUpdatedCallback() is the change notification callback function called when a camera’s data acquisition status has changed.

Syntax
The syntax for PicamAcquisitionUpdatedCallback() is:

```c
typedef PicamError (PIL_CALL* PicamAcquisitionUpdatedCallback)(
    PicamHandle device,
    const PicamAvailableData* available,
    const PicamAcquisitionStatus* status);
```

Input Parameters
Input parameters for PicamAcquisitionUpdatedCallback() are:

- device: Handle for the camera which is acquiring data.
- available: Pointer to the array in which newly acquired data are stored. If no data are available, this is null.
- status: Pointer to the data acquisition status.

7.3.7.2 PicamAcquisitionStateUpdatedCallback()

Description
PicamAcquisitionStateUpdatedCallback() is the notification callback function called when a camera has transitioned into the acquisition state requested for detection.

Syntax
The syntax for PicamAcquisitionStateUpdatedCallback() is:

```c
typedef PicamError (PIL_CALL* PicamAcquisitionStateUpdatedCallback)(
    PicamHandle device,
    PicamAcquisitionState current,
    const PicamAcquisitionStateCounters* counters,
    PicamAcquisitionStateErrorsMask errors);
```

Input Parameters
Input parameters for PicamAcquisitionStateUpdatedCallback() are:

- device: Handle for the device which transitioned into the acquisition state.
- current: Acquisition state whose transition was detected.
- counters: Pointer to the counted transitions at the time of detection.
- errors: Indicates if any errors have occurred.
7.4 Programmers’ Reference for Advanced APIs

This section provides detailed programming information for the following advanced APIs:

- **Camera-Specific Advanced Discovery APIs**
  - PicamAdvanced_RegisterForDiscovery()
  - PicamAdvanced_UnregisterForDiscovery()
  - PicamAdvanced_DiscoverCameras()
  - PicamAdvanced_StopDiscoveringCameras()
  - PicamAdvanced_IsDiscoveringCameras()

- **Accessory-Specific Advanced Discovery APIs**
  - PicamAccessory_RegisterForDiscovery()
  - PicamAccessory_UnregisterForDiscovery()
  - PicamAccessory_DiscoverAccessories()
  - PicamAccessory_StopDiscoveringAccessories()
  - PicamAccessory_IsDiscoveringAccessories()

- **Camera-Specific Advanced Access APIs**
  - PicamAdvanced_OpenCameraDevice()
  - PicamAdvanced_CloseCameraDevice()
  - PicamAdvanced_GetOpenCameraDevices()
  - PicamAdvanced_GetCameraModel()
  - PicamAdvanced_GetCameraDevice()

- **Shared Camera/Accessory Advanced Access APIs**
  - PicamAdvanced_GetHandleType()

- **Camera-Specific Advanced Information APIs**
  - PicamAdvanced_DestroyPixelDefectMaps()
  - PicamAdvanced_GetPixelDefectMap()

- **Accessory-Specific Information APIs**
  - PicamAccessory_GetLightSourceReference()

- **Shared Camera/Accessory Advanced Information APIs**
  - PicamAdvanced_GetUserState()
  - PicamAdvanced_SetUserState()

- **Camera-Specific Advanced Parameter Value APIs**
  - PicamAdvanced_RegisterForLargeIntegerValueChanged()
  - PicamAdvanced_UnregisterForLargeIntegerValueChanged()
  - PicamAdvanced_RegisterForRoisValueChanged()
  - PicamAdvanced_UnregisterForRoisValueChanged()
  - PicamAdvanced_RegisterForPulseValueChanged()
  - PicamAdvanced_UnregisterForPulseValueChanged()
  - PicamAdvanced_RegisterForModulationsValueChanged()
  - PicamAdvanced_UnregisterForModulationsValueChanged()

- **Shared Camera/Accessory Advanced Parameter Value APIs**
  - PicamAdvanced_RegisterForIntegerValueChanged()
  - PicamAdvanced_UnregisterForIntegerValueChanged()
  - PicamAdvanced_RegisterForExtrinsicIntegerValueChanged()
  - PicamAdvanced_UnregisterForExtrinsicIntegerValueChanged()
  - PicamAdvanced_RegisterForFloatingPointValueChanged()
  - PicamAdvanced_UnregisterForFloatingPointValueChanged()
  - PicamAdvanced_RegisterForExtrinsicFloatingPointValueChanged()
  - PicamAdvanced_UnregisterForExtrinsicFloatingPointValueChanged()
  - PicamAdvanced_NotifyWhenStatusParameterValue()
  - PicamAdvanced_CancelNotifyWhenStatusParameterValue()

*continued on next page*
• Shared Camera/Accessory Advanced Parameter Information APIs
  - PicamAdvanced_RegisterForIsRelevantChanged()
  - PicamAdvanced_UnregisterForIsRelevantChanged()
  - PicamAdvanced_RegisterForValueAccessChanged()
  - PicamAdvanced_UnregisterForValueAccessChanged()
  - PicamAdvanced_GetParameterDynamics()
  - PicamAdvanced_GetParameterExtrinsicDynamics()

• Camera-Specific Advanced Parameter Constraints APIs
  - PicamAdvanced_GetParameterRoisConstraints()
  - PicamAdvanced_RegisterForDependentRoisConstraintChanged()
  - PicamAdvanced_UnregisterForDependentRoisConstraintChanged()
  - PicamAdvanced_GetParameterPulseConstraints()
  - PicamAdvanced_RegisterForDependentPulseConstraintChanged()
  - PicamAdvanced_UnregisterForDependentPulseConstraintChanged()
  - PicamAdvanced_GetParameterModulationsConstraints()
  - PicamAdvanced_RegisterForDependentModulationsConstraintChanged()
  - PicamAdvanced_UnregisterForDependentModulationsConstraintChanged()

• Shared Camera/Accessory Advanced Parameter Constraints APIs
  - PicamAdvanced_GetParameterCollectionConstraints()
  - PicamAdvanced_RegisterForDependentCollectionConstraintChanged()
  - PicamAdvanced_UnregisterForDependentCollectionConstraintChanged()
  - PicamAdvanced_GetParameterRangeConstraints()
  - PicamAdvanced_RegisterForDependentRangeConstraintChanged()
  - PicamAdvanced_UnregisterForDependentRangeConstraintChanged()

• Camera-Specific Advanced Commitment APIs
  - Picam_DestroyValidationResult()
  - Picam_DestroyValidationResults()
  - PicamAdvanced_ValidateParameter()
  - PicamAdvanced_ValidateParameters()
  - Picam_DestroyDependentValidationResult()
  - PicamAdvanced_ValidateDependentParameter()
  - PicamAdvanced_CommitParametersToCameraDevice()
  - PicamAdvanced_RefreshParameterFromCameraDevice()
  - PicamAdvanced_RefreshParametersFromCameraDevice()

• Camera-Specific Advanced Acquisition Setup APIs
  - PicamAdvanced_GetAcquisitionBuffer()
  - PicamAdvanced_SetAcquisitionBuffer()

• Camera-Specific Advanced Acquisition Notification APIs
  - PicamAdvanced_RegisterForAcquisitionUpdated()
  - PicamAdvanced_UnregisterForAcquisitionUpdated()

• Camera-Specific Advanced Acquisition State Notification APIs
  - PicamAdvanced_CanRegisterForAcquisitionStateUpdated()
  - PicamAdvanced_RegisterForAcquisitionStateUpdated()
  - PicamAdvanced_UnregisterForAcquisitionStateUpdated()

• Camera-Specific Advanced Acquisition Control APIs
  - PicamAdvanced_HasAcquisitionBufferOverrun()
  - PicamAdvanced_CanClearReadoutCountOnline()
  - PicamAdvanced_ClearReadoutCountOnline()
7.4.1 Camera-Specific Advanced Discovery APIs

This section provides programming information for advanced camera-specific discovery APIs.

7.4.1.1 PicamAdvanced_RegisterForDiscovery()

Description

PicamAdvanced_RegisterForDiscovery() registers a function to call when camera discovery is made.

NOTE: Multiple functions may be registered. When this is the case, the functions are called in the order in which they have been registered.

Callback functions are called when any camera state that affects availability changes, such as when:

- A camera is powered on and/or connected to the host computer;
- A connected camera is powered off or disconnected from the host computer;
- A camera is opened in another process;
- A camera is closed in another process.

Callback functions are also called when a camera has suffered a critical malfunction.

Callbacks are called asynchronously from another thread, but are serialized on that thread. This means that additional notifications do not occur simultaneously, but occur after each callback returns.

A camera may be unavailable for multiple reasons. Therefore, although callbacks may repeatedly indicate a camera is lost each time one of the above states change, but the camera is still not available.

Call PicamAdvanced_UnregisterForDiscovery() to unregister each callback once it is no longer required.

Syntax

The syntax for PicamAdvanced_RegisterForDiscovery() is:

```
PICAM_API PicamAdvanced_RegisterForDiscovery(
    PicamDiscoveryCallback discover)
```

Input Parameters

Input parameters for PicamAdvanced_RegisterForDiscovery() are:

- `discover`: The name assigned to the discovery callback function being registered.

Output Parameters

There are no output parameters associated with PicamAdvanced_RegisterForDiscovery().

Related APIs

For additional information, refer to the following related APIs:

- PicamAdvanced_UnregisterForDiscovery()
### 7.4.1.2 PicamAdvanced_UnregisterForDiscovery()

**Description**

`PicamAdvanced_UnregisterForDiscovery()` removes the function from the discovery process such that it is no longer called when a camera discovery is made.

**Syntax**

The syntax for `PicamAdvanced_UnregisterForDiscovery()` is:

```c
PICAM_API PicamAdvanced_UnregisterForDiscovery(
    PicamDiscoveryCallback discover);
```

**Input Parameters**

Input parameters for `PicamAdvanced_UnregisterForDiscovery()` are:

- `discover`: The name assigned to the discovery callback function being unregistered.

**Output Parameters**

There are no output parameters associated with `PicamAdvanced_UnregisterForDiscovery()`.

**Related APIs**

For additional information, refer to the following related APIs:

- `PicamAdvanced_RegisterForDiscovery()`

### 7.4.1.3 PicamAdvanced_DiscoverCameras()

**Description**

`PicamAdvanced_DiscoverCameras()` asynchronously initiates the camera discovery process.

To halt the discovery process, call `PicamAdvanced_StopDiscoveringCameras()`.

**Syntax**

The syntax for `PicamAdvanced_DiscoverCameras()` is:

```c
PICAM_API PicamAdvanced_DiscoverCameras (void);
```

**Input Parameters**

There are no input parameters associated with `PicamAdvanced_DiscoverCameras()`.

**Output Parameters**

There are no output parameters associated with `PicamAdvanced_DiscoverCameras()`.

**Related APIs**

For additional information, refer to the following related APIs:

- `PicamAdvanced_StopDiscoveringCameras()`
7.4.1.4 PicamAdvanced_StopDiscoveringCameras()

Description
PicamAdvanced_StopDiscoveringCameras() stops the camera discovery process.

Syntax
The syntax for PicamAdvanced_StopDiscoveringCameras() is:

```c
PICAM_API PicamAdvanced_StopDiscoveringCameras (void);
```

Input Parameters
There are no input parameters associated with PicamAdvanced_StopDiscoveringCameras().

Output Parameters
There are no output parameters associated with PicamAdvanced_StopDiscoveringCameras().

Related APIs
For additional information, refer to the following related APIs:
- PicamAdvanced_DiscoverCameras()

7.4.1.5 PicamAdvanced_IsDiscoveringCameras()

Description
PicamAdvanced_IsDiscoveringCameras() determines if camera discovery is enabled.

Syntax
The syntax for PicamAdvanced_IsDiscoveringCameras() is:

```c
PICAM_API PicamAdvanced_IsDiscoveringCameras(
    pibln* discovering);
```

Input Parameters
There are no input parameters associated with PicamAdvanced_IsDiscoveringCameras().

Output Parameters
Output parameters for PicamAdvanced_IsDiscoveringCameras() are:

```c
discovering: Indicates if camera discovery is currently enabled.
Valid values are:
```
- TRUE
  Camera discovery is enabled.
- FALSE
  Camera discovery is disabled.
```
7.4.2 Accessory-Specific Advanced Discovery APIs

This section provides programming information for accessory-specific advanced discovery APIs.

7.4.2.1 PicamAccessory_RegisterForDiscovery()

Description

PicamAccessory_RegisterForDiscovery() registers a function to call when accessory discovery is made.

NOTE:

Multiple functions may be registered. When this is the case, the functions are called in the order in which they have been registered.

Callback functions are called when any accessory state that affects availability changes, such as when:

- An accessory is powered on and/or connected to the host computer;
- A connected accessory is powered off or disconnected from the host computer;
- An accessory is opened in another process;
- An accessory is closed in another process.

Callback functions are also called when an accessory has suffered a critical malfunction.

Callbacks are called asynchronously from another thread, but are serialized on that thread. This means that additional notifications do not occur simultaneously, but occur after each callback returns.

An accessory may be unavailable for multiple reasons. Therefore, although callbacks may repeatedly indicate an accessory is lost each time one of the above states change, but the accessory is still not available.

Call PicamAccessory_UnregisterForDiscovery() to unregister each callback once it is no longer required.

Syntax

The syntax for PicamAccessory_RegisterForDiscovery() is:

```c
PICAM_API PicamAccessory_RegisterForDiscovery(
    PicamAccessoryDiscoveryCallback discover)
```

Input Parameters

Input parameters for PicamAccessory_RegisterForDiscovery() are:

```
discover:  The name assigned to the discovery callback function being registered.
```

Output Parameters

There are no output parameters associated with PicamAdvanced_RegisterForDiscovery().

Related APIs

For additional information, refer to the following related APIs:

- PicamAccessory_UnregisterForDiscovery()
7.4.2.2 PicamAccessory_UnregisterForDiscovery()

Description
PicamAccessory_UnregisterForDiscovery() removes the function from the
discovery process such that it is no longer called when an accessory discovery is made.

Syntax
The syntax for PicamAccessory_UnregisterForDiscovery() is:

```
PICAM_API PicamAccessory_UnregisterForDiscovery(
    PicamAccessoryDiscoveryCallback discover);
```

Input Parameters
Input parameters for PicamAccessory_UnregisterForDiscovery() are:

```
discover: The name assigned to the discovery callback function being
unregistered.
```

Output Parameters
There are no output parameters associated with
PicamAccessory_UnregisterForDiscovery().

Related APIs
For additional information, refer to the following related APIs:

- PicamAccessory_RegisterForDiscovery()

7.4.2.3 PicamAccessory_DiscoverAccessories()

Description
PicamAccessory_DiscoverAccessories() asynchronously initiates the accessory
discovery process.

To halt the discovery process, call
PicamAccessory_StopDiscoveringAccessories().

Syntax
The syntax for PicamAccessory_DiscoverAccessories() is:

```
PICAM_API PicamAccessory_DiscoverAccessories (void);
```

Input Parameters
There are no input parameters associated with
PicamAccessory_DiscoverAccessories().

Output Parameters
There are no output parameters associated with
PicamAccessory_DiscoverAccessories().

Related APIs
For additional information, refer to the following related APIs:

- PicamAccessory_StopDiscoveringAccessories()
7.4.2.4 PicamAccessory_StopDiscoveringAccessories()

Description
PicamAccessory_StopDiscoveringAccessories() stops the accessory discovery process.

Syntax
The syntax for PicamAccessory_StopDiscoveringAccessories() is:

```c
PICAM_API PicamAccessory_StopDiscoveringAccessories (void);
```

Input Parameters
There are no input parameters associated with PicamAccessory_StopDiscoveringAccessories().

Output Parameters
There are no output parameters associated with PicamAccessory_StopDiscoveringAccessories().

Related APIs
For additional information, refer to the following related APIs:
- PicamAccessory_DiscoverAccessories()

7.4.2.5 PicamAccessory_IsDiscoveringAccessories()

Description
PicamAccessory_IsDiscoveringAccessories() determines if accessory discovery is enabled.

Syntax
The syntax for PicamAccessory_IsDiscoveringAccessories() is:

```c
PICAM_API PicamAccessory_IsDiscoveringAccessories(
    pibln* discovering);
```

Input Parameters
There are no input parameters associated with PicamAccessory_IsDiscoveringAccessories().

Output Parameters
Output parameters for PicamAccessory_IsDiscoveringAccessories() are:

- discovering: Indicates if accessory discovery is currently enabled. Valid values are:
  - TRUE
    Accessory discovery is enabled.
  - FALSE
    Accessory discovery is disabled.
7.4.3 Camera-Specific Advanced Access APIs

This section provides programming information for camera-specific advanced access APIs.

7.4.3.1 PicamAdvanced_OpenCameraDevice()

Description
PicamAdvanced_OpenCameraDevice() opens the specified camera and returns a handle to the device.

When done, all resources that have been assigned for use by the camera/device must be released by calling:

- Picam_CloseCamera();
- PicamAdvanced_CloseCameraDevice();

Syntax
The syntax for PicamAdvanced_OpenCameraDevice() is:

```c
PICAM_API PicamAdvanced_OpenCameraDevice(
    const PicamCameraID* id,
    PicamHandle* device);
```

Input Parameters
Input parameters for PicamAdvanced_OpenCameraDevice() are:

- id: Pointer to the camera id for the camera device to be opened.

Output Parameters
Output parameters for PicamAdvanced_OpenCameraDevice() are:

- device: Pointer to the handle assigned to the camera device that has been opened.

Related APIs
For additional information, refer to the following related APIs:

- Picam_CloseCamera();
- PicamAdvanced_CloseCameraDevice();

7.4.3.2 PicamAdvanced_CloseCameraDevice()

Description
PicamAdvanced_CloseCameraDevice() releases all resources associated with the specified device.

Syntax
The syntax for PicamAdvanced_CloseCameraDevice() is:

```c
PICAM_API PicamAdvanced_CloseCameraDevice(
    PicamHandle device)
```

Input Parameters
Input parameters for PicamAdvanced_CloseCameraDevice() are:

- device: Handle for the camera for which all resources are to be released.

Output Parameters
There are no output parameters associated with PicamAdvanced_CloseCameraDevice().
7.4.3.3 PicamAdvanced_GetOpenCameraDevices()

Description
PicamAdvanced_GetOpenCameraDevices() returns an allocated array of open camera device handles.
in device_array whose number of items is in device_count.

Returns null and 0 (respectively) if no cameras are opened in this process.

Syntax
The syntax for PicamAdvanced_GetOpenCameraDevices() is:

```
PICAM_API PicamAdvanced_GetOpenCameraDevices(
    const PicamHandle** device_array,
    UINT* device_count);
```

Input Parameters
There are no input parameters associated with PicamAdvanced_GetOpenCameraDevices().

Output Parameters
Output parameters for PicamAdvanced_GetOpenCameraDevices() are:

- **device_array**: Pointer to the array in which the list of handles for open camera devices is stored.
  This is null when there are no open camera devices.

- **device_count**: Pointer to the memory location in which the number of open camera devices is stored.
  This is 0 when there are no open camera devices.

Related APIs
For additional information, refer to the following related APIs:
- Picam_DestroyHandles()

7.4.3.4 PicamAdvanced_GetCameraModel()

Description
PicamAdvanced_GetCameraModel() returns the handle for a specified camera model.

Syntax
The syntax for PicamAdvanced_GetCameraModel() is:

```
PICAM_API PicamAdvanced_GetCameraModel(
    PicamHandle camera,
    PicamHandle* model);
```

Input Parameters
Input parameters for PicamAdvanced_GetCameraModel() are:

- **camera**: Specifies the camera model or camera device for which the handle is to be returned.

Output Parameters
Output parameters for PicamAdvanced_GetCameraModel() are:

- **model**: Pointer to the memory location in which the handle for the camera model is stored.
7.4.3.5 \texttt{PicamAdvanced\_GetCameraDevice()}

\textbf{Description}

\texttt{PicamAdvanced\_GetCameraDevice()} returns the handle for a specified camera device.

\textbf{Syntax}

The syntax for \texttt{PicamAdvanced\_GetCameraDevice()} is:

\begin{verbatim}
PICAM\_API PicamAdvanced\_GetCameraDevice(
    PicamHandle camera,
    PicamHandle* device);
\end{verbatim}

\textbf{Input Parameters}

Input parameters for \texttt{PicamAdvanced\_GetCameraDevice()} are:

\begin{itemize}
  \item \texttt{camera}: Specifies the camera device or camera model for which the handle is to be returned.
\end{itemize}

\textbf{Output Parameters}

Output parameters for \texttt{PicamAdvanced\_GetCameraDevice()} are:

\begin{itemize}
  \item \texttt{device}: Pointer to the memory location in which the handle for the camera device is stored.
\end{itemize}
7.4.4 Shared Camera/Accessory Advanced Access APIs

This section provides programming information for shared camera and accessory advanced access APIs.

7.4.4.1 PicamAdvanced_GetHandleType()

Description
PicamAdvanced_GetHandleType() returns the type of handle for a specified handle.

Syntax
The syntax for PicamAdvanced_GetHandleType() is:

    PICAM_API PicamAdvanced_GetHandleType(
            PicamHandle handle,
            PicamHandleType* type);

Input Parameters
Input parameters for PicamAdvanced_GetHandleType() are:

    handle: Handle for which the handle type is to be determined.

Output Parameters
Output parameters for PicamAdvanced_GetHandleType() are:

    type: The handle type for the specified handle.
7.4.5 Camera-Specific Information APIs

This section provides programming information about advanced camera-specific information APIs.

7.4.5.1 PicamAdvanced_DestroyPixelDefectMaps()

Description

PicamAdvanced_DestroyPixelDefectMaps() releases memory that has been allocated by PICam for use by defect_map_array. If defect_map_array is null, calling PicamAdvanced_DestroyPixelDefectMaps() has no effect.

NOTE: defect_map_array may be a single PicamPixelDefectMap allocated by PICam.

Syntax

The syntax for PicamAdvanced_DestroyPixelDefectMaps() is:

```c
PICAM_API PicamAdvanced_DestroyPixelDefectMaps(
    const PicamPixelDefectMap* pixel_defect_map_array);
```

Input Parameters

Input parameters for PicamAdvanced_DestroyPixelDefectMaps() are:

- `pixel_defect_map_array`: Pointer to the array that is to be released.

Output Parameters

There are no output parameters associated with PicamAdvanced_DestroyPixelDefectMaps().

Related Structures

For additional information, refer to the following related structures:

- PicamPixelDefectMap
7.4.5.2 PicamAdvanced_GetPixelDefectMap()

Description
PicamAdvanced_GetPixelDefectMap() returns an allocated array/map in which
defective pixels information for a specified camera is stored.

Syntax
The syntax for PicamAdvanced_GetPixelDefectMap() is:

```c
PICAM_API PicamAdvanced_GetPixelDefectMap(
    PicamHandle camera,
    const PicamPixelDefectMap** pixel_defect_map);
```

Input Parameters
Input parameters for PicamAdvanced_GetPixelDefectMap() are:

- camera: Handle for the camera for which PicamPixelDefectMap is to be returned.
  Valid values are:
  - device handle;
  - model handle.
  
  **NOTE:** device and model share the same PicamPixelDefectMap.

Output Parameters
Output parameters for PicamAdvanced_GetPixelDefectMap() are:

- pixel_defect_map: Pointer to the PicamPixelDefectMap array in which defective pixel information is stored.
  When no information is available for the specified camera, this is an array describing zero defects.
  
  **NOTE:** This memory is allocated by PICam and must be released by calling
  PicamAdvanced_DestroyPixelDefectMaps().

Related APIs
For additional information, refer to the following related APIs:

- PicamAdvanced_DestroyPixelDefectMaps()

Related Structures
For additional information, refer to the following related structures:

- PicamPixelDefectMap
7.4.6 Accessory-Specific Information APIs

This section provides programming information about advanced accessory-specific information APIs.

7.4.6.1 PicamAccessory_GetLightSourceReference()

Description

PicamAccessory_GetLightSourceReference() returns a wavelength reference calibration for an accessory.

![NOTE:]

Prior to program termination, memory that has been dynamically allocated to calibration_array must be released by calling Picam_DestroyCalibrations().

Syntax

The syntax for PicamAccessory_GetLightSourceReference() is:

```c
PICAM_API PicamAccessory_GetLightSourceReference(
    PicamHandle accessory,
    const PicamCalibration** counts_vs_nm);
```

Input Parameters

Input parameters for PicamAccessory_GetLightSourceReference() are:

- accessory: Handle for the accessory for which the light source reference is returned.

Output Parameters

Output parameters for PicamAccessory_GetLightSourceReference() are:

- counts_vs_nm: Pointer to the allocated wavelength reference calibration where the x-coordinates are wavelengths, in nanometers (nm), and the y-coordinates are the intensity values at those wavelengths.

Related APIs

For additional information, refer to the following related APIs:

- Picam_DestroyCalibrations().

Related Structures

For additional information, refer to the following related structure definition:

- PicamCalibration.
7.4.7 Shared Camera/Accessory Advanced Information APIs

This section provides programming information about shared camera and accessory advanced information APIs.

7.4.7.1 PicamAdvanced_GetUserState()

Description

PicamAdvanced_GetUserState() returns user-state information for the specified hardware.

NOTE: This API is thread safe.

Syntax

The syntax for PicamAdvanced_GetUserState() is:

```
PICAM_API PicamAdvanced_GetUserState(
    PicamHandle camera_or_accessory,
    void** user_state);
```

Input Parameters

Input parameters for PicamAdvanced_GetUserState() are:

- `camera_or_accessory`: Handle for the hardware for which user state information is to be returned.
  Valid values are:
  - `device` handle;
  - `model` handle.

  NOTE: device and model share the same user state.

Output Parameters

Output parameters for PicamAdvanced_GetUserState() are:

- `user_state`: Pointer to the memory location where user-state information is stored.
7.4.7.2  PicamAdvanced_SetUserState()

Description
PicamAdvanced_SetUserState() sets user-state information for the specified hardware.

NOTE: This API is thread safe.

Syntax
The syntax for PicamAdvanced_SetUserState() is:

```c
PICAM_API PicamAdvanced_SetUserState(
    PicamHandle camera_or_accessory,
    void* user_state);
```

Input Parameters
Input parameters for PicamAdvanced_SetUserState() are:

- camera_or_accessory: Handle for the hardware for which user state information is to be configured.
  Valid values are:
  - device handle;
  - model handle.
  
  NOTE: device and model share the same user state.

Output Parameters
Output parameters for PicamAdvanced_SetUserState() are:

- user_state: Pointer to the memory location where user-state information is stored.
7.4.8 Camera-Specific Advanced Parameter Value APIs

This section provides programming information for camera-specific advanced parameter value APIs.

7.4.8.1 PicamAdvanced_RegisterForLargeIntegerValueChanged()

Description
PicamAdvanced_RegisterForLargeIntegerValueChanged() registers a function to call when the large integer value for specified camera parameter has been set, even if it is changed as a result of a different parameter’s value being changed.

NOTE: Multiple functions may be registered. When this occurs, functions are called in the order in which they have been registered.

Registered callbacks are called synchronously from within the thread in which associated parameter values are being set (i.e., called prior to returning from the set operation.)

NOTE: Parameters whose values have change due to external influences (e.g., representing the status of camera hardware,) do not result in a callback’s being called.

Call PicamAdvanced_UnregisterForLargeIntegerValueChanged() to unregister each callback once it is no longer required.

Syntax
The syntax for PicamAdvanced_RegisterForLargeIntegerValueChanged() is:

PICAM_API PicamAdvanced_RegisterForLargeIntegerValueChanged(
    PicamHandle camera,  
    PicamParameter parameter,  
    PicamLargeIntegerValueChangedCallback changed);

Input Parameters
Input parameters for PicamAdvanced_RegisterForLargeIntegerValueChanged() are:

    camera: Handle for the camera for which the callback is being registered.
    parameter: The parameter for which the callback is being registered.
    changed: The name assigned to the callback function being registered.

Output Parameters
There are no output parameters associated with PicamAdvanced_RegisterForLargeIntegerValueChanged().

Related APIs
For additional information, refer to the following related APIs:

- Picam_SetParameterLargeIntegerValue();
- PicamAdvanced_UnregisterForLargeIntegerValueChanged().
7.4.8.2 PicamAdvanced_UnregisterForLargeIntegerValueChanged()

Description
PicamAdvanced_UnregisterForLargeIntegerValueChanged() removes the callback function so that it is no longer called when the large integer value for a specified parameter is changed.

Syntax
The syntax for PicamAdvanced_UnregisterForLargeIntegerValueChanged() is:

```c
PICAM_API PicamAdvanced_UnregisterForLargeIntegerValueChanged(
    PicamHandle camera,
    PicamParameter parameter,
    PicamLargeIntegerValueChangedCallback changed);
```

Input Parameters
Input parameters for PicamAdvanced_UnregisterForLargeIntegerValueChanged() are:
- `camera`: Handle for the camera for which the callback is being unregistered.
- `parameter`: The parameter for which the callback is being unregistered.
- `changed`: The name assigned to the callback function being unregistered.

Output Parameters
There are no output parameters associated with PicamAdvanced_UnregisterForLargeIntegerValueChanged().

Related APIs
For additional information, refer to the following related APIs:
- PicamAdvanced_RegisterForLargeIntegerValueChanged().
7.4.8.3 PicamAdvanced_RegisterForRoisValueChanged()

Description
PicamAdvanced_RegisterForRoisValueChanged() registers a function to call when the value of a specified Rois parameter has been set, even if it is changed as a result of a different parameter's value being changed.

**NOTE:**

Multiple functions may be registered. When this is the case, the functions are called in the order in which they have been registered.

Registered callbacks are called synchronously from within the thread in which associated parameter values are being set (i.e., called prior to returning from the set operation.)

**NOTE:**

Parameters whose values have change due to external influences (e.g., representing the status of camera hardware,) do not result in a callback's being called.

Call PicamAdvanced_UnregisterForRoisValueChanged() to unregister each callback once it is not longer required.

**Syntax**
The syntax for PicamAdvanced_RegisterForRoisValueChanged() is:

```c
PICAM_API PicamAdvanced_RegisterForRoisValueChanged(
    PicamHandle camera,
    PicamParameter parameter,
    PicamRoisValueChangedCallback changed);
```

**Input Parameters**
Input parameters for PicamAdvanced_RegisterForRoisValueChanged() are:

- **camera**: Handle for the camera for which the callback is being registered.
- **parameter**: The parameter for which the callback is being registered.
- **changed**: The name assigned to the callback function being registered.

**Output Parameters**
There are no output parameters associated with PicamAdvanced_RegisterForRoisValueChanged().

**Related APIs**
For additional information, refer to the following related APIs:

- Picam_SetParameterRoisValue();
- PicamAdvanced_UnregisterForRoisValueChanged().
7.4.8.4 PicamAdvanced_UnregisterForRoisValueChanged()

Description
PicamAdvanced_UnregisterForRoisValueChanged() removes the callback function so that it is no longer called when the value of a specified Rois parameter is changed.

Syntax
The syntax for PicamAdvanced_UnregisterForRoisValueChanged() is:

```
PICAM_API PicamAdvanced_UnregisterForRoisValueChanged(
    PicamHandle camera,
    PicamParameter parameter,
    PicamRoisValueChangedCallback changed);
```

Input Parameters
Input parameters for PicamAdvanced_UnregisterForRoisValueChanged() are:

- **camera**: Handle for the camera for which the callback is being unregistered.
- **parameter**: The parameter for which the callback is being unregistered.
- **changed**: The name assigned to the callback function being unregistered.

Output Parameters
There are no output parameters associated with PicamAdvanced_UnregisterForRoisValueChanged().

Related APIs
For additional information, refer to the following related APIs:

- PicamAdvanced_RegisterForRoisValueChanged().
7.4.8.5 PicamAdvanced_RegisterForPulseValueChanged()

Description
PicamAdvanced_RegisterForPulseValueChanged() registers a function to call when the value of a specified gate pulse parameter has been set, even if it is changed as a result of a different parameter’s value being changed.

**NOTE:**

Multiple functions may be registered. When this is the case, the functions are called in the order in which they have been registered.

Registered callbacks are called synchronously from within the thread in which associated parameter values are being set (i.e., called prior to returning from the set operation.)

**NOTE:**

Parameters whose values have change due to external influences (e.g., representing the status of camera hardware,) do not result in a callback’s being called.

Call PicamAdvanced_UnregisterForPulseValueChanged() to unregister each callback once it is not longer required.

Syntax
The syntax for PicamAdvanced_RegisterForPulseValueChanged() is:

```
PICAM_API PicamAdvanced_RegisterForPulseValueChanged(
    PicamHandle camera,
    PicamParameter parameter,
    PicamPulseValueChangedCallback changed);
```

Input Parameters
Input parameters for PicamAdvanced_RegisterForPulseValueChanged() are:

- **camera:** Handle for the camera for which the callback is being registered.
- **parameter:** The parameter for which the callback is being registered.
- **changed:** The name assigned to the callback function being registered.

Output Parameters
There are no output parameters associated with PicamAdvanced_RegisterForPulseValueChanged().

Related APIs
For additional information, refer to the following related APIs:

- Picam_SetParameterPulseValue();
- PicamAdvanced_UnregisterForPulseValueChanged().
7.4.8.6 PicamAdvanced_UnregisterForPulseValueChanged()

Description
PicamAdvanced_UnregisterForPulseValueChanged() removes the callback function so that it is no longer called when the value of a specified gate pulse parameter is changed.

Syntax
The syntax for PicamAdvanced_UnregisterForPulseValueChanged() is:

```c
PICAM_API PicamAdvanced_UnregisterForPulseValueChanged(
    PicamHandle camera,
    PicamParameter parameter,
    PicamPulseValueChangedCallback changed);
```

Input Parameters
Input parameters for PicamAdvanced_UnregisterForPulseValueChanged() are:

- camera: Handle for the camera for which the callback is being unregistered.
- parameter: The parameter for which the callback is being unregistered.
- changed: The name assigned to the callback function being unregistered.

Output Parameters
There are no output parameters associated with PicamAdvanced_UnregisterForPulseValueChanged().

Related APIs
For additional information, refer to the following related APIs:
- PicamAdvanced_RegisterForPulseValueChanged().
7.4.8.7 PicamAdvanced_RegisterForModulationsValueChanged()

Description
PicamAdvanced_RegisterForModulationsValueChanged() registers a function to call when the value of a specified intensifier modulation sequence parameter has been set, even if it is changed as a result of a different parameter’s value being changed.

NOTE: Multiple functions may be registered. When this is the case, the functions are called in the order in which they have been registered.

Registered callbacks are called synchronously from within the thread in which associated parameter values are being set (i.e., called prior to returning from the set operation.)

NOTE: Parameters whose values have change due to external influences (e.g., representing the status of camera hardware,) do not result in a callback’s being called.

Call PicamAdvanced_UnregisterForModulationsValueChanged() to unregister each callback once it is not longer required.

Syntax
The syntax for PicamAdvanced_RegisterForModulationsValueChanged() is:

```c
PICAM_API PicamAdvanced_RegisterForModulationsValueChanged(
    PicamHandle camera,
    PicamParameter parameter,
    PicamModulationsValueChangedCallback changed);
```

Input Parameters
Input parameters for PicamAdvanced_RegisterForModulationsValueChanged() are:

- camera: Handle for the camera for which the callback is being registered.
- parameter: The parameter for which the callback is being registered.
- changed: The name assigned to the callback function being registered.

Output Parameters
There are no output parameters associated with PicamAdvanced_RegisterForModulationsValueChanged().

Related APIs
For additional information, refer to the following related APIs:

- Picam_SetParameterModulationsValue();
- PicamAdvanced_UnregisterForModulationsValueChanged().
7.4.8.8  PicamAdvanced_UnregisterForModulationsValueChanged()

Description
PicamAdvanced_UnregisterForModulationsValueChanged() removes the callback function so that it is no longer called when the value of an intensifier modulation sequence parameter is changed.

Syntax
The syntax for PicamAdvanced_UnregisterForModulationsValueChanged() is:

```c
PICAM_API PicamAdvanced_UnregisterForModulationsValueChanged(
    PicamHandle camera,
    PicamParameter parameter,
    PicamModulationsValueChangedCallback changed);
```

Input Parameters
Input parameters for PicamAdvanced_UnregisterForModulationsValueChanged() are:

- `camera`: Handle for the camera for which the callback is being unregistered.
- `parameter`: The parameter for which the callback is being unregistered.
- `changed`: The name assigned to the callback function being unregistered.

Output Parameters
There are no output parameters associated with PicamAdvanced_UnregisterForModulationsValueChanged().

Related APIs
For additional information, refer to the following related APIs:

- PicamAdvanced_RegisterForModulationsValueChanged().
7.4.9 Shared Camera/Accessory Advanced Parameter Value APIs

This section provides programming information for shared camera/accessory advanced parameter value APIs.

7.4.9.1 PicamAdvanced_RegisterForIntegerValueChanged()

Description
PicamAdvanced_RegisterForIntegerValueChanged() registers a function to call when the integer value for specified hardware parameter has been set, even if it is changed as a result of a different parameter’s value being changed.

**NOTE:**
Multiple functions may be registered. When this is the case, the functions are called in the order in which they have been registered.

Registered callbacks are called synchronously from within the thread in which associated parameter values are being set (i.e., called prior to returning from the set operation.)

**NOTE:**
Parameters whose values have changed due to external influences (e.g., representing the status of camera hardware,) do not result in a callback’s being called.

Call PicamAdvanced_UnregisterForIntegerValueChanged() to unregister each callback once it is not longer required.

Syntax
The syntax for PicamAdvanced_RegisterForIntegerValueChanged() is:

```
PICAM_API PicamAdvanced_RegisterForIntegerValueChanged(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamIntegerValueChangedCallback changed);
```

Input Parameters
Input parameters for PicamAdvanced_RegisterForIntegerValueChanged() are:

- camera_or_accessory: Handle for the hardware for which the callback is being registered.
- parameter: The parameter for which the callback is being registered.
- changed: The name assigned to the callback function being registered.

Output Parameters
There are no output parameters associated with PicamAdvanced_RegisterForIntegerValueChanged().

Related APIs
For additional information, refer to the following related APIs:

- Picam_SetParameterIntegerValue();
- PicamAdvanced_UnregisterForIntegerValueChanged().
7.4.9.2 PicamAdvanced_UnregisterForIntegerValueChanged()

Description
PicamAdvanced_UnregisterForIntegerValueChanged() removes the callback function so that it is no longer called when the integer value for a specified parameter is changed.

Syntax
The syntax for PicamAdvanced_UnregisterForIntegerValueChanged() is:

```c
PICAM_API PicamAdvanced_UnregisterForIntegerValueChanged(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamIntegerValueChangedCallback changed);
```

Input Parameters
Input parameters for PicamAdvanced_UnregisterForIntegerValueChanged() are:

- `camera_or_accessory`: Handle for the hardware for which the callback is being unregistered.
- `parameter`: The parameter for which the callback is being unregistered.
- `changed`: The name assigned to the callback function being unregistered.

Output Parameters
There are no output parameters associated with PicamAdvanced_UnregisterForIntegerValueChanged().

Related APIs
For additional information, refer to the following related APIs:

- PicamAdvanced_RegisterForIntegerValueChanged().
7.4.9.3 *PicamAdvanced_RegisterForExtrinsicIntegerValueChanged()*

**Description**

*PicamAdvanced_RegisterForExtrinsicIntegerValueChanged()* registers a function to call when the integer value for specified hardware parameter has changed due to external influences (e.g., representing the status of hardware).

**NOTE:**

Multiple functions may be registered. When this is the case, the functions are called in the order in which they have been registered.

Registered callbacks are called asynchronously on another thread.

**NOTE:**

*PicamAdvanced_UnregisterForExtrinsicIntegerValueChanged()* must be called to unregister each callback once it is no longer required.

**Syntax**

The syntax for *PicamAdvanced_RegisterForExtrinsicIntegerValueChanged()* is:

```c
PICAM_API PicamAdvanced_RegisterForExtrinsicIntegerValueChanged(
    PicamHandle device_or_accessory,
    PicamParameter parameter,
    PicamIntegerValueChangedCallback changed);
```

**Input Parameters**

Input parameters for *PicamAdvanced_RegisterForExtrinsicIntegerValueChanged()* are:

- `device_or_accessory`: Handle for the hardware for which the callback is being registered.
- `parameter`: The parameter for which the callback is being registered.
- `changed`: The name assigned to the callback function being registered.

**Output Parameters**

There are no output parameters associated with *PicamAdvanced_RegisterForExtrinsicIntegerValueChanged()*.

**Related APIs**

For additional information, refer to the following related APIs:

- *PicamAdvanced_UnregisterForExtrinsicIntegerValueChanged()*
7.4.9.4 PicamAdvanced_UnregisterForExtrinsicIntegerValueChanged()

Description
PicamAdvanced_UnregisterForExtrinsicIntegerValueChanged() removes the callback function so that it is no longer called when the integer value for a specified parameter is changed due to external influences (e.g., representing the status of camera hardware).

Syntax
The syntax for PicamAdvanced_UnregisterForExtrinsicIntegerValueChanged() is:

```
PICAM_API PicamAdvanced_UnregisterForExtrinsicIntegerValueChanged(
    PicamHandle device_or_accessory,
    PicamParameter parameter,
    PicamIntegerValueChangedCallback changed);
```

Input Parameters
Input parameters for PicamAdvanced_UnregisterForExtrinsicIntegerValueChanged() are:

- `device_or_accessory`: Handle for the hardware for which the callback is being unregistered.
- `parameter`: The parameter for which the callback is being unregistered.
- `changed`: The name assigned to the callback function being unregistered.

Output Parameters
There are no output parameters associated with PicamAdvanced_UnregisterForExtrinsicIntegerValueChanged().

Related APIs
For additional information, refer to the following related APIs:

- PicamAdvanced_RegisterForExtrinsicIntegerValueChanged()
7.4.9.5 *PicamAdvanced_RegisterForFloatingPointValueChanged()*

**Description**

*PicamAdvanced_RegisterForFloatingPointValueChanged()* registers a function to call when the floating point value for specified hardware parameter has been set, even if it is changed as a result of a different parameter’s value being changed.

---

**NOTE:**

Multiple functions may be registered. When this is the case, the functions are called in the order in which they have been registered.

Registered callbacks are called synchronously from within the thread in which associated parameter values are being set (i.e., called prior to returning from the set operation.)

---

**NOTE:**

Parameters whose values have changed due to external influences (e.g., representing the status of hardware,) do not result in a callback’s being called.

Call *PicamAdvanced_UnregisterForFloatingPointValueChanged()* to unregister each callback once it is not longer required.

**Syntax**

The syntax for *PicamAdvanced_RegisterForFloatingPointValueChanged()* is:

```c
PICAM_API PicamAdvanced_RegisterForFloatingPointValueChanged(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamFloatingPointValueChangedCallback changed);
```

**Input Parameters**

Input parameters for *PicamAdvanced_RegisterForFloatingPointValueChanged()* are:

- `camera_or_accessory`: Handle for the hardware for which the callback is being registered.
- `parameter`: The parameter for which the callback is being registered.
- `changed`: The name assigned to the callback function being registered.

**Output Parameters**

There are no output parameters associated with *PicamAdvanced_RegisterForFloatingPointValueChanged()*.

**Related APIs**

For additional information, refer to the following related APIs:

- *Picam_SetParameterFloatingPointValue()*
- *PicamAdvanced_UnregisterForFloatingPointValueChanged()*
7.4.9.6 PicamAdvanced_UnregisterForFloatingPointValueChanged()

Description
PicamAdvanced_UnregisterForFloatingPointValueChanged() removes the callback function so that it is no longer called when the floating point value for a specified parameter is changed.

Syntax
The syntax for PicamAdvanced_UnregisterForFloatingPointValueChanged() is:

```c
PICAM_API PicamAdvanced_UnregisterForFloatingPointValueChanged(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamFloatingPointValueChangedCallback changed);
```

Input Parameters
Input parameters for PicamAdvanced_UnregisterForFloatingPointValueChanged() are:

- `camera_or_accessory`: Handle for the hardware for which the callback is being unregistered.
- `parameter`: The parameter for which the callback is being unregistered.
- `changed`: The name assigned to the callback function being unregistered.

Output Parameters
There are no output parameters associated with PicamAdvanced_UnregisterForFloatingPointValueChanged().

Related APIs
For additional information, refer to the following related APIs:

- PicamAdvanced_RegisterForFloatingPointValueChanged().
7.4.9.7 PicamAdvanced_RegisterForExtrinsicFloatingPointValueChanged()

Description
PicamAdvanced_RegisterForExtrinsicFloatingPointValueChanged() registers a function to call when the floating point value for specified hardware parameter has changed due to external influences (e.g., representing the status of hardware).

NOTE: Multiple functions may be registered. When this is the case, the functions are called in the order in which they have been registered.

Registered callbacks are called asynchronously on another thread.

NOTE: PicamAdvanced_UnregisterForExtrinsicFloatingPointValueChanged() must be called to unregister each callback once it is no longer required.

Syntax
The syntax for PicamAdvanced_RegisterForExtrinsicFloatingPointValueChanged() is:

```c
PICAM_API PicamAdvanced_RegisterForExtrinsicFloatingPointValueChanged(
    PicamHandle device_or_accessory,
    PicamParameter parameter,
    PicamFloatingPointValueChangedCallback changed);
```

Input Parameters
Input parameters for PicamAdvanced_RegisterForExtrinsicFloatingPointValueChanged() are:

- `device_or_accessory`: Handle for the hardware for which the callback is being registered.
- `parameter`: The parameter for which the callback is being registered.
- `changed`: The name assigned to the callback function being registered.

Output Parameters
There are no output parameters associated with PicamAdvanced_RegisterForExtrinsicFloatingPointValueChanged().

Related APIs
For additional information, refer to the following related APIs:

- PicamAdvanced_UnregisterForExtrinsicFloatingPointValueChanged()
7.4.9.8 PicamAdvanced_UnregisterForExtrinsicFloatingPointValueChanged()

Description
PicamAdvanced_UnregisterForExtrinsicFloatingPointValueChanged() removes the callback function so that it is no longer called when the floating point value for a specified parameter is changed due to external influences (e.g., representing the status of hardware).

Syntax
The syntax for PicamAdvanced_UnregisterForExtrinsicFloatingPointValueChanged() is:

```c
PICAM_API PicamAdvanced_UnregisterForExtrinsicFloatingPointValueChanged(
    PicamHandle device_or_accessory,
    PicamParameter parameter,
    PicamIntegerValueChangedCallback changed);
```

Input Parameters
Input parameters for PicamAdvanced_UnregisterForExtrinsicFloatingPointValueChanged() are:

- `device_or_accessory`: Handle for the hardware for which the callback is being unregistered.
- `parameter`: The parameter for which the callback is being unregistered.
- `changed`: The name assigned to the callback function being unregistered.

Output Parameters
There are no output parameters associated with PicamAdvanced_UnregisterForExtrinsicFloatingPointValueChanged().

Related APIs
For additional information, refer to the following related APIs:

- PicamAdvanced_RegisterForExtrinsicFloatingPointValueChanged()
7.4.9.9 **PicamAdvanced_NotifyWhenStatusParameterValue()**

**Description**

`PicamAdvanced_NotifyWhenStatusParameterValue()` sets a function to call once when the value of a specified status has been met or when an error has occurred.

**NOTE:**

Multiple functions may be set. When this is the case, the functions are called in the order in which they have been set.

Set callbacks are called asynchronously from within the thread.

**Syntax**

The syntax for `PicamAdvanced_NotifyWhenStatusParameterValue()` is:

```c
PICAM_API PicamAdvanced_NotifyWhenStatusParameterValue(
    PicamHandle device_or_accessory,
    PicamParameter parameter,
    uint value,
    PicamWhenStatusParameterValueCallback when);
```

**Input Parameters**

Input parameters for `PicamAdvanced_NotifyWhenStatusParameterValue()` are:

- `device_or_accessory`: Handle for the hardware for which the callback is being set.
- `parameter`: The parameter for which the callback is being set.
  
  **NOTE:** The specified parameter must be a waitable status.
  
  Refer to `Picam_CanWaitForStatusParameter()` for additional information.

- `value`: The status value to notify when met.
- `when`: The name assigned to the callback function being set.

**Output Parameters**

There are no output parameters associated with `PicamAdvanced_NotifyWhenStatusParameterValue()`.

**Related APIs**

For additional information, refer to the following related APIs:

- `Picam_CanWaitForStatusParameter()`;
- `PicamAdvanced_CancelNotifyWhenStatusParameterValue()`.
7.4.9.10 PicamAdvanced_CancelNotifyWhenStatusParameterValue()

Description
PicamAdvanced_CancelNotifyWhenStatusParameterValue() cancels a function to call once when the value of a specified status has been met or when an error has occurred.

Syntax
The syntax for PicamAdvanced_CancelNotifyWhenStatusParameterValue() is:

```c
PICAM_API PicamAdvanced_CancelNotifyWhenStatusParameterValue(
    PicamHandle device_or_accessory,
    PicamParameter parameter,
    int value,
    PicamWhenStatusParameterValueCallback when);
```

Input Parameters
Input parameters for PicamAdvanced_CancelNotifyWhenStatusParameterValue() are:

- device_or_accessory: Handle for the hardware for which the callback is being canceled.
- parameter: The parameter for which the callback is being canceled.

**NOTE:** The specified parameter must be a waitable status. Refer to Picam_CanWaitForStatusParameter() for additional information.

- value: The status value to no longer notify when met.
- when: The name assigned to the callback function being canceled.

Output Parameters
There are no output parameters associated with PicamAdvanced_CancelNotifyWhenStatusParameterValue().

Related APIs
For additional information, refer to the following related APIs:

- Picam_CanWaitForStatusParameter();
- PicamAdvanced_NotifyWhenStatusParameterValue().
7.4.10 Shared Camera/Accessory Advanced Parameter Information APIs

This section provides programming information for camera and accessory advanced parameter information APIs.

7.4.10.1 PicamAdvanced_RegisterForIsRelevantChanged()

Description

PicamAdvanced_RegisterForIsRelevantChanged() registers a function to call when the relevance for a parameter has been changed, even if it is changed as a result of a different parameter's value being changed.

**NOTE:**

Multiple functions may be registered. When this is the case, the functions are called in the order in which they have been registered.

Registered callbacks are called synchronously from within the thread in which associated parameter values are being set (i.e., called prior to returning from the set operation.)

Call PicamAdvanced_UnregisterForIsRelevantChanged() to unregister each callback once it is not longer required.

Syntax

The syntax for PicamAdvanced_RegisterForIsRelevantChanged() is:

```c
PICAM_API PicamAdvanced_RegisterForIsRelevantChanged(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamIsRelevantChangedCallback changed);
```

Input Parameters

Input parameters for PicamAdvanced_RegisterForIsRelevantChanged() are:

- `camera_or_accessory`: Handle for the hardware for which the callback is being registered.
- `parameter`: The parameter for which the callback is being registered.
- `changed`: The name assigned to the callback function being registered.

Output Parameters

There are no output parameters associated with PicamAdvanced_RegisterForIsRelevantChanged().

Related APIs

For additional information, refer to the following related APIs:

- PicamAdvanced_UnregisterForIsRelevantChanged().
### 7.4.10.2 PicamAdvanced_UnregisterForIsRelevantChanged()

**Description**

PicamAdvanced_UnregisterForIsRelevantChanged() removes the callback function so that it is no longer called when the relevance for a parameter has been changed.

**Syntax**

The syntax for PicamAdvanced_UnregisterForIsRelevantChanged() is:

```c
PICAM_API PicamAdvanced_UnregisterForIsRelevantChanged(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamIsRelevantChangedCallback changed);
```

**Input Parameters**

Input parameters for PicamAdvanced_UnregisterForIsRelevantChanged() are:

- **camera_or_accessory**: Handle for the hardware for which the callback is being unregistered.
- **parameter**: The parameter for which the callback is being unregistered.
- **changed**: The name assigned to the callback function being unregistered.

**Output Parameters**

There are no output parameters associated with PicamAdvanced_UnregisterForIsRelevantChanged().

**Related APIs**

For additional information, refer to the following related APIs:

- PicamAdvanced_RegisterForIsRelevantChanged().
7.4.10.3 *PicamAdvanced_RegisterForValueAccessChanged()*

**Description**

`PicamAdvanced_RegisterForValueAccessChanged()` registers a function to call when the value access for a parameter has been changed, even if it is changed as a result of a different parameter's value being changed.

### NOTE:

Multiple functions may be registered. When this is the case, the functions are called in the order in which they have been registered.

Registered callbacks are called synchronously from within the thread in which associated parameter values are being set (i.e., called prior to returning from the set operation.)

Call `PicamAdvanced_UnregisterForValueAccessChanged()` to unregister each callback once it is not longer required.

**Syntax**

The syntax for `PicamAdvanced_RegisterForValueAccessChanged()` is:

```c
PICAM_API PicamAdvanced_RegisterForValueAccessChanged(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamValueAccessChangedCallback changed);
```

**Input Parameters**

Input parameters for `PicamAdvanced_RegisterForValueAccessChanged()` are:

- `camera_or_accessory`: Handle for the hardware for which the callback is being registered.
- `parameter`: The parameter for which the callback is being registered.
- `changed`: The name assigned to the callback function being registered.

**Output Parameters**

There are no output parameters associated with `PicamAdvanced_RegisterForValueAccessChanged()`.

**Related APIs**

For additional information, refer to the following related APIs:

- `PicamAdvanced_UnregisterForValueAccessChanged()`.
7.4.10.4 PicamAdvanced_UnregisterForValueAccessChanged()

Description
PicamAdvanced_UnregisterForValueAccessChanged() removes the callback function so that it is no longer called when the value access for a parameter has been changed.

Syntax
The syntax for PicamAdvanced_UnregisterForValueAccessChanged() is:

```c
PICAM_API PicamAdvanced_UnregisterForValueAccessChanged(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamValueAccessChangedCallback changed);
```

Input Parameters
Input parameters for PicamAdvanced_UnregisterForValueAccessChanged() are:

- camera_or_accessory: Handle for the hardware for which the callback is being unregistered.
- parameter: The parameter for which the callback is being unregistered.
- changed: The name assigned to the callback function being unregistered.

Output Parameters
There are no output parameters associated with PicamAdvanced_UnregisterForValueAccessChanged().

Related APIs
For additional information, refer to the following related APIs:
- PicamAdvanced_RegisterForValueAccessChanged().

7.4.10.5 PicamAdvanced_GetParameterDynamics()

Description
PicamAdvanced_GetParameterDynamics() returns the dynamics for a specified parameter.

Syntax
The syntax for PicamAdvanced_GetParameterDynamics() is:

```c
PICAM_API PicamAdvanced_GetParameterDynamics(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamDynamicsMask* dynamics);
```

Input Parameters
Input parameters for PicamAdvanced_GetParameterDynamics() are:

- camera_or_accessory: Handle for the hardware for which the dynamics information is to be returned.
- parameter: The parameter for which dynamics information is to be returned.

Output Parameters
Output parameters for PicamAdvanced_GetParameterDynamics() are:

- dynamics: Pointer to the memory location in which the dynamics information is stored.
7.4.10.6  PicamAdvanced_GetParameterExtrinsicDynamics()

Description
PicamAdvanced_GetParameterExtrinsicDynamics() returns the dynamics for a specified parameter that can change due to external influences (e.g., representing the status of hardware).

Syntax
The syntax for PicamAdvanced_GetParameterExtrinsicDynamics() is:

```c
PICAM_API PicamAdvanced_GetParameterExtrinsicDynamics(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamDynamicsMask* extrinsic);
```

Input Parameters
Input parameters for PicamAdvanced_GetParameterExtrinsicDynamics() are:

- **camera_or_accessory**: Handle for the hardware for which the extrinsic dynamics information is to be returned.
- **parameter**: The parameter for which extrinsic dynamics information is to be returned.

Output Parameters
Output parameters for PicamAdvanced_GetParameterExtrinsicDynamics() are:

- **extrinsic**: Pointer to the memory location in which the extrinsic dynamics information is stored.
7.4.11 Camera-Specific Advanced Parameter Constraints APIs

This section provides programming information for camera-specific advanced parameter constraint APIs.

7.4.11.1 PicamAdvanced_GetParameterRoisConstraints()

Description
PicamAdvanced_GetParameterRoisConstraints() returns an allocated array in which all Rois constraints for a specified camera parameter are stored.

Syntax
The syntax for PicamAdvanced_GetParameterRoisConstraints() is:

```c
PICAM_API PicamAdvanced_GetParameterRoisConstraints(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamRoisConstraint** constraint_array,
    piint* constraint_count);
```

Input Parameters
Input parameters for PicamAdvanced_GetParameterRoisConstraints() are:

- camera: Handle for the camera for which the Rois constraint information is to be returned.
- parameter: The parameter for which Rois constraint information is to be returned.

Output Parameters
Output parameters for PicamAdvanced_GetParameterRoisConstraints() are:

- constraint_array: Pointer to the array in which Rois constraint information is stored.
  
  **NOTE:** This memory is allocated by PICam and must be released by calling Picam_DestroyRoisConstraints()

- constraint_count: Pointer to the memory location in which the number of constraints is stored.

Related APIs
For additional information, refer to the following related APIs:

- Picam_DestroyRoisConstraints()
7.4.11.2 PicamAdvanced_RegisterForDependentRoisConstraintChanged()

Description
PicamAdvanced_RegisterForDependentRoisConstraintChanged() registers a function to call when any dependent Rois constraint has been changed due to the setting of a DIFFERENT parameter’s value.

NOTE: Multiple functions may be registered. When this is the case, the functions are called in the order in which they have been registered.

Registered callbacks are called synchronously from within the thread in which associated parameter values are being set (i.e., called prior to returning from the set operation.)

Call PicamAdvanced_UnregisterForDependentRoisConstraintChanged() to unregister each callback once it is not longer required.

Syntax
The syntax for PicamAdvanced_RegisterForDependentRoisConstraintChanged() is:

```
PICAM_API PicamAdvanced_RegisterForDependentRoisConstraintChanged(
    PicamHandle camera,
    PicamParameter parameter,
    PicamDependentRoisConstraintChangedCallback changed);
```

Input Parameters
Input parameters for PicamAdvanced_RegisterForDependentRoisConstraintChanged() are:

- camera: Handle for the camera for which the callback is being registered.
- parameter: The parameter for which the callback is being registered.
- changed: The name assigned to the callback function being registered.

Output Parameters
There are no output parameters associated with PicamAdvanced_RegisterForDependentRoisConstraintChanged().

Related APIs
For additional information, refer to the following related APIs:

- PicamAdvanced_UnregisterForDependentRoisConstraintChanged()
7.4.11.3 PicamAdvanced_UnregisterForDependentRoisConstraintChanged()

Description
PicamAdvanced_UnregisterForDependentRoisConstraintChanged() removes the callback function so that it is no longer called when any dependent Rois constraint has been changed.

Syntax
The syntax for PicamAdvanced_UnregisterForDependentRoisConstraintChanged() is:

    PICAM_API PicamAdvanced_UnregisterForDependentRoisConstraintChanged(
        PicamHandle camera,
        PicamParameter parameter,
        PicamDependentRoisConstraintChangedCallback changed);

Input Parameters
Input parameters for PicamAdvanced_UnregisterForDependentRoisConstraintChanged() are:

    camera:  Handle for the camera for which the callback is being unregistered.
    parameter:  The parameter for which the callback is being unregistered.
    changed:  The name assigned to the callback function being unregistered.

Output Parameters
There are no output parameters associated with PicamAdvanced_UnregisterForDependentRoisConstraintChanged().

Related APIs
For additional information, refer to the following related APIs:

    • PicamAdvanced_RegisterForDependentRoisConstraintChanged().
7.4.11.4 PicamAdvanced_GetParameterPulseConstraints()

**Description**

PicamAdvanced_GetParameterPulseConstraints() returns an allocated array in which all Pulse constraints for a specified camera parameter are stored.

**Syntax**

The syntax for PicamAdvanced_GetParameterPulseConstraints() is:

```c
PICAM_API PicamAdvanced_GetParameterPulseConstraints(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamPulseConstraint** constraint_array,
    piint* constraint_count);
```

**Input Parameters**

Input parameters for PicamAdvanced_GetParameterPulseConstraints() are:

- `camera`: Handle for the camera for which the Pulse constraint information is to be returned.
- `parameter`: The parameter for which Pulse constraint information is to be returned.

**Output Parameters**

Output parameters for PicamAdvanced_GetParameterPulseConstraints() are:

- `constraint_array`: Pointer to the array in which Pulse constraint information is stored. **NOTE:** This memory is allocated by PICam and must be released by calling Picam_DestroyPulseConstraints()
- `constraint_count`: Pointer to the memory location in which the number of constraints is stored.

**Related APIs**

For additional information, refer to the following related APIs:

- Picam_DestroyPulseConstraints()
7.4.11.5  PicamAdvanced_RegisterForDependentPulseConstraintChanged()

Description
PicamAdvanced_RegisterForDependentPulseConstraintChanged() registers a function to call when any dependent Pulse constraint has been changed due to the setting of a DIFFERENT parameter's value.

NOTE: Multiple functions may be registered. When this is the case, the functions are called in the order in which they have been registered.

Registered callbacks are called synchronously from within the thread in which associated parameter values are being set (i.e., called prior to returning from the set operation.)

Call PicamAdvanced_UnregisterForDependentPulseConstraintChanged() to unregister each callback once it is not longer required.

Syntax
The syntax for PicamAdvanced_RegisterForDependentPulseConstraintChanged() is:

PICAM_API PicamAdvanced_RegisterForDependentPulseConstraintChanged(
   PicamHandle camera,
   PicamParameter parameter,
   PicamDependentPulseConstraintChangedCallback changed);

Input Parameters
Input parameters for PicamAdvanced_RegisterForDependentPulseConstraintChanged() are:

   camera:  Handle for the camera for which the callback is being registered.
   parameter:  The parameter for which the callback is being registered.
   changed:  The name assigned to the callback function being registered.

Output Parameters
There are no output parameters associated with PicamAdvanced_RegisterForDependentPulseConstraintChanged().

Related APIs
For additional information, refer to the following related APIs:

   - PicamAdvanced_UnregisterForDependentPulseConstraintChanged().
7.4.11.6 PicamAdvanced_UnregisterForDependentPulseConstraintChanged()

Description
PicamAdvanced_UnregisterForDependentPulseConstraintChanged() removes the callback function so that it is no longer called when any dependent Pulse constraint has been changed.

Syntax
The syntax for PicamAdvanced_UnregisterForDependentPulseConstraintChanged() is:

```
PICAM_API PicamAdvanced_UnregisterForDependentPulseConstraintChanged(
    PicamHandle camera,
    PicamParameter parameter,
    PicamDependentPulseConstraintChangedCallback changed);
```

Input Parameters
Input parameters for PicamAdvanced_UnregisterForDependentPulseConstraintChanged() are:

- camera: Handle for the camera for which the callback is being unregistered.
- parameter: The parameter for which the callback is being unregistered.
- changed: The name assigned to the callback function being unregistered.

Output Parameters
There are no output parameters associated with PicamAdvanced_UnregisterForDependentPulseConstraintChanged().

Related APIs
For additional information, refer to the following related APIs:

- PicamAdvanced_RegisterForDependentPulseConstraintChanged().
7.4.11.7 PicamAdvanced_GetParameterModulationsConstraints()

**Description**

PicamAdvanced_GetParameterModulationsConstraints() returns an allocated array in which all Modulation constraints for a specified camera parameter are stored.

**Syntax**

The syntax for PicamAdvanced_GetParameterModulationsConstraints() is:

```c
PICAM_API PicamAdvanced_GetParameterModulationsConstraints(
    PicamHandle camera,
    PicamParameter parameter,
    const PicamModulationsConstraint** constraint_array,
    piint* constraint_count);
```

**Input Parameters**

Input parameters for PicamAdvanced_GetParameterModulationsConstraints() are:

- **camera**: Handle for the camera for which the Modulation constraint information is to be returned.
- **parameter**: The parameter for which Modulation constraint information is to be returned.

**Output Parameters**

Output parameters for PicamAdvanced_GetParameterModulationsConstraints() are:

- **constraint_array**: Pointer to the array in which Modulation constraint information is stored.
  
  **NOTE**: This memory is allocated by PICam and must be released by calling Picam_DestroyModulationsConstraints().
- **constraint_count**: Pointer to the memory location in which the number of constraints is stored.

**Related APIs**

For additional information, refer to the following related APIs:

- Picam_DestroyModulationsConstraints()
7.4.11.8 PicamAdvanced_RegisterForDependentModulationsConstraintChanged()

Description
PicamAdvanced_RegisterForDependentModulationsConstraintChanged() registers a function to call when any dependent Modulation constraint has been changed due to the setting of a DIFFERENT parameter's value.

**NOTE:**

Multiple functions may be registered. When this is the case, the functions are called in the order in which they have been registered.

Registered callbacks are called synchronously from within the thread in which associated parameter values are being set (i.e., called prior to returning from the set operation.)

Call PicamAdvanced_UnregisterForDependentModulationsConstraintChanged() to unregister each callback once it is no longer required.

Syntax
The syntax for PicamAdvanced_RegisterForDependentModulationsConstraintChanged() is:

```c
PICAM_API PicamAdvanced_RegisterForDependentModulationsConstraintChanged(
                        PicamHandle camera,
                        PicamParameter parameter,
                        PicamDependentModulationsConstraintChangedCallback changed);
```

Input Parameters
Input parameters for PicamAdvanced_RegisterForDependentModulationsConstraintChanged() are:

- camera: Handle for the camera for which the callback is being registered.
- parameter: The parameter for which the callback is being registered.
- changed: The name assigned to the callback function being registered.

Output Parameters
There are no output parameters associated with PicamAdvanced_RegisterForDependentModulationsConstraintChanged().

Related APIs
For additional information, refer to the following related APIs:

- PicamAdvanced_UnregisterForDependentModulationsConstraintChanged()
7.4.11.9 PicamAdvanced_UnregisterForDependentModulationsConstraintChanged()

Description

PicamAdvanced_UnregisterForDependentModulationsConstraintChanged() removes the callback function so that it is no longer called when any dependent Modulation constraint has been changed.

Syntax

The syntax for PicamAdvanced_UnregisterForDependentModulationsConstraintChanged() is:

```c
PICAM_API PicamAdvanced_UnregisterForDependentModulationsConstraintChanged(
    PicamHandle camera,
    PicamParameter parameter,
    PicamDependentModulationsConstraintChangedCallback changed);
```

Input Parameters

Input parameters for PicamAdvanced_UnregisterForDependentModulationsConstraintChanged() are:

- `camera`: Handle for the camera for which the callback is being unregistered.
- `parameter`: The parameter for which the callback is being unregistered.
- `changed`: The name assigned to the callback function being unregistered.

Output Parameters

There are no output parameters associated with PicamAdvanced_UnregisterForDependentModulationsConstraintChanged().

Related APIs

For additional information, refer to the following related APIs:

- PicamAdvanced_RegisterForDependentModulationsConstraintChanged().
7.4.12 Shared Camera/Accessory Advanced Parameter Constraints APIs

This section provides programming information for camera and accessory advanced parameter constraint APIs.

7.4.12.1 PicamAdvanced_GetParameterCollectionConstraints()

Description
PicamAdvanced_GetParameterCollectionConstraints() returns an allocated array in which all collection constraints for a specified hardware parameter are stored.

Syntax
The syntax for PicamAdvanced_GetParameterCollectionConstraints() is:

```c
PICAM_API PicamAdvanced_GetParameterCollectionConstraints(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    const PicamCollectionConstraint** constraint_array,
    pint* constraint_count);
```

Input Parameters
Input parameters for PicamAdvanced_GetParameterCollectionConstraints() are:

- camera_or_accessory: Handle for the hardware for which the collection constraint information is to be returned.
- parameter: The parameter for which collection constraint information is to be returned.

Output Parameters
Output parameters for PicamAdvanced_GetParameterCollectionConstraints() are:

- constraint_array: Pointer to the array in which collection constraint information is stored.

  **NOTE:** This memory is allocated by PICam and must be released by calling
  ```c
  Picam_DestroyCollectionConstraints()
  ```

- constraint_count: Pointer to the memory location in which the number of constraints is stored.

Related APIs
For additional information, refer to the following related APIs:

- Picam_DestroyCollectionConstraints().
7.4.12.2 PicamAdvanced_RegisterForDependentCollectionConstraintChanged()

Description

PicamAdvanced_RegisterForDependentCollectionConstraintChanged() registers a function to call when any dependent collection constraint has been changed due to the setting of a DIFFERENT parameter's value.

NOTE: Multiple functions may be registered. When this is the case, the functions are called in the order in which they have been registered.

Registered callbacks are called synchronously from within the thread in which associated parameter values are being set (i.e., called prior to returning from the set operation.)

Call PicamAdvanced_UnregisterForDependentCollectionConstraintChanged() to unregister each callback once it is no longer required.

Syntax

The syntax for PicamAdvanced_RegisterForDependentCollectionConstraintChanged() is:

```
PICAM_API PicamAdvanced_RegisterForDependentCollectionConstraintChanged(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamDependentCollectionConstraintChangedCallback changed);
```

Input Parameters

Input parameters for PicamAdvanced_RegisterForDependentCollectionConstraintChanged() are:

- camera_or_accessory: Handle for the hardware for which the callback is being registered.
- parameter: The parameter for which the callback is being registered.
- changed: The name assigned to the callback function being registered.

Output Parameters

There are no output parameters associated with PicamAdvanced_RegisterForDependentCollectionConstraintChanged().

Related APIs

For additional information, refer to the following related APIs:

- PicamAdvanced_UnregisterForDependentCollectionConstraintChanged().
### 7.4.12.3 PicamAdvanced_UnregisterForDependentCollectionConstraintChanged()

**Description**
PicamAdvanced_UnregisterForDependentCollectionConstraintChanged() removes the callback function so that it is no longer called when any dependent collection constraint has been changed.

**Syntax**
The syntax for PicamAdvanced_UnregisterForDependentCollectionConstraintChanged() is:

```c
PICAM_API PicamAdvanced_UnregisterForDependentCollectionConstraintChanged(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamDependentCollectionConstraintChangedCallback changed);
```

**Input Parameters**
Input parameters for PicamAdvanced_UnregisterForDependentCollectionConstraintChanged() are:

- `camera_or_accessory`: Handle for the hardware for which the callback is being unregistered.
- `parameter`: The parameter for which the callback is being unregistered.
- `changed`: The name assigned to the callback function being unregistered.

**Output Parameters**
There are no output parameters associated with PicamAdvanced_UnregisterForDependentCollectionConstraintChanged().

**Related APIs**
For additional information, refer to the following related APIs:
- PicamAdvanced_RegisterForDependentCollectionConstraintChanged().
7.4.12.4 PicamAdvanced_GetParameterRangeConstraints()

Description
PicamAdvanced_GetParameterRangeConstraints() returns an allocated array in which all range constraints for a specified hardware parameter are stored.

Syntax
The syntax for PicamAdvanced_GetParameterRangeConstraints() is:

```c
PICAM_API PicamAdvanced_GetParameterRangeConstraints(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    const PicamRangeConstraint** constraint_array,
    intptr* constraint_count);
```

Input Parameters
Input parameters for PicamAdvanced_GetParameterRangeConstraints() are:

- `camera_or_accessory`: Handle for the hardware for which the range constraint information is to be returned.
- `parameter`: The parameter for which range constraint information is to be returned.

Output Parameters
Output parameters for PicamAdvanced_GetParameterRangeConstraints() are:

- `constraint_array`: Pointer to the array in which range constraint information is stored.
  
  **NOTE:** This memory is allocated by PICam and must be released by calling
  `Picam_DestroyRangeConstraints()`.

- `constraint_count`: Pointer to the memory location in which the number of constraints is stored.

Related APIs
For additional information, refer to the following related APIs:

- `Picam_DestroyRangeConstraints()`.
7.4.12.5  

**PicamAdvanced_RegisterForDependentRangeConstraintChanged()**

**Description**

`PicamAdvanced_RegisterForDependentRangeConstraintChanged()` registers a function to call when any dependent range constraint has been changed due to the setting of a DIFFERENT parameter's value.

**NOTE:**

Multiple functions may be registered. When this is the case, the functions are called in the order in which they have been registered.

Registered callbacks are called synchronously from within the thread in which associated parameter values are being set (i.e., called prior to returning from the set operation.)

Call `PicamAdvanced_UnregisterForDependentRangeConstraintChanged()` to unregister each callback once it is not longer required.

**Syntax**

The syntax for `PicamAdvanced_RegisterForDependentRangeConstraintChanged()` is:

```c
PICAM_API PicamAdvanced_RegisterForDependentRangeConstraintChanged
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    PicamDependentRangeConstraintChangedCallback changed);
```

**Input Parameters**

Input parameters for `PicamAdvanced_RegisterForDependentRangeConstraintChanged()` are:

- `camera_or_accessory`: Handle for the hardware for which the callback is being registered.
- `parameter`: The parameter for which the callback is being registered.
- `changed`: The name assigned to the callback function being registered.

**Output Parameters**

There are no output parameters associated with `PicamAdvanced_RegisterForDependentRangeConstraintChanged()`.

**Related APIs**

For additional information, refer to the following related APIs:

- `PicamAdvanced_UnregisterForDependentRangeConstraintChanged()`.
7.4.12.6 PicamAdvanced_UnregisterForDependentRangeConstraintChanged()

Description
PicamAdvanced_UnregisterForDependentRangeConstraintChanged() removes the callback function so that it is no longer called when any dependent range constraint has been changed.

Syntax
The syntax for PicamAdvanced_UnregisterForDependentRangeConstraintChanged() is:

```c
PICAM_API PicamAdvanced_UnregisterForDependentRangeConstraintChanged(
    PicHandle camera_or_accessory,
    PicParameter parameter,
    PicDependentRangeConstraintChangedCallback changed);
```

Input Parameters
Input parameters for PicamAdvanced_UnregisterForDependentRangeConstraintChanged() are:

- camera_or_accessory: Handle for the hardware for which the callback is being unregistered.
- parameter: The parameter for which the callback is being unregistered.
- changed: The name assigned to the callback function being unregistered.

Output Parameters
There are no output parameters associated with PicamAdvanced_UnregisterForDependentRangeConstraintChanged().

Related APIs
For additional information, refer to the following related APIs:

- PicamAdvanced_RegisterForDependentRangeConstraintChanged().
7.4.13 Camera-Specific Advanced Commitment APIs

This section provides programming information for camera-specific advanced commitment APIs.

7.4.13.1 Picam_DestroyValidationResult()

Description
Picam_DestroyValidationResult() releases memory that has been allocated by PICam for use by result.
If result is null, calling Picam_DestroyValidationResult() has no effect.

Syntax
The syntax for Picam_DestroyValidationResult() is:

```c
PICAM_API Picam_DestroyValidationResult(
    const PicamValidationResult* result);
```

Input Parameters
Input parameters for Picam_DestroyValidationResult() are:

- `result`: Pointer to the array that is to be released.

Output Parameters
There are no output parameters associated with Picam_DestroyValidationResult().

Related Structures
For additional information, refer to the following related structures:

- PicamValidationResult.

7.4.13.2 Picam_DestroyValidationResults()

Description
Picam_DestroyValidationResults() releases memory that has been allocated by PICam for use by results.
If results is null, calling Picam_DestroyValidationResults() has no effect.

Syntax
The syntax for Picam_DestroyValidationResults() is:

```c
PICAM_API Picam_DestroyValidationResults(
    const PicamValidationResults* results);
```

Input Parameters
Input parameters for Picam_DestroyValidationResults() are:

- `results`: Pointer to the array that is to be released.

Output Parameters
There are no output parameters associated with Picam_DestroyValidationResults().

Related Structures
For additional information, refer to the following related structures:

- PicamValidationResults.
7.4.13.3 PicamAdvanced_ValidateParameter()

Description

PicamAdvanced_ValidateParameter() validates a single, specified parameter against all associated constraints and returns the results.

Syntax

The syntax for PicamAdvanced_ValidateParameter() is:

```c
PICAM_API PicamAdvanced_ValidateParameter(
    PicamHandle model,
    PicamParameter parameter,
    const PicamValidationResult** result);
```

Input Parameters

Input parameters for PicamAdvanced_ValidateParameter() are:

- `model`: Handle for the model for which the parameter is being validated.
- `parameter`: The parameter being validated.

Output Parameters

Output parameters for PicamAdvanced_ValidateParameter() are:

- `result`: Pointer to the array in which the validation results for all constraints are stored.

  NOTE: This memory is allocated by PICam and must be released by calling Picam_DestroyValidationResult().

Related APIs

For additional information, refer to the following related APIs:

- Picam_DestroyValidationResult().

Related Structures

For additional information, refer to the following related structures:

- PicamValidationResult.
7.4.13.4 PicamAdvanced_VerifyParameters()

Description
PicamAdvanced_VerifyParameters() validates all parameters against all associated constraints and returns the results.

Syntax
The syntax for PicamAdvanced_VerifyParameters() is:

```
PICAM_API PicamAdvanced_VerifyParameters(
    PicamHandle model,
    const PicamValidationResults** results);
```

Input Parameters
Input parameters for PicamAdvanced_VerifyParameters() are:

- `model`: Handle for the model for which all parameters are being validated.

Output Parameters
Output parameters for PicamAdvanced_VerifyParameters() are:

- `result`: Pointer to the array in which the validation results for all constraints are stored.

  **NOTE:** This memory is allocated by PICam and must be released by calling Picam_DestroyValidationResults().

Related APIs
For additional information, refer to the following related APIs:

- Picam_DestroyValidationResults().

Related Structures
For additional information, refer to the following related structures:

- PicamValidationResults.

7.4.13.5 Picam_DestroyDependentValidationResult()

Description
Picam_DestroyDependentValidationResult() releases memory that has been allocated by PICam for use by result.

If result is null, calling Picam_DestroyDependentValidationResult() has no effect.

Syntax
The syntax for Picam_DestroyDependentValidationResult() is:

```
PICAM_API Picam_DestroyDependentValidationResult(
    const PicamDependentValidationResult* result);
```

Input Parameters
Input parameters for Picam_DestroyDependentValidationResult() are:

- `result`: Pointer to the array that is to be released.

Output Parameters
There are no output parameters associated with Picam_DestroyDependentValidationResult().

Related APIs
For additional information, refer to the following related APIs:

- PicamDependentValidationResult.
7.4.13.6 **PicamAdvanced.ValidateDependentParameter()**

**Description**

`PicamAdvanced.ValidateDependentParameter()` validates all parameters of a specified model whose constraints are dependent on a specified parameter.

**Syntax**

The syntax for `PicamAdvanced.ValidateDependentParameter()` is:

```c
PICAM_API PicamAdvanced.ValidateDependentParameter(
    PicamHandle model,
    PicamParameter parameter,
    const PicamDependentValidationResult** result);
```

**Input Parameters**

Input parameters for `PicamAdvanced.ValidateDependentParameter()` are:

- `model`: Handle for the model for which all dependent parameters are being validated.
- `parameter`: The parameter on which all constraints being validated are dependent.

**Output Parameters**

Output parameters for `PicamAdvanced.ValidateDependentParameter()` are:

- `result`: Pointer to the array in which the validation results for all constraints are stored.

**NOTE:** This memory is allocated by PICam and must be released by calling `Picam_DestroyDependentValidationResult()`.

**Related APIs**

For additional information, refer to the following related APIs:

- `Picam_DestroyDependentValidationResult()`

7.4.13.7 **PicamAdvanced.CommitParametersToCameraDevice()**

**Description**

`PicamAdvanced.CommitParametersToCameraDevice()` attempts to configure a camera device with the set of parameter values stored in `model`.

**NOTE:**

If this action leads to a camera device error, the action fails and the camera device remains untouched.

**Syntax**

The syntax for `PicamAdvanced.CommitParametersToCameraDevice()` is:

```c
PICAM_API PicamAdvanced.CommitParametersToCameraDevice(
    PicamHandle model);
```

**Input Parameters**

Input parameters for `PicamAdvanced.CommitParametersToCameraDevice()` are:

- `model`: Handle for the model for which all parameters are to be committed.

**Output Parameters**

There are no output parameters associated with `PicamAdvanced.CommitParametersToCameraDevice()`.
7.4.13.8 PicamAdvanced_RefreshParameterFromCameraDevice()

Description
PicamAdvanced_RefreshParameterFromCameraDevice() updates a single parameter's value stored in model with the value from the connected camera device.

Syntax
The syntax for PicamAdvanced_RefreshParameterFromCameraDevice() is:

```c
PICAM_API PicamAdvanced_RefreshParameterFromCameraDevice(
    PicamHandle model,
    PicamParameter parameter);
```

Input Parameters
Input parameters for PicamAdvanced_RefreshParameterFromCameraDevice() are:

- model: Handle for the model for which the parameter's value is to be overwritten.
- parameter: The parameter for which the value is to be overwritten.

Output Parameters
There are no output parameters associated with PicamAdvanced_RefreshParameterFromCameraDevice().

7.4.13.9 PicamAdvanced_RefreshParametersFromCameraDevice()

Description
PicamAdvanced_RefreshParametersFromCameraDevice() updates all parameter values stored in model with values from the connected camera device.

Syntax
The syntax for PicamAdvanced_RefreshParametersFromCameraDevice() is:

```c
PICAM_API PicamAdvanced_RefreshParametersFromCameraDevice(
    PicamHandle model);
```

Input Parameters
Input parameters for PicamAdvanced_RefreshParametersFromCameraDevice() are:

- model: Handle for the model for which all parameter values are to be overwritten.

Output Parameters
There are no output parameters associated with PicamAdvanced_RefreshParametersFromCameraDevice().
7.4.14 Camera-Specific Advanced Acquisition Setup APIs

This section provides programming information about camera-specific advanced acquisition setup APIs.

7.4.14.1 PicamAdvanced_GetAcquisitionBuffer()

Description
PicamAdvanced_GetAcquisitionBuffer() returns the user-allocated buffer to be used during data acquisition.

Syntax
The syntax for PicamAdvanced_GetAcquisitionBuffer() is:

```c
PICAM_API PicamAdvanced_GetAcquisitionBuffer(
    PicamHandle device,
    PicamAcquisitionBuffer* buffer);
```

Input Parameters
Input parameters for PicamAdvanced_GetAcquisitionBuffer() are:

- device: Handle for the device to which the data acquisition buffer is allocated.

Output Parameters
Output parameters for PicamAdvanced_GetAcquisitionBuffer() are:

- buffer: Pointer to the user-allocated data acquisition buffer. If no buffer has been created/allocated, this points to a null buffer with zero size.

7.4.14.2 PicamAdvanced_SetAcquisitionBuffer()

Description
PicamAdvanced_SetAcquisitionBuffer() assigns a user-allocated buffer to a specific device.

Syntax
The syntax for PicamAdvanced_SetAcquisitionBuffer() is:

```c
PICAM_API PicamAdvanced_SetAcquisitionBuffer(
    PicamHandle device,
    const PicamAcquisitionBuffer* buffer);
```

Input Parameters
Input parameters for PicamAdvanced_SetAcquisitionBuffer() are:

- device: Handle for the device to which the data acquisition buffer is to be allocated.

Output Parameters
Output parameters for PicamAdvanced_SetAcquisitionBuffer() are:

- buffer: Pointer to the user-allocated data acquisition buffer. To clear this buffer, point to null with zero size. This buffer can be used to create a circular buffer.
7.4.15 Camera-Specific Advanced Acquisition Notification APIs

This section provides programming information for camera-specific advanced acquisition notification APIs.

7.4.15.1 PicamAdvanced_RegisterForAcquisitionUpdated()

Description
PicamAdvanced_RegisterForAcquisitionUpdated() registers a function to call during data acquisition when:
- New data are available, or
- A change in acquisition status has occurred.

**NOTE:** Multiple functions may be registered. When this is the case, the functions are called in the order in which they have been registered.

Callbacks are called asynchronously from another thread, but are serialized on that thread. This means that additional notifications do not occur simultaneously, but occur after each callback returns.

Call PicamAdvanced_UnregisterForAcquisitionUpdated() to unregister each callback once it is no longer required.

Syntax
The syntax for PicamAdvanced_RegisterForAcquisitionUpdated() is:

```c
PICAM_API PicamAdvanced_RegisterForAcquisitionUpdated(  
    PicamHandle device,  
    PicamAcquisitionUpdatedCallback updated);
```

Input Parameters
Input parameters for PicamAdvanced_RegisterForAcquisitionUpdated() are:

- device: Handle for the device for which the callback is being registered.
- changed: The name assigned to the callback function being registered.

Output Parameters
There are no output parameters associated with PicamAdvanced_RegisterForAcquisitionUpdated().

Related APIs
For additional information, refer to the following related APIs:

- PicamAdvanced_UnregisterForAcquisitionUpdated().
7.4.15.2 PicamAdvanced_UnregisterForAcquisitionUpdated()

Description
PicamAdvanced_UnregisterForAcquisitionUpdated() removes the callback function so that it is no longer called during data acquisition.

Syntax
The syntax for PicamAdvanced_UnregisterForAcquisitionUpdated() is:

```
PICAM_API PicamAdvanced_UnregisterForAcquisitionUpdated(
    PicamHandle device,
    PicamAcquisitionUpdatedCallback updated);
```

Input Parameters
Input parameters for PicamAdvanced_UnregisterForAcquisitionUpdated() are:

- device: Handle for the device for which the callback is being unregistered.
- changed: The name assigned to the callback function being unregistered.

Output Parameters
There are no output parameters associated with PicamAdvanced_UnregisterForAcquisitionUpdated().

Related APIs
For additional information, refer to the following related APIs:

- PicamAdvanced_RegisterForAcquisitionUpdated()
7.4.16 Camera-Specific Advanced Acquisition State Notification APIs

This section provides programming information for camera-specific advanced acquisition state notification APIs.

7.4.16.1 PicamAdvanced_CanRegisterForAcquisitionStateUpdated()

Description
PicamAdvanced_CanRegisterForAcquisitionStateUpdated() determines if an acquisition state can be detected.

Syntax
The syntax for PicamAdvanced_CanRegisterForAcquisitionStateUpdated() is:

```c
PICAM_API PicamAdvanced_CanRegisterForAcquisitionStateUpdated(
    PicamHandle device,
    PicamAcquisitionState state,
    pibln* detectable);
```

Input Parameters
Input parameters for PicamAdvanced_CanRegisterForAcquisitionStateUpdated() are:

- device: Handle for the device under test.
- state: Specifies the acquisition state to be queried for detectability.

Output Parameters
Output parameters for PicamAdvanced_CanRegisterForAcquisitionStateUpdated() are:

- detectable: Pointer to the test results. Indicates if the specified acquisition state is detectable. Valid values are:
  - TRUE: Indicates the specified acquisition state is detectable.
  - FALSE: Indicates the specified acquisition state is not detectable.
7.4.16.2 PicamAdvanced_RegisterForAcquisitionStateUpdated()

Description
PicamAdvanced_RegisterForAcquisitionStateUpdated() registers a function to call during data acquisition when the camera transitions to an acquisition state.

NOTE: Multiple functions may be registered. When this is the case, the functions are called in the order in which they have been registered.

Callbacks are called asynchronously from another thread, but are serialized on that thread. This means that additional notifications do not occur simultaneously, but occur after each callback returns.

Syntax
The syntax for PicamAdvanced_RegisterForAcquisitionStateUpdated() is:

```
PICAM_API PicamAdvanced_RegisterForAcquisitionStateUpdated(
    PicamHandle device,
    PicamAcquisitionState state,
    PicamAcquisitionStateUpdatedCallback updated);
```

Input Parameters
Input parameters for PicamAdvanced_RegisterForAcquisitionStateUpdated() are:

- `device`: Handle for the device for which the callback is being registered.
- `state`: Specifies the acquisition state to detect.

Output Parameters
There are no output parameters associated with PicamAdvanced_RegisterForAcquisitionStateUpdated().

Related APIs
For additional information, refer to the following related APIs:
- PicamAdvanced_UnregisterForAcquisitionStateUpdated()
7.4.16.3  PicamAdvanced_UnregisterForAcquisitionStateUpdated()

Description
PicamAdvanced_UnregisterForAcquisitionStateUpdated() removes the callback function so that it is no longer called during data acquisition.

Syntax
The syntax for PicamAdvanced_UnregisterForAcquisitionStateUpdated() is:

```c
PICAM_API PicamAdvanced_UnregisterForAcquisitionStateUpdated(
    PicamHandle device,
    PicamAcquisitionState state,
    PicamAcquisitionStateUpdatedCallback updated);
```

Input Parameters
Input parameters for PicamAdvanced_UnregisterForAcquisitionStateUpdated() are:

- **device**: Handle for the device for which the callback is being unregistered.
- **state**: Specifies the acquisition state to detect no longer.

Output Parameters
There are no output parameters associated with PicamAdvanced_UnregisterForAcquisitionStateUpdated().

Related APIs
For additional information, refer to the following related APIs:

- PicamAdvanced_RegisterForAcquisitionStateUpdated()
7.4.17 Camera-Specific Advanced Acquisition Control APIs

This section provides programming information for camera-specific advanced acquisition control APIs.

7.4.17.1 PicamAdvanced_HasAcquisitionBufferOverrun()

Description
PicamAdvanced_HasAcquisitionBufferOverrun() determines if a user-allocated circular buffer has overflowed.

Syntax
The syntax for PicamAdvanced_HasAcquisitionBufferOverrun() is:

```c
PICAM_API PicamAdvanced_HasAcquisitionBufferOverrun(
    PicamHandle device,
    pibln* overran);
```

Input Parameters
Input parameters for PicamAdvanced_HasAcquisitionBufferOverrun() are:

- `device`: Handle for the device for which the status of the associated user-allocated circular buffer is being tested.

Output Parameters
Output parameters for PicamAdvanced_HasAcquisitionBufferOverrun() are:

- `overran`: Pointer to the results. Indicates if the user-allocated circular data buffer has overflowed. Valid values are:
  - `TRUE`: Indicates that the buffer has overflowed.
  - `FALSE`: Indicates that the buffer has not overflowed.
7.4.17.2 PicamAdvanced_CanClearReadoutCountOnline()

Description
PicamAdvanced_CanClearReadoutCountOnline() indicates if it is possible to set the readout count to 0 [zero] while the camera is running.

Syntax
The syntax for PicamAdvanced_CanClearReadoutCountOnline() is:

```
PICAM_API PicamAdvanced_CanClearReadoutCountOnline(
    PicamHandle device,
    pibln* clearable);
```

Input Parameters
Input parameters for PicamAdvanced_CanClearReadoutCountOnline() are:

```
device: Handle for the device for which the ability to manipulate the readout count online is being tested.
```

Output Parameters
Output parameters for PicamAdvanced_CanClearReadoutCountOnline() are:

```
clearable: Pointer to the results.
Indicates if it is possible to clear the readout count online.
Valid values are:

- TRUE
  Indicates that it is possible to clear the readout count online.
- FALSE
  Indicates that the readout count can never be cleared online.
```
7.4.17.3 PicamAdvanced_ClearReadoutCountOnline()

Description
PicamAdvanced_ClearReadoutCountOnline() tries to set the readout count to 0 [zero] while the camera is running.

Syntax
The syntax for PicamAdvanced_ClearReadoutCountOnline() is:

   PICAM_API PicamAdvanced_ClearReadoutCountOnline(
       PicamHandle device,
       pibln* cleared);

Input Parameters
Input parameters for PicamAdvanced_ClearReadoutCountOnline() are:

   device: Handle for the device for which the status of the readout count is being tested.

Output Parameters
Output parameters for PicamAdvanced_ClearReadoutCountOnline() are:

   cleared: Pointer to the results.
   Indicates if the readout count has been cleared online.
   Valid values are:
   • TRUE
     Indicates that the readout count has been cleared online.
   • FALSE
     Indicates that the readout count has not been cleared online.

NOTE: There is an inherent race between clearing the readout count and the camera stopping when it has acquired the number of readouts. It is advised to check the value of cleared to determine the actual effect of the function.
Chapter 8: EM Calibration APIs

This chapter provides information about the EM gain calibration APIs. All functions, data definitions, and structures are located in the `picam_em_calibration.h` file.

NOTE: The information and APIs described within this chapter are NOT applicable to emICCD cameras.

8.1 EM Calibration Applications

Each ProEM camera is factory-calibrated for linear EM Gain. Over time, however, aging of the EMCCD array may degrade gain linearity. Because aging appears to be a strong function of the amount of charge that flows through the multiplication register, users who consistently operate the camera at high gain at high light levels may need to recalculate EM gain more frequently than those who are looking at lower light levels at lower gain.

To compensate for aging, each ProEM includes a built-in shutter (either manual or electro-mechanical) and a light source that allows users to perform on-demand EM Gain Calibration using a calibration application. Once the EM gain calibration has been performed, the gain value entered in the software by the user will be the actual multiplication gain applied to the input signal.

CAUTION! When calibrating a ProEM camera with a manual shutter (e.g., ProEM:1600,) the shutter MUST be closed manually before launching any calibration program.

This is not necessary for a camera with an internal electro-mechanical shutter because the program will automatically close the shutter before beginning the calibration.

PICam users have two options available when creating an EM Calibration application:

- Build the sample code `EMGainCalibration.exe` that is included with PICam. This option requires the least amount of development time and overhead since `EMGainCalibration.exe` is a fully functional application once it has been built.

REFERENCES: Refer to Appendix B, EM Gain Calibration Code Sample, for additional information about using the sample code.

- Create a custom EM Calibration application using the API routines, structures, and callbacks described in this chapter. When building a custom application, the `EMGainCalibration.exe` sample code included with PICam is a good resource for the developer when learning about the EM Calibration API library.
8.2 Structure Definitions

This section provides programming information about PICam structure definitions.

8.2.1 EM Calibration Structures

This section provides detailed programming information about the following EM Calibration data structures:

- PicamEMCalibrationDate

8.2.1.1 PicamEMCalibrationDate

Description

PicamEMCalibrationDate specifies the calibration date.

Structure Definition

The structure definition for PicamEMCalibrationDate is:

```c
typedef struct PicamEMCalibrationDate
{
    puint year;
    puint month;
    puint day;
} PicamEMCalibrationDate;
```

Variable Definitions

The variables required by PicamEMCalibrationDate are:

- **year**: The year as an integer (e.g., 2011.)
- **month**: The month as an integer.
  
  Valid values are from [1...12], inclusive.
  
  For example, 3 = March.
- **day**: The day of the month as an integer.
  
  Valid values are from [1...31], inclusive.
8.3 Callback Functions

This section provides programming information about callback functions used by PICam.

8.3.1 EM Calibration

This section provides information about the following callback functions:

- PicamEMCalibrationCallback().

8.3.1.1 PicamEMCalibrationCallback()

Description

PicamEMCalibrationCallback() is the callback function for EM calibration progress and/or cancellation.

Syntax

The syntax for PicamEMCalibrationCallback() is:

```c
typedef pibln (PIL_CALL* PicamEMCalibrationCallback)(
    PicamHandle calibration,
    piflt progress,
    void* user_state);
```

Input Parameters

The input parameters for PicamEMCalibrationCallback() are:

- calibration: Handle for the camera which is being calibrated.
- progress: This is the percentage of calibration completion. Valid values are [0...100], inclusive.
- user_state: User-supplied data provided when calibration is started.

Return Values

Return values for PicamEMCalibrationCallback() are:

- TRUE: Calibration continues.
- FALSE: Cancels the calibration.
8.4 Programmers' Reference for EM Calibration APIs

This section provides a detailed programmers' reference guide for the following EM Calibration APIs:

- **EM Calibration Access APIs**
  - PicamEMCalibration_OpenCalibration()
  - PicamEMCalibration_CloseCalibration()
  - PicamEMCalibration_GetOpenCalibrations()
  - PicamEMCalibration_GetCameraID()

- **EM Calibration Parameter Value APIs**
  - PicamEMCalibration_GetCalibrationDate()
  - PicamEMCalibration_ReadSensorTemperatureReading()
  - PicamEMCalibration_ReadSensorTemperatureStatus()
  - PicamEMCalibration_GetSensorTemperatureSetPoint()
  - PicamEMCalibration_SetSensorTemperatureSetPoint()

- **EM Calibration Parameter Constraints APIs**
  - PicamEMCalibration_GetSensorTemperatureSetPointConstraint()

- **EM Calibration APIs**
  - PicamEMCalibration_SetSensorTemperatureSetPoint()

8.4.1 EM Calibration Access APIs

This section provides programming information about EM Calibration Access APIs.

8.4.1.1 PicamEMCalibration_OpenCalibration()

**Description**

PicamEMCalibration_OpenCalibration() opens a camera for calibration and returns a handle to it.

**NOTE:**

Opening a camera for calibration is mutually exclusive with opening it for normal usage.

**Syntax**

The syntax for PicamEMCalibration_OpenCalibration() is:

```c
PICAM_API PicamEMCalibration_OpenCalibration(
    const PicamCameraID* id,
    PicamHandle* calibration);
```

**Input Parameters**

Input parameters for PicamEMCalibration_OpenCalibration() are:

- `id`: Pointer to the camera id for the camera being calibrated.

**Output Parameters**

Output parameters for PicamEMCalibration_OpenCalibration() are:

- `calibration`: Pointer to the handle assigned to the camera that will be calibrated.

**Related APIs**

For additional information, refer to the following related APIs:

- PicamEMCalibration_CloseCalibration().
8.4.1.2 PicamEMCalibration_CloseCalibration()

Description
PicamEMCalibration_CloseCalibration() releases all resources that have been associated with a specified calibration process.

Syntax
The syntax for PicamEMCalibration_CloseCalibration() is:

```
PICAM_API PicamEMCalibration_CloseCalibration( 
    PicamHandle calibration);
```

Input Parameters
Input parameters for PicamEMCalibration_CloseCalibration() are:

- `calibration`: Pointer to the handle for the calibration process for which resources are to be released.

Output Parameters
There are no output parameters associated with PicamEMCalibration_CloseCalibration().

Related APIs
For additional information, refer to the following related APIs:
- PicamEMCalibration_OpenCalibration().

8.4.1.3 PicamEMCalibration_GetOpenCalibrations()

Description
PicamEMCalibration_GetOpenCalibrations() returns an allocated array of open calibration handles.

Syntax
The syntax for PicamEMCalibration_GetOpenCalibrations() is:

```
PICAM_API PicamEMCalibration_GetOpenCalibrations( 
    const PicamHandle** calibrations_array, 
    pint* calibrations_count);
```

Input Parameters
There are no input parameters associated with PicamEMCalibration_GetOpenCalibrations().

Output Parameters
Output parameters for PicamEMCalibration_GetOpenCalibrations() are:

- `calibrations_array`: Pointer to the array of handles to open calibration processes. Returns null when there are no open calibration processes.
- `calibrations_count`: Pointer to the memory location in which the number of open calibration processes is stored. Returns 0 when there are no open calibration processes.

**NOTE:** This memory is allocated by PICam and must be released by calling Picam_DestroyHandles().
8.4.1.4 PicamEMCalibration_GetCameraID()

Description
PicamEMCalibration_GetCameraID() returns the camera id associated with a specified calibration process.

Syntax
The syntax for PicamEMCalibration_GetCameraID() is:

```c
PICAM_API PicamEMCalibration_GetCameraID(
    PicamHandle calibration,
    PicamCameraID* id);
```

Input Parameters
Input parameters for PicamEMCalibration_GetCameraID() are:

- calibration: Handle associated with the calibration process for which the associated camera is to be determined.

Output Parameters
Output parameters for PicamEMCalibration_GetCameraID() are:

- id: Pointer to the ID of the camera associated with the specified calibration process.
8.4.2 EM Calibration Parameter Value APIs

This section provides programming information about EM Calibration Parameter Value APIs.

8.4.2.1 PicamEMCalibration_GetCalibrationDate()

Description
PicamEMCalibration_GetCalibrationDate() returns the date of the most recent successful calibration.

Syntax
The syntax for PicamEMCalibration_GetCalibrationDate() is:

    PICAM_API PicamEMCalibration_GetCalibrationDate(
        PicamHandle calibration,
        PicamEMCalibrationDate* value);

Input Parameters
Input parameters for PicamEMCalibration_GetCalibrationDate() are:

    calibration: Handle of the camera for which the calibration date is to be determined.

Output Parameters
Output parameters for PicamEMCalibration_GetCalibrationDate() are:

    value: Pointer to the calibration date.

Related Structures
For additional information, refer to the following related APIs:

- PicamEMCalibrationDate.

8.4.2.2 PicamEMCalibration_ReadSensorTemperatureReading()

Description
PicamEMCalibration_ReadSensorTemperatureReading() returns the current sensor temperature, in degrees Celsius, for a specified camera.

Syntax
The syntax for PicamEMCalibration_ReadSensorTemperatureReading() is:

    PICAM_API PicamEMCalibration_ReadSensorTemperatureReading(
        PicamHandle calibration,
        piflt* value);

Input Parameters
Input parameters for PicamEMCalibration_ReadSensorTemperatureReading() are:

    calibration: Handle of the camera for which the sensor temperature is to be determined.

Output Parameters
Output parameters for PicamEMCalibration_ReadSensorTemperatureReading() are:

    value: Pointer to the memory location in which the sensor temperature is stored.
8.4.2.3 PicamEMCalibration_ReadSensorTemperatureStatus()

Description
PicamEMCalibration_ReadSensorTemperatureStatus() returns the status of the current sensor temperature for a specified camera.

NOTE: Calibration cannot begin until the status of the current sensor temperature is locked.

Syntax
The syntax for PicamEMCalibration_ReadSensorTemperatureStatus() is:

```
PICAM_API PicamEMCalibration_ReadSensorTemperatureStatus(
    PicamHandle calibration,
    PicamSensorTemperatureStatus* value);
```

Input Parameters
Input parameters for PicamEMCalibration_ReadSensorTemperatureStatus() are:

- calibration: Handle of the camera for which the status of the sensor temperature is to be determined.

Output Parameters
Output parameters for PicamEMCalibration_ReadSensorTemperatureStatus() are:

- value: Pointer to the memory location in which the status information is stored.

Related Structures
For additional information, refer to the following related APIs:
- PicamSensorTemperatureStatus.
8.4.2.4 PicamEMCalibration_GetSensorTemperatureSetPoint()

Description
PicamEMCalibration_GetSensorTemperatureSetPoint() returns the temperature set point that has been programmed for a specified camera.

Syntax
The syntax for PicamEMCalibration_GetSensorTemperatureSetPoint() is:

PICAM_API PicamEMCalibration_GetSensorTemperatureSetPoint(
    PicamHandle calibration,
    piflt* value);

Input Parameters
Input parameters for PicamEMCalibration_GetSensorTemperatureSetPoint() are:

    calibration: Handle of the camera for which the programmed temperature set point is to be determined.

Output Parameters
Output parameters for PicamEMCalibration_GetSensorTemperatureSetPoint() are:

    value: Pointer to the memory location in which the set point information is stored.

Related APIs
For additional information, refer to the following related APIs:
- PicamEMCalibration_SetSensorTemperatureSetPoint().

8.4.2.5 PicamEMCalibration_SetSensorTemperatureSetPoint()

Description
PicamEMCalibration_SetSensorTemperatureSetPoint() configures the sensor temperature set point for a specified camera to a specified value.

Syntax
The syntax for PicamEMCalibration_SetSensorTemperatureSetPoint() is:

PICAM_API PicamEMCalibration_SetSensorTemperatureSetPoint(
    PicamHandle calibration,
    piflt value);

Input Parameters
Input parameters for PicamEMCalibration_SetSensorTemperatureSetPoint() are:

    calibration: Handle of the camera for which the temperature set point is to be programmed.
    value: The desired temperature set point, in degrees Celsius.

Output Parameters
There are no output parameters associated with PicamEMCalibration_SetSensorTemperatureSetPoint().
8.4.3 EM Calibration Parameter Constraints APIs

This section provides programming information about EM Calibration Parameter Constraint APIs.

8.4.3.1 PicamEMCalibration_GetSensorTemperatureSetPointConstraint()

Description
PicamEMCalibration_GetSensorTemperatureSetPointConstraint() returns an allocated constraint in which the set of valid temperature set points, in degrees Celsius, for a specified camera is stored.

Syntax
The syntax for PicamEMCalibration_GetSensorTemperatureSetPointConstraint() is:

```c
PICAM_API PicamEMCalibration_GetSensorTemperatureSetPointConstraint(
    PicamHandle calibration,
    const PicamRangeConstraint** constraint);
```

Input Parameters
Input parameters for PicamEMCalibration_GetSensorTemperatureSetPointConstraint() are:

- calibration: Handle for the camera for which the valid range of temperature set points is to be returned.

Output Parameters
Output parameters for PicamEMCalibration_GetSensorTemperatureSetPointConstraint() are:

- constraint: Pointer to the allocated constraint in which the set of valid temperature set points is stored.

**NOTE:** This memory is allocated by PICam and must be released by calling

```c
Picam_DestroyCollectionConstraints()
```

Related APIs
For additional information, refer to the following related APIs:

- Picam_DestroyHandles().

Related Structures
For additional information, refer to the following related APIs:

- PicamRangeConstraint.
8.4.4 EM Calibration APIs

This section provides programming information about EM Calibration APIs.

8.4.4.1 PicamEMCalibration_Calibrate()

Description
PicamEMCalibration_Calibrate() calibrates the EM Gain for a specified camera.

NOTE: Calibration cannot begin until the status of the current sensor temperature is locked.

NOTE: If calibration is canceled (via the use of the callback function PicamEMCalibrationCallback()) this function returns PicamError_OperationCanceled.

Syntax
The syntax for PicamEMCalibration_Calibrate() is:

```c
PICAM_API PicamEMCalibration_Calibrate(
    PicamHandle calibration,
    PicamEMCalibrationCallback callback,
    void* user_state);
```

Input Parameters
Input parameters for PicamEMCalibration_Calibrate() are:

- calibration: Handle for the camera for which EM calibration is to be performed.
- callback: Optional Callback function. Specifying a Callback provides additional functionality, such as:
  - The ability to cancel a calibration process;
  - The ability to obtain calibration progress information.
- user_state: [optional] When used, allows the caller to provide user-defined data to the callback function.

Output Parameters
There are no output parameters associated with PicamEMCalibration_Calibrate().

Related APIs
For additional information, refer to the following related APIs:

- PicamEMCalibration_ReadSensorTemperatureStatus();
- PicamEMCalibrationCallback().
This page is intentionally blank.
Appendix A: Available Parameters

**NOTES:**

1. Parameters are listed using a truncated version of their names (e.g., the PicamParameter_prefix has been dropped.)
   
   For example, the parameter named PicamParameter_ExposureTime is listed as ExposureTime.
   
   2. An asterisk indicates that the parameter does not apply to all members of a camera family.

<table>
<thead>
<tr>
<th>Value Types</th>
<th>Constraint Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>F = Floating Point</td>
<td>M = Modulations</td>
</tr>
<tr>
<td>E = Enumeration</td>
<td>R = Region of Interest</td>
</tr>
<tr>
<td>B = Boolean</td>
<td>P = Pulse</td>
</tr>
<tr>
<td>I = Integer</td>
<td>L = Large Integer</td>
</tr>
</tbody>
</table>

A.1 Camera Parameter Information

Refer to Table A-2 for the list of available Camera parameters

**NOTE:**

Information for PI-MTE and PI-MTE3 is provided in separate columns because their underlying architectures differ significantly.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Read Only</th>
<th>Value Type</th>
<th>Constraint Type</th>
<th>BLAZE</th>
<th>FERGIE/ISO-81</th>
<th>KURO</th>
<th>NIRvanaLN</th>
<th>PI-MAX 3/4</th>
<th>PI-MTE</th>
<th>PI-MTE3</th>
<th>ProEM/ + / -HS</th>
<th>PyLoN</th>
<th>PyLoN-IR</th>
<th>SOPHIA</th>
<th>Quad-RO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shutter Timing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ActiveShutter</td>
<td>E</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>ExposureTime</td>
<td>F</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Read Only</td>
<td>Value Type</td>
<td>Constraint Type</td>
<td>BLAZE</td>
<td>FERGIE/ISO-81</td>
<td>KURO</td>
<td>NIRvana LN</td>
<td>PI-MAX 3/4</td>
<td>PI-MTE</td>
<td>PI-MTE3</td>
<td>PIoNIR/NIRvana/ST</td>
<td>PIXIS</td>
<td>ProEM +/−HS</td>
<td>Pylon</td>
<td>Pylon-IR</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------</td>
<td>------------</td>
<td>----------------</td>
<td>-------</td>
<td>---------------</td>
<td>------</td>
<td>------------</td>
<td>------------</td>
<td>--------</td>
<td>---------</td>
<td>-------------------</td>
<td>-------</td>
<td>-------------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>ExternalShutter Status</td>
<td>✓</td>
<td>E</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InactiveShutter TimingModeResult</td>
<td>✓</td>
<td>E</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InternalShutter Status</td>
<td>✓</td>
<td>E</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InternalShutter Type</td>
<td>✓</td>
<td>E</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ShutterClosingDelay</td>
<td>F</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ShutterDelay Resolution</td>
<td>F</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ShutterOpeningDelay</td>
<td>F</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ShutterTimingMode</td>
<td>E</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Gating</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DifEndingGate</td>
<td>P</td>
<td>P</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DifStartingGate</td>
<td>P</td>
<td>P</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GatingMode</td>
<td>E</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RepetitiveGate</td>
<td>P</td>
<td>P</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SequentialEndingGate</td>
<td>P</td>
<td>P</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SequentialGateStep Count</td>
<td>L</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SequentialGateStep Iterations</td>
<td>L</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SequentialStarting Gate</td>
<td>P</td>
<td>P</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intensifier</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BracketGating</td>
<td>B</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CustomModulation Sequence</td>
<td>M</td>
<td>M</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>EMIccdGain</td>
<td>I</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>EMIccdGainControl Mode</td>
<td>E</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
### Table A-2: Parameter Information and Camera Support (Sheet 3 of 9)

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Read Only</th>
<th>Value Type</th>
<th>Constraint Type</th>
<th>BLAZE</th>
<th>FERGIE/ISO-81</th>
<th>KURO</th>
<th>NIRvana LN</th>
<th>PI-MAX 3/4</th>
<th>PI-MTE</th>
<th>PI-MTE3</th>
<th>PIoNIR/Rvana/ST</th>
<th>PIXIS</th>
<th>ProEM/+/−HS</th>
<th>PyLoN</th>
<th>PyLoN-IR</th>
<th>SOPHIA</th>
<th>Quad-RO</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnableIntensifier</td>
<td>✓</td>
<td>B</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableModulation</td>
<td>✓</td>
<td>B</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GatingSpeed</td>
<td>✓</td>
<td>E</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IntensifierDiameter</td>
<td>✓</td>
<td>F</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IntensifierGain</td>
<td>✓</td>
<td>I</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IntensifierOptions</td>
<td>✓</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IntensifierStatus</td>
<td>✓</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ModulationDuration</td>
<td>F</td>
<td>R</td>
<td>✓*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ModulationFrequency</td>
<td>F</td>
<td>R</td>
<td>✓*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PhosphorDecayDelay</td>
<td>F</td>
<td>R</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PhosphorDecayDelay Resolution</td>
<td>F</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PhosphorType</td>
<td>✓</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Photocathode Sensitivity</td>
<td>✓</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RepetitiveModulation Phase</td>
<td>F</td>
<td>R</td>
<td>✓*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SequentialEnding ModulationPhase</td>
<td>F</td>
<td>R</td>
<td>✓*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SequentialStarting ModulationPhase</td>
<td>F</td>
<td>R</td>
<td>✓*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Analog to Digital Conversion

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Read Only</th>
<th>Value Type</th>
<th>Constraint Type</th>
<th>BLAZE</th>
<th>FERGIE/ISO-81</th>
<th>KURO</th>
<th>NIRvana LN</th>
<th>PI-MAX 3/4</th>
<th>PI-MTE</th>
<th>PI-MTE3</th>
<th>PIoNIR/Rvana/ST</th>
<th>PIXIS</th>
<th>ProEM/+/−HS</th>
<th>PyLoN</th>
<th>PyLoN-IR</th>
<th>SOPHIA</th>
<th>Quad-RO</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdcAnalogGain</td>
<td>E</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AdcBitDepth</td>
<td>I</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AdcEMGain</td>
<td>I</td>
<td>R</td>
<td>✓*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AdcQuality</td>
<td>E</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AdcSpeed</td>
<td>F</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CorrectPixelBias</td>
<td>B</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Read Only</td>
<td>Value Type</td>
<td>Constraint Type</td>
<td>BLAZE</td>
<td>FERGIE/ISO-81</td>
<td>KURO</td>
<td>NIRvana LN</td>
<td>PI-MAX 3/4</td>
<td>PI-MTE</td>
<td>PI-MTE3</td>
<td>PLoNIR/NIRvana/ST</td>
<td>PIoNIR/NIRvana/ST</td>
<td>PyLoN</td>
<td>PyLoN-IR</td>
<td>SOPHIA</td>
<td>Quad-RO</td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------</td>
<td>------------</td>
<td>-----------------</td>
<td>-------</td>
<td>---------------</td>
<td>------</td>
<td>------------</td>
<td>------------</td>
<td>--------</td>
<td>--------</td>
<td>-------------------</td>
<td>------------------</td>
<td>--------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>AnticipateTrigger</td>
<td>B</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AuxOutput</td>
<td>P</td>
<td>P</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DelayFromPreTrigger</td>
<td>F</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableAuxOutput</td>
<td>B</td>
<td>C</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableModulation</td>
<td>B</td>
<td>C</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OutputSignal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableSyncMaster</td>
<td>B</td>
<td>C</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InvertOutputSignal</td>
<td>B</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InvertOutputSignal2</td>
<td>B</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ModulationOutput</td>
<td>F</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SignalAmplitude</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableSyncMaster2Delay</td>
<td>F</td>
<td>R</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TriggerCoupling</td>
<td>E</td>
<td>C</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TridderDelay</td>
<td>F</td>
<td>R</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TriggerDetermination</td>
<td>E</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TriggerFrequency</td>
<td>F</td>
<td>R</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TriggerResponse</td>
<td>E</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TriggerSource</td>
<td>E</td>
<td>C</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TriggerTermination</td>
<td>E</td>
<td>C</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TriggerThreshold</td>
<td>F</td>
<td>R</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table A-2: Parameter Information and Camera Support (Sheet 5 of 9)

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Read Only</th>
<th>Value Type</th>
<th>Constraint Type</th>
<th>BLAZE</th>
<th>FERGIE/ISO-81</th>
<th>KURO</th>
<th>NIRvana LN</th>
<th>PI-MAX 3/4</th>
<th>PI-MTE</th>
<th>PI-MTE3</th>
<th>PIoNIR/NIRvana/ST</th>
<th>PI-PI-</th>
<th>ProEM/ + /-HS</th>
<th>PyLoN</th>
<th>PyLoN-IR</th>
<th>SOPHIA</th>
<th>Quad-RO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Readout Control</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accumulations</td>
<td>L R</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableNondestructiveReadout</td>
<td>B C</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KineticsWindowHeight</td>
<td>I R</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NondestructiveReadoutPeriod</td>
<td>F R</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReadoutControlMode</td>
<td>E C</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReadoutOrientation</td>
<td>✓ E</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReadoutPortCount</td>
<td>I C</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReadoutTimeCalculation</td>
<td>✓ F</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SeNsRWindowHeight</td>
<td>I R</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VerticalShiftRate</td>
<td>F C</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data Acquisition</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DisableDataFormatting</td>
<td>B C</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ExactReadoutCountMaximum</td>
<td>✓ L</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FrameRateCalculation</td>
<td>✓ F</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FrameSize</td>
<td>✓ I</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FramesPerReadout</td>
<td>✓ I</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FrameStride</td>
<td>✓ I</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FrameTrackingBitDepth</td>
<td>I C</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GateTracking</td>
<td>E C</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GateTrackingBitDepth</td>
<td>I C</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ModulationTracking</td>
<td>E C</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ModulationTrackingBitDepth</td>
<td>I C</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NormalizeOrientation</td>
<td>B C</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Read Only</td>
<td>Value Type</td>
<td>Constraint Type</td>
<td>BLAZE</td>
<td>FERGIE/ISO-81</td>
<td>KURO</td>
<td>NIRvana LN</td>
<td>PI-MAX 3/4</td>
<td>PI-MTE</td>
<td>PI-MTE3</td>
<td>PloNIR/NIRvana/ST</td>
<td>PI-050</td>
<td>ProEM</td>
<td>Pylon</td>
<td>Pylon-IR</td>
<td>SOPHIA</td>
<td>Quad-RO</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------</td>
<td>------------</td>
<td>-----------------</td>
<td>-------</td>
<td>---------------</td>
<td>------</td>
<td>------------</td>
<td>------------</td>
<td>--------</td>
<td>---------</td>
<td>-----------------</td>
<td>--------</td>
<td>-------</td>
<td>-------</td>
<td>----------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>OnlineReadoutRate Calculation</td>
<td>✓</td>
<td>F</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Orientation</td>
<td>✓</td>
<td>E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PhotonDetectionMode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PhotonDetection Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PhotonDetection Threshold</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PixelBitDepth</td>
<td>✓</td>
<td>I</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PixelFormat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReadoutCount</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReadoutRate Calculation</td>
<td>✓</td>
<td>F</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ReadoutStride</td>
<td>✓</td>
<td>I</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Rois</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TimeStampBitDepth</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TimeStampResolution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TimeStamps</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TrackFrames</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### Sensor Information

| CcdCharacteristics             | ✓         | E          | ✓               | ✓     | ✓             | ✓    | ✓          | ✓          | ✓      | ✓       | ✓               | ✓      | ✓     | ✓     | ✓        | ✓      | ✓       |
| PixelGapHeight                 | ✓         | F          | ✓               | ✓     | ✓             | ✓    | ✓          | ✓          | ✓      | ✓       | ✓               | ✓      | ✓     | ✓     | ✓        | ✓      | ✓       |
| PixelGapWidth                  | ✓         | F          | ✓               | ✓     | ✓             | ✓    | ✓          | ✓          | ✓      | ✓       | ✓               | ✓      | ✓     | ✓     | ✓        | ✓      | ✓       |
| PixelHeight                    | ✓         | F          | ✓               | ✓     | ✓             | ✓    | ✓          | ✓          | ✓      | ✓       | ✓               | ✓      | ✓     | ✓     | ✓        | ✓      | ✓       |
| PixelWidth                     | ✓         | F          | ✓               | ✓     | ✓             | ✓    | ✓          | ✓          | ✓      | ✓       | ✓               | ✓      | ✓     | ✓     | ✓        | ✓      | ✓       |
| SensorActiveBottom Margin      | ✓         | I          | ✓               | ✓     | ✓             | ✓    | ✓          | ✓          | ✓      | ✓       | ✓               | ✓      | ✓     | ✓     | ✓        | ✓      | ✓       |
| SensorActiveExtended Height    | ✓         | I          | ✓               | ✓     | ✓             | ✓    | ✓          | ✓          | ✓      | ✓       | ✓               | ✓      | ✓     | ✓     | ✓        | ✓      | ✓       |
| SensorActiveHeight             | ✓         | I          | ✓               | ✓     | ✓             | ✓    | ✓          | ✓          | ✓      | ✓       | ✓               | ✓      | ✓     | ✓     | ✓        | ✓      | ✓       |
| SensorActiveLeft Margin        | ✓         | I          | ✓               | ✓     | ✓             | ✓    | ✓          | ✓          | ✓      | ✓       | ✓               | ✓      | ✓     | ✓     | ✓        | ✓      | ✓       |
### Table A-2: Parameter Information and Camera Support (Sheet 7 of 9)

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Read Only</th>
<th>Value Type</th>
<th>Constraint Type</th>
<th>BLAZE</th>
<th>FERGIE/ISO-81</th>
<th>KURO</th>
<th>NIRvana LN</th>
<th>PI-MAX 3/4</th>
<th>PI-MTE</th>
<th>PI-MTE3</th>
<th>PI-MIR/NIRvana/ST</th>
<th>PIoNIR/NIRvana/ST</th>
<th>PI-MITY</th>
<th>ProEM/ +/ -HS</th>
<th>PyLoN</th>
<th>PyLoN-IR</th>
<th>SOPHIA</th>
<th>Quad-RO</th>
</tr>
</thead>
<tbody>
<tr>
<td>SensorActiveRight Margin</td>
<td>✓</td>
<td>I</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SensorActiveTop Margin</td>
<td>✓</td>
<td>I</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SensorActiveWidth</td>
<td>✓</td>
<td>I</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SensorMaskedBottom Margin</td>
<td>✓</td>
<td>I</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SensorMaskedHeight</td>
<td>✓</td>
<td>I</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SensorMaskedTop Margin</td>
<td>✓</td>
<td>I</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SensorSecondaryActiveHeight</td>
<td>✓</td>
<td>I</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SensorSecondaryMaskedHeight</td>
<td>✓</td>
<td>I</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SensorType</td>
<td>✓</td>
<td>E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### Sensor Layout

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value Type</th>
<th>Constraint Type</th>
<th>BLAZE</th>
<th>FERGIE/ISO-81</th>
<th>KURO</th>
<th>NIRvana LN</th>
<th>PI-MAX 3/4</th>
<th>PI-MTE</th>
<th>PI-MTE3</th>
<th>PI-MIR/NIRvana/ST</th>
<th>PIoNIR/NIRvana/ST</th>
<th>PI-MITY</th>
<th>ProEM/ +/ -HS</th>
<th>PyLoN</th>
<th>PyLoN-IR</th>
<th>SOPHIA</th>
<th>Quad-RO</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveBottomMargin</td>
<td>I</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ActiveExtendedHeight</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ActiveHeight</td>
<td>I</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ActiveLeftMargin</td>
<td>I</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ActiveRightMargin</td>
<td>I</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ActiveTopMargin</td>
<td>I</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ActiveWidth</td>
<td>I</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MaskedBottomMargin</td>
<td>I</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MaskedHeight</td>
<td>I</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MaskedTopMargin</td>
<td>I</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SecondaryActiveHeight</td>
<td>I</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SecondaryMaskedHeight</td>
<td>I</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
## Table A-2: Parameter Information and Camera Support (Sheet 8 of 9)

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Read Only</th>
<th>Value Type</th>
<th>Constraint Type</th>
<th>BLAZE</th>
<th>FERGIE/ISO-81</th>
<th>KURO</th>
<th>NIRvana LN</th>
<th>PI-MAX 3/4</th>
<th>PI-MTE</th>
<th>PI-MTE3</th>
<th>PioNIR/Nirvana/ST</th>
<th>PIXIS</th>
<th>ProEM/+/-HS</th>
<th>PyLoN</th>
<th>PyLoN-IR</th>
<th>SOPHIA</th>
<th>Quad-RO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sensor Cleaning</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CleanBeforeExposure</td>
<td>B</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CleanCycleCount</td>
<td>I</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CleanCycleHeight</td>
<td>I</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CleanSectionFinal Height</td>
<td>I</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CleanSectionFinal HeightCount</td>
<td>I</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CleanSerialRegister</td>
<td>B</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CleanUntilTrigger</td>
<td>B</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>StopCleaningOnPreTrigger</td>
<td>B</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sensor Temperature</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoolingFanStatus</td>
<td>✓</td>
<td>E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DisableCoolingFan</td>
<td>B</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableSensorWindow Heater</td>
<td>B</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SensorTemperatureReading</td>
<td>✓</td>
<td>F</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SensorTemperatureSetPoint</td>
<td>F</td>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SensorTemperatureStatus</td>
<td>✓</td>
<td>E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VacuumStatus</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table A-2: Parameter Information and Camera Support (Sheet 9 of 9)

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Read Only</th>
<th>Value Type</th>
<th>Constraint Type</th>
<th>BLAZE</th>
<th>FERGIE/ISO-81</th>
<th>KURO</th>
<th>NIRvana LN</th>
<th>PI-MAX 3/4</th>
<th>PI-MTE</th>
<th>PI-MTE3</th>
<th>PioNIR/NIRvana/ST</th>
<th>PI-EMS</th>
<th>ProEM/ + / -HS</th>
<th>PyLoN</th>
<th>PyLoN-IR</th>
<th>SOPHIA</th>
<th>Quad-RO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spectrograph</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CenterWavelength Reading</td>
<td>✓</td>
<td>F</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CenterWavelength Set Point</td>
<td></td>
<td>F</td>
<td>R</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CenterWavelength Status</td>
<td>✓</td>
<td>E</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FocalLength</td>
<td>✓</td>
<td>F</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GratingBlazing Wavelength</td>
<td>✓</td>
<td>F</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GratingCoating</td>
<td>✓</td>
<td>E</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GratingGrooveDensity</td>
<td>✓</td>
<td>F</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GratingType</td>
<td>✓</td>
<td>E</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InclusionAngle</td>
<td>✓</td>
<td>F</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SensorAngle</td>
<td>✓</td>
<td>F</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### A.2 Accessory Parameter Information

Refer to Table A-3 for the list of available Accessory parameters.

**Table A-3: Parameter Information and Accessory Support**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Read Only</th>
<th>Value Type</th>
<th>Constraint Type</th>
<th>FERGIE AEL</th>
<th>FERGIE QTH</th>
<th>FERGIE Laser 532/785</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accessory – Lamp</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Age</td>
<td>✓</td>
<td>F</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LifeExpectancy</td>
<td>✓</td>
<td>F</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>LightSource</td>
<td></td>
<td>E</td>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>LightSourceStatus</td>
<td>✓</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Accessory – Laser</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InputTriggerStatus</td>
<td>✓</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td>✓*</td>
</tr>
<tr>
<td>LaserOutputMode</td>
<td></td>
<td>E</td>
<td>C</td>
<td>✓</td>
<td>✓*</td>
<td></td>
</tr>
<tr>
<td>LaserPower</td>
<td></td>
<td>F</td>
<td>R</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LaserStatus</td>
<td>✓</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix B: EM Gain Calibration Code
Sample

⚠️ CAUTION! ⚠️
The information provided within this appendix is NOT applicable to emICCD cameras.

The EMGainCalibration.exe file is sample code included with PICam which, when built, allows PICam users to perform an EM Gain Calibration that may occasionally be required by ProEM systems.

**NOTE:**
Users with access to LightField do not need to build the sample code in order to perform an EM Gain Calibration. LightField includes EMGainCalibration.exe as part of its normal installation. The fully-functional executable file is located within the standard LightField installation directory (i.e., where PrincetonInstruments.LightField.exe is stored.

EMGainCalibration.exe is an excellent alternative for developers who do not need to create a custom EM gain calibration application. Once the sample code has been built, it can be included as part of the standard customer installation process and allows users to perform EM calibration on an as needed basis.

⚠️ CAUTION! ⚠️
When calibrating a ProEM camera with a manual shutter (e.g., ProEM:1600,) the shutter MUST be closed manually before launching the calibration program.
This is not necessary for a camera with an internal electro-mechanical shutter because the program will automatically close the shutter before beginning the calibration.

### B.1 EM Gain Calibration Procedure

Perform the following procedure to perform an EM Gain calibration:

1. If a data acquisition program is running (e.g., custom PICam application, LightField, etc.) close it.
2. Verify that the ProEM camera that is to be calibrated is the only ProEM camera connected to the host computer and that it is turned on.
3. If the camera has a manual shutter, verify that it is closed. If necessary, close it.
4. Launch EMGainCalibration.exe.

⚠️ **CAUTION!**

Do not operate the camera while EM gain calibration is in process.

5. When the **EM Gain Calibration** dialog is displayed, the default temperature for the camera is shown in the **Target** field. See Figure B-1.

**Figure B-1: Typical EM Gain Calibration Dialog**

If the camera typically operates at a different temperature, manually adjust it as necessary.

6. Once the **Current** temperature reaches the **Target** temperature specified:
   - The internal shutter closes;

   **NOTE:**
   When using a manual shutter, it must be closed **prior** to initiating the calibration procedure.

   - The internal light illuminates the sensor;
   - A series of data frames is acquired;
   - The calibration map is then calculated.

   **NOTE:**
   Wait until the calibration has completed before launching the data acquisition program. It may take up to 10 minutes for the calibration to be completed.
Appendix C: Firmware Upgrade/Restore

This appendix provides the procedures to upgrade and restore a GigE camera’s firmware.

NOTE: It is strongly recommended that cameras be upgraded one at a time to avoid confusion.

C.1 Firmware Upgrade Procedure

Perform the following procedure to upgrade a GigE camera’s firmware to be compatible with PICam 5.x:

1. On the Host Computer, navigate to the following directory:
   `c:\program files\princeton instruments\picam\firmware`

2. Double-click on the `Firmware_Upgrade.exe` file to launch the upgrade tool. The IP Engine Selection dialog is displayed. See Figure C-1.

Figure C-1: Firmware Upgrade: Typical IP Engine Selection Dialog

The IP Engine Selection dialog is displayed. See Figure C-1.
3. Within the **Available IP Engines** field, select the desired IP Engine from the list of available devices.

**NOTE:**

Each IP Engine listed represents one camera.

See Figure C-2.

Figure C-2: Firmware Upgrade: Selecting Device to be Upgraded

4. Once selected, click **OK** to begin the automated firmware upgrade process. The **Updating** dialog is displayed, similar to that shown in Figure C-3.

Figure C-3: Firmware Upgrade: Typical Updating Dialog

As the upgrade proceeds, the dialog displays appropriate messages, and the progress bar provides a visual indication.
5. Once the upgrade is complete, click **OK** on the **Updating** dialog. See **Figure C-4**.

![Figure C-4: Firmware Upgrade: Upgrade Complete](image)

6. Finally, cycle power to the camera to complete the Firmware Upgrade.

## C.2 Restore Firmware

In the unlikely event that PICam 3.x firmware must be restored onto a camera, this section provides detailed information about using the Princeton Instruments provided Firmware Restore tool.

### C.2.1 Precautions

Unlike the firmware upgrade procedure that requires no special preparation or precautions, restoring PICam 3.x firmware requires some planning to avoid unnecessary complications.

It is strongly recommended that PICam 3.x firmware be restored on all affected GigE cameras before uninstalling PICam 5.x from the host computer. The Firmware Restore Tool is not included with PICam 3.x installations. Uninstalling PICam 5.x from the host computer will completely remove it making it unavailable for use afterward.

**NOTE:**

If it is anticipated that additional GigE cameras will require a firmware restore after PICam 5.x has been uninstalled, move the Firmware Restore Tool into a non-PICam directory on the host computer prior to uninstalling PICam 5.x. Additionally, Pleora EBUS Support is required in order to run the Firmware Restore Tool. Do not uninstall Pleora EBUS Support.
C.2.2 Procedure

Perform the following procedure:

1. On the Host Computer, navigate to the following directory:
   
   c:\program files\princeton instruments\picam\firmware

2. Double-click on the Firmware_Restore.exe file to launch the firmware restore tool.
   The IP Engine Selection dialog is displayed. See Figure C-5.

Figure C-5: Firmware Restore: Typical IP Engine Selection Dialog

3. Within the Available IP Engines field, select the desired IP Engine from the list of available devices. See Figure C-6.

   NOTE: Each IP Engine listed represents one camera.

Figure C-6: Firmware Restore: Selecting Device to be Restored
4. Once selected, click **OK** to begin restoring the firmware. The **Updating** dialog is displayed, similar to that shown in **Figure C-7**.

**Figure C-7: Firmware Restore: Typical Updating Dialog**

![Updating Dialog](image)

As the firmware restoration proceeds, the dialog displays appropriate messages, and the progress bar provides a visual indication.

5. Once the firmware has been restored, click **OK** on the **Updating** dialog. See **Figure C-8**.

**Figure C-8: Firmware Restore: Complete**

![Updating Dialog](image)

6. Finally, cycle power to the camera to complete the Firmware Restore.
This page is intentionally blank.
Appendix D: Debugging GigE Cameras

Beginning with PICam 5.x, Princeton Instruments’ GigE cameras incorporate a Camera Heartbeat that enables the camera and PICam to coordinate communication with one another. As long as a Heartbeat signal has been received by the camera before the specified Heartbeat Timeout has expired, the Camera will continue to be controlled exclusively by PICam.

Under normal operation, the implementation of a Camera Heartbeat is completely transparent to end-users of GigE cameras. However, developers must be keenly aware of how the Heartbeat Timeout impacts camera availability during successive debugging sessions when a PICam process either crashes or is intentionally killed.

Once PICam tasks have been completed, an orderly cessation of the Camera Heartbeat is initiated, and the communication channel between PICam and the camera is closed. At this point, if desired, the camera can again be controlled by PICam or by another program/device.

If, however, PICam halts unexpectedly (e.g., it crashes, the process is killed,) the camera will continue to wait for the next incoming Heartbeat signal until such time as the Heartbeat Timeout has expired. While waiting, the camera remains unavailable to other processes, devices, and programs.

**NOTE:**

The primary symptom of an expired Heartbeat is a continuous string of unexpected errors being received.

Only after the Heartbeat Timeout has expired without an incoming Heartbeat signal having been detected will the camera close its communication channel with PICam and become available to other processes or devices. At this point, PICam will need to be reinitialized/restarted.

D.1 Debugging

The introduction of the Camera Heartbeat presents additional challenges to developers during the Debugging stage of software development. When a breakpoint is reached, PICam execution halts, and no additional Camera Heartbeats are sent to the camera. If the configured Heartbeat Timeout is too short, it will expire, the camera will close its communication channel with PICam requiring it to be reinitialized, and thus preventing subsequent sections of code from easily being executed, examined, and debugged.

The solution to this dilemma is to extend the timeout period sufficiently by configuring the Heartbeat Timeout for an appropriately large value (e.g., 5 minutes.) Increasing the timeout permits the executed code to be examined/debugged while the camera waits patiently for the next incoming Heartbeat signal from PICam. As long as execution of the next chunk of code has been manually initiated before the Heartbeat Timeout expires, PICam will send another Heartbeat signal to the camera (which, in turn, resets the Heartbeat Timeout timer,) and the next chunk of code executes until it reaches the next breakpoint.
D.1.1 Timeout Period Considerations

When deciding on an appropriate timeout period, achieving a balance between having adequate time to review/debug each section of code while not consistently timing out can be tricky.

If too long of a timeout period has been selected and the PICam process crashes or is subsequently killed (a typical action following any debugging session,) the GigE camera will remain unavailable to a future debugging session until the Heartbeat Timeout has expired.

In order to immediately release the camera following a crashed/killed process, cycling its power will clean up any processes that have been abnormally terminated. However, unless the Heartbeat Timeout is programmed for a shorter time period, if the program experiences a subsequent abnormal termination, the camera will again remain unavailable to future debug sessions, and power will need to be cycled again.

D.1.2 Following Debugging

Once debugging has concluded, be sure to reset the value of the `PICAM_GIGE_TIMEOUT` environment variable to a more appropriate timeout.

D.2 Timeout Configuration

The Heartbeat Timeout, in milliseconds, is defined by the environment variable `PICAM_GIGE_TIMEOUT`.

Valid values are:

- **Minimum**: 500 ms;
- **Default**: 2000 ms;
- **Maximum**: 4,294,967,295 ms (approximately 49.7 days).
Appendix E: PICam 5.0 Compatibility Issues

Beginning with PICam 5.0, usage of the suite of Left/Right Margin Parameters has been modified for scenarios where Readout Orientation is not Normal. Specifically, the parameter PicamParameter_ReadoutOrientation does not have the value PicamOrientationMask_Normal. Originally, the Left/Right Margin parameters would swap positions in this scenario. However, this is incorrect and has been fixed to their proper locations.

The specific set of parameters impacted by this change varies by camera/system, and the default value for each impacted parameter has been properly updated. Although it is extremely rare to either change these parameters or make coding decisions based on their values, if either of these have been implemented, information is provided in the following sections with suggestions about required code updates when upgrading to PICam 5.0.

E.1 FERGIE: 256F/FT, FERGIE: 256B/FT, FERGIE: 256BR/FT, and eXcelon Variant Cameras

This section provides information necessary to update customer-modified code (PICam 4.x and older) for the following cameras in order for it to work as designed with PICam 5.0:

- FERGIE: 256F/FT;
- FERGIE: 256B/FT;
- FERGIE: 256BR/FT;
- eXcelon Variants.

E.1.1 What Changed with PICam 5.0?

In PICam 5.0, the following parameters have been swapped:

- PicamParameter_ActiveLeftMargin and PicamParameter_ActiveRightMargin

By default, these two parameters remain symmetric, so programs will break only when one, or both, of these parameters have been set asymmetrically.
### E.1.2 Code Updates to Retain Existing Behavior

Table E-1 lists code changes required to PICam 4.x (and older) code in order to continue working properly with PICam 5.0.

#### Table E-1: PICam 4.x (and Older) Code Changes: FERGIE: 256F/FT, 256B/FT, 256BR/FT, and eXcelon Variants

<table>
<thead>
<tr>
<th>In PICam 4.x and Older Code, if you:</th>
<th>... in PICam 5.0, you need to:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Action</strong></td>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Set</td>
<td>PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Set</td>
<td>PicamParameter_ActiveRightMargin</td>
</tr>
</tbody>
</table>
E.2 PI-MAX4: 2048B, PI-MAX4: 2048B-RF Cameras

This section provides information necessary to update customer-modified code (PICam 4.x and older) for the following cameras in order for it to work as designed with PICam 5.0:

- PI-MAX4:2048B;
- PI-MAX4:2048B-RF.

E.2.1 What Changed with PICam 5.0?

In PICam 5.0, the following parameters have been swapped:

- PicamParameter_ActiveLeftMargin and PicamParameter_ActiveRightMargin

By default, these two parameters remain symmetric, so programs will break only when one, or both, of these parameters have been set asymmetrically.

E.2.2 Code Updates to Retain Existing Behavior

Table E-2 lists code changes required to PICam 4.x (and older) code in order to continue working properly with PICam 5.0.

Table E-2: PICam 4.x (and Older) Code Changes: PI-MAX4: 2048B/2048B-RF

<table>
<thead>
<tr>
<th>In PICam 4.x and Older Code, if you:</th>
<th>... in PICam 5.0, you need to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Parameter</td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Set</td>
<td>PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Set</td>
<td>PicamParameter_ActiveRightMargin</td>
</tr>
</tbody>
</table>
E.3 PI-MAX4: 512B/EM, PI-MAX4: 1024B/EM

This section provides information necessary to update customer-modified code (PICam 4.x and older) for the following cameras in order for it to work as designed with PICam 5.0:
- PI-MAX4: 512B/EM;
- PI-MAX4: 1024B/EM.

E.3.1 What Changed with PICam 5.0?

In PICam 5.0, the following parameters have been made symmetrical in general, and have been swapped in Electron Multiplied (PicamAdcQuality_ElectronMultiplied):
- PicamParameter_ActiveLeftMargin and PicamParameter_ActiveRightMargin
- PicamParameter_SensorActiveLeftMargin and PicamParameter_SensorActiveRightMargin

E.3.2 Code Updates to Retain Existing Behavior

Table E-3 lists code changes required to PICam 4.x (and older) code in order to continue working properly with PICam 5.0.

Table E-3: PICam 4.x (and Older) Code Changes: PI-MAX4: 512B/EM, 1024B/EM (Sheet 1 of 2)

<table>
<thead>
<tr>
<th>In PICam 4.x and Older Code, if you:</th>
<th>... in PICam 5.0, you need to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Parameter</td>
</tr>
<tr>
<td>Get PicamParameter_ActiveLeftMargin</td>
<td>Get PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Set PicamParameter_ActiveLeftMargin</td>
<td>Set PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Get PicamParameter_ActiveRightMargin</td>
<td>Get Subtract PicamParameter_ActiveLeftMargin 16 from the value</td>
</tr>
<tr>
<td>Set PicamParameter_ActiveRightMargin</td>
<td>Add Set 16 to the value PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Get PicamParameter_SensorActiveLeftMargin</td>
<td>Get PicamParameter_SensorActiveRightMargin</td>
</tr>
<tr>
<td>Get PicamParameter_SensorActiveRightMargin</td>
<td>Get Subtract PicamParameter_SensorActiveLeftMargin 16 from the value</td>
</tr>
</tbody>
</table>
## Appendix E PICam 5.0 Compatibility Issues

Table E-3: PICam 4.x (and Older) Code Changes: PI-MAX4: 512B/EM, 1024B/EM (Sheet 2 of 2)

<table>
<thead>
<tr>
<th>Action</th>
<th>Parameter</th>
<th>Action</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ADC Quality: Low Noise</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get</td>
<td><code>PicamParameter_ActiveRightMargin</code></td>
<td>Get</td>
<td><code>PicamParameter_ActiveRightMargin</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Subtract</td>
<td>16 from the value</td>
</tr>
<tr>
<td>Set</td>
<td><code>PicamParameter_ActiveRightMargin</code></td>
<td>Add</td>
<td>16 to the value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set</td>
<td><code>PicamParameter_ActiveRightMargin</code></td>
</tr>
<tr>
<td>Get</td>
<td><code>PicamParameter_SensorActiveRightMargin</code></td>
<td>Get</td>
<td><code>PicamParameter_SensorActiveRightMargin</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Subtract</td>
<td>16 from the value</td>
</tr>
</tbody>
</table>
E.4 PI-MAX4: 512EM/1024EM Cameras

This section provides information necessary to update customer-modified code (PICam 4.x and older) for the following cameras in order for it to work as designed with PICam 5.0:

- PI-MAX4: 512EM;
- PI-MAX4: 1024EM.

E.4.1 What Changed with PICam 5.0?

In PICam 5.0, the following parameters have been made symmetrical in general, and have been swapped in Low Noise (PicamAdcQuality_LowNoise):

- PicamParameter_ActiveLeftMargin and PicamParameter_ActiveRightMargin
- PicamParameter_SensorActiveLeftMargin and PicamParameter_SensorActiveRightMargin

E.4.2 Code Updates to Retain Existing Behavior

Table E-4 lists code changes required to PICam 4.x (and older) code in order to continue working properly with PICam 5.0.

Table E-4: PICam 4.x (and Older) Code Changes: PI-MAX4: 512EM/1024EM

<table>
<thead>
<tr>
<th>In PICam 4.x and Older Code, if you:</th>
<th>... in PICam 5.0, you need to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Parameter</td>
</tr>
<tr>
<td>ADC Quality: Electron Multiplied</td>
<td></td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Set</td>
<td>PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_SensorActiveRightMargin</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC Quality: Low Noise</td>
<td></td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Set</td>
<td>PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Set</td>
<td>PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_SensorActiveLeftMargin</td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_SensorActiveRightMargin</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
E.5 PI-MTE: 1300B/1300BR Cameras

This section provides information necessary to update customer-modified code (PICam 4.x and older) for the following cameras in order for it to work as designed with PICam 5.0:

- PI-MTE: 1300B;
- PI-MTE: 1300BR.

E.5.1 What Changed with PICam 5.0?

In PICam 5.0, the following High Capacity (PicamAdcQuality_HighCapacity) parameters have been swapped:

- PicamParameter_ActiveLeftMargin and PicamParameter_ActiveRightMargin

By default, these two parameters remain symmetric, so programs will break only when one, or both, of these parameters have been set asymmetrically in High Capacity.

E.5.2 Code Updates to Retain Existing Behavior

Table E-5 lists code changes required to PICam 4.x (and older) code in order to continue working properly with PICam 5.0.

Table E-5: PICam 4.x (and Older) Code Changes: PI-MTE: 1300B/1300BR

<table>
<thead>
<tr>
<th>In PICam 4.x and Older Code, if you:</th>
<th>... in PICam 5.0, you need to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Parameter</td>
</tr>
<tr>
<td>Get PicamParameter_ActiveLeftMargin</td>
<td>Get PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Set PicamParameter_ActiveLeftMargin</td>
<td>Set PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Get PicamParameter_ActiveRightMargin</td>
<td>Get PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Set PicamParameter_ActiveRightMargin</td>
<td>Set PicamParameter_ActiveLeftMargin</td>
</tr>
</tbody>
</table>
E.6  PI-MTE: 1300R Cameras

This section provides information necessary to update customer-modified code (PICam 4.x and older) for the following cameras in order for it to work as designed with PICam 5.0:

- PI-MTE: 1300R.

E.6.1  What Changed with PICam 5.0?

In PICam 5.0, the following Low Noise (PicamAdcQuality_LowNoise) parameters have been swapped:

- PicamParameter_ActiveLeftMargin and PicamParameter_ActiveRightMargin

By default, these two parameters remain symmetric, so programs will break only when one, or both, of these parameters have been set asymmetrically in Low Noise.

E.6.2  Code Updates to Retain Existing Behavior

Table E-6 lists code changes required to PICam 4.x (and older) code in order to continue working properly with PICam 5.0.

Table E-6: PICam 4.x (and Older) Code Changes: PI-MTE: 1300R

<table>
<thead>
<tr>
<th>In PICam 4.x and Older Code, if you:</th>
<th>... in PICam 5.0, you need to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Parameter</td>
</tr>
<tr>
<td>Get PicamParameter_ActiveLeftMargin</td>
<td>Get PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Set PicamParameter_ActiveLeftMargin</td>
<td>Set PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Get PicamParameter_ActiveRightMargin</td>
<td>Get PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Set PicamParameter_ActiveRightMargin</td>
<td>Set PicamParameter_ActiveLeftMargin</td>
</tr>
</tbody>
</table>
E.7 PIXIS: 100B/100BR/400B/400BR/1300B/1300BR, and XO/XF/XB/eXcelon Variant Cameras

This section provides information necessary to update customer-modified code (PICam 4.x and older) for the following cameras in order for it to work as designed with PICam 5.0:

- PIXIS: 100B;
- PIXIS: 100BR;
- PIXIS: 400B;
- PIXIS: 400BR;
- PIXIS: 1300B;
- PIXIS: 1300BR;
- XO/XF/XF eXcelon Variants.

E.7.1 What Changed with PICam 5.0?

In PICam 5.0, the following High Capacity (PicamAdcQuality_HighCapacity) parameters have been swapped:

- PicamParameter_ActiveLeftMargin and PicamParameter_ActiveRightMargin

By default, these two parameters remain symmetric, so programs will break only when one, or both, of these parameters have been set asymmetrically in High Capacity.

E.7.2 Code Updates to Retain Existing Behavior

Table E-7 lists code changes required to PICam 4.x (and older) code in order to continue working properly with PICam 5.0.

<table>
<thead>
<tr>
<th>In PICam 4.x and Older Code, if you:</th>
<th>... in PICam 5.0, you need to:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Action</strong></td>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>ADC Quality: High Capacity</td>
<td></td>
</tr>
<tr>
<td>Get PicamParameter_ActiveLeftMargin</td>
<td>Get PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Set PicamParameter_ActiveLeftMargin</td>
<td>Set PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Get PicamParameter_ActiveRightMargin</td>
<td>Get PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Set PicamParameter_ActiveRightMargin</td>
<td>Set PicamParameter_ActiveLeftMargin</td>
</tr>
</tbody>
</table>
E.8 PIXIS: 100F/100R/100C/400F/400R/1300F/1300F-2, and XB Variant Cameras

This section provides information necessary to update customer-modified code (PICam 4.x and older) for the following cameras in order for it to work as designed with PICam 5.0:

- PIXIS: 100F;
- PIXIS: 100R;
- PIXIS: 100C;
- PIXIS: 400F;
- PIXIS: 400R;
- PIXIS: 1300F;
- PIXIS: 1300F-2;
- XB Variants.

E.8.1 What Changed with PICam 5.0?

In PICam 5.0, the following Low Noise (PicamAdcQuality_LowNoise) parameters have been swapped:

- PicamParameter_ActiveLeftMargin and PicamParameter_ActiveRightMargin

By default, these two parameters remain symmetric, so programs will break only when one, or both, of these parameters have been set asymmetrically in Low Noise.

E.8.2 Code Updates to Retain Existing Behavior

Table E-8 lists code changes required to PICam 4.x (and older) code in order to continue working properly with PICam 5.0.

Table E-8: PICam 4.x (and Older) Code Changes: PIXIS: 100F/100R/100C/400F/400R/1300F/1300F-2, and XB Variants

<table>
<thead>
<tr>
<th>In PICam 4.x and Older Code, if you:</th>
<th>... in PICam 5.0, you need to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Parameter</td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Set</td>
<td>PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Set</td>
<td>PicamParameter_ActiveRightMargin</td>
</tr>
</tbody>
</table>
Appendix E

PICam 5.0 Compatibility Issues

E.9 PIXIS: 512F, PIXIS-XO: 512F, PIXIS-XF: 512F Cameras

This section provides information necessary to update customer-modified code (PICam 4.x and older) for the following cameras in order for it to work as designed with PICam 5.0:

- PIXIS: 512F;
- PIXIS-XO: 512F;
- PIXIS-XF: 512F.

E.9.1 What Changed with PICam 5.0?

In PICam 5.0, the following parameters have been swapped:

- `PicamParameter_ActiveLeftMargin` and `PicamParameter_ActiveRightMargin`
- `PicamParameter_SensorActiveLeftMargin` and `PicamParameter_SensorActiveRightMargin`

E.9.2 Code Updates to Retain Existing Behavior

Table E-9 lists code changes required to PICam 4.x (and older) code in order to continue working properly with PICam 5.0.

Table E-9: PICam 4.x (and Older) Code Changes: PIXIS: 512F, PIXIS-XO: 512F, and PIXIS-XF: 512F

<table>
<thead>
<tr>
<th>In PICam 4.x and Older Code, if you:</th>
<th>... in PICam 5.0, you need to:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Action</strong></td>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>Get</td>
<td><code>PicamParameter_ActiveLeftMargin</code></td>
</tr>
<tr>
<td>Set</td>
<td><code>PicamParameter_ActiveLeftMargin</code></td>
</tr>
<tr>
<td>Get</td>
<td><code>PicamParameter_ActiveRightMargin</code></td>
</tr>
<tr>
<td>Set</td>
<td><code>PicamParameter_ActiveRightMargin</code></td>
</tr>
<tr>
<td>Get</td>
<td><code>PicamParameter_SensorActiveLeftMargin</code></td>
</tr>
<tr>
<td>Get</td>
<td><code>PicamParameter_SensorActiveRightMargin</code></td>
</tr>
</tbody>
</table>
E.10 ProEM Cameras (All Models)

This section provides information necessary to update customer-modified code (PICam 4.x and older) for the following cameras in order for it to work as designed with PICam 5.0:

- ProEM (All Models)

E.10.1 What Changed with PICam 5.0?

In PICam 5.0, the following parameters have been made symmetrical in general, and have been swapped in Electron Multiplied (PicamAdcQuality_ElectronMultiplied):

- PicamParameter_ActiveLeftMargin and PicamParameter_ActiveRightMargin
- PicamParameter_SensorActiveLeftMargin and PicamParameter_SensorActiveRightMargin

E.10.2 Code Updates to Retain Existing Behavior

Table E-10 lists code changes required to PICam 4.x (and older) code in order to continue working properly with PICam 5.0.

Table E-10: PICam 4.x (and Older) Code Changes: ProEM (All Models)

<table>
<thead>
<tr>
<th>In PICam 4.x and Older Code, if you:</th>
<th>... in PICam 5.0, you need to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Parameter</td>
</tr>
<tr>
<td>ADC Quality: Electron Multiplied</td>
<td></td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Set</td>
<td>PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Set</td>
<td>PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_SensorActiveLeftMargin</td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_SensorActiveRightMargin</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ADC Quality: Low Noise

<table>
<thead>
<tr>
<th>In PICam 4.x and Older Code, if you:</th>
<th>... in PICam 5.0, you need to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Parameter</td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Set</td>
<td>PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_SensorActiveRightMargin</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
E.11 ProEM-HS: 1KB-10 and eXcelon Variant Cameras

This section provides information necessary to update customer-modified code (PICam 4.x and older) for the following cameras in order for it to work as designed with PICam 5.0:

- ProEM-HS: 1KB-10;
- eXcelon Variants.

E.11.1 What Changed with PICam 5.0?

In PICam 5.0, the following parameters have been swapped:

- `PicamParameter_ActiveLeftMargin` and `PicamParameter_ActiveRightMargin`
- `PicamParameter_SensorActiveLeftMargin` and `PicamParameter_SensorActiveRightMargin`

E.11.2 Code Updates to Retain Existing Behavior

Table E-11 lists code changes required to PICam 4.x (and older) code in order to continue working properly with PICam 5.0.

Table E-11: PICam 4.x (and Older) Code Changes: ProEM-HS: 1KB-10 and eXcelon Variants

<table>
<thead>
<tr>
<th>In PICam 4.x and Older Code, if you:</th>
<th>... in PICam 5.0, you need to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Parameter</td>
</tr>
<tr>
<td>Get</td>
<td><code>PicamParameter_ActiveLeftMargin</code></td>
</tr>
<tr>
<td>Set</td>
<td><code>PicamParameter_ActiveLeftMargin</code></td>
</tr>
<tr>
<td>Get</td>
<td><code>PicamParameter_ActiveRightMargin</code></td>
</tr>
<tr>
<td>Set</td>
<td><code>PicamParameter_ActiveRightMargin</code></td>
</tr>
<tr>
<td>Get</td>
<td><code>PicamParameter_SensorActiveLeftMargin</code></td>
</tr>
<tr>
<td>Get</td>
<td><code>PicamParameter_SensorActiveRightMargin</code></td>
</tr>
</tbody>
</table>
E.12 ProEM-HS: 512B/512BK/1024B and eXcelon Variant Cameras

This section provides information necessary to update customer-modified code (PICam 4.x and older) for the following cameras in order for it to work as designed with PICam 5.0:

- ProEM-HS: 512B;
- ProEM-HS: 512BK;
- ProEM-HS: 1024B;
- eXcelon Variants

E.12.1 What Changed with PICam 5.0?

In PICam 5.0, the following parameters have been made symmetrical in general, and have been swapped in Electron Multiplied (PicamAdcQuality_ElectronMultiplied):

- PicamParameter_ActiveLeftMargin and PicamParameter_ActiveRightMargin
- PicamParameter_SensorActiveLeftMargin and PicamParameter_SensorActiveRightMargin

E.12.2 Code Updates to Retain Existing Behavior

Table E-12 lists code changes required to PICam 4.x (and older) code in order to continue working properly with PICam 5.0.

Table E-12: PICam 4.x (and Older) Code Changes: ProEM-HS: 512B/512BK/1024B and eXcelon Variants (Sheet 1 of 2)

<table>
<thead>
<tr>
<th>In PICam 4.x and Older Code, if you:</th>
<th>... in PICam 5.0, you need to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Parameter</td>
</tr>
<tr>
<td>ADC Quality: Electron Multiplied</td>
<td></td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Set</td>
<td>PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Set</td>
<td>PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_SensorActiveLeftMargin</td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_SensorActiveRightMargin</td>
</tr>
</tbody>
</table>
### Table E-12: PICam 4.x (and Older) Code Changes: ProEM-HS: 512B/512BK/1024B and eXcelon Variants (Sheet 2 of 2)

<table>
<thead>
<tr>
<th>In PICam 4.x and Older Code, if you:</th>
<th>… in PICam 5.0, you need to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Parameter</td>
</tr>
<tr>
<td>Get</td>
<td><strong>PicamParameter_ActiveRightMargin</strong></td>
</tr>
<tr>
<td>Set</td>
<td><strong>PicamParameter_ActiveRightMargin</strong></td>
</tr>
<tr>
<td>Get</td>
<td><strong>PicamParameter_SensorActiveRightMargin</strong></td>
</tr>
<tr>
<td></td>
<td>Subtract <strong>16</strong> from the value <strong>PicamParameter_SensorActiveRightMargin</strong></td>
</tr>
</tbody>
</table>
E.13 ProEM+ (All Models)

This section provides information necessary to update customer-modified code (PICam 4.x and older) for the following cameras in order for it to work as designed with PICam 5.0:
- ProEM+ (All Models)

E.13.1 What Changed with PICam 5.0?

In PICam 5.0, the following parameters have been made symmetrical in general, and have been swapped in Electron Multiplied (PicamAdcQuality_ElectronMultiplied):

- PicamParameter_ActiveLeftMargin and PicamParameter_ActiveRightMargin
- PicamParameter_SensorActiveLeftMargin and PicamParameter_SensorActiveRightMargin

E.13.2 Code Updates to Retain Existing Behavior

Table E-13 lists code changes required to PICam 4.x (and older) code in order to continue working properly with PICam 5.0.

Table E-13: PICam 4.x (and Older) Code Changes: ProEM+ (All Models)

<table>
<thead>
<tr>
<th>In PICam 4.x and Older Code, if you:</th>
<th>... in PICam 5.0, you need to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Parameter</td>
</tr>
<tr>
<td>ADC Quality: Electron Multiplied</td>
<td></td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Set</td>
<td>PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Set</td>
<td>PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_SensorActiveLeftMargin</td>
</tr>
<tr>
<td>Get</td>
<td>PicamParameter_SensorActiveRightMargin</td>
</tr>
</tbody>
</table>

ADC Quality: Low Noise

| Get | PicamParameter_ActiveRightMargin | Get | PicamParameter_ActiveRightMargin |
| Subtract | 16 from the value |
| Set | PicamParameter_ActiveRightMargin | Add | 16 to the value |
| Set | PicamParameter_ActiveRightMargin | Subtract | 16 from the value |
E.14 PyLoN: 100B/100BR/400B/400BR/1300B/1300BR, and eXcelon Variant Cameras

This section provides information necessary to update customer-modified code (PICam 4.x and older) for the following cameras in order for it to work as designed with PICam 5.0:

- PyLoN: 100B;
- PyLoN: 100BR;
- PyLoN: 400B;
- PyLoN: 400BR;
- PyLoN: 1300B;
- PyLoN: 1300BR;
- eXcelon Variants.

E.14.1 What Changed with PICam 5.0?

In PICam 5.0, the following High Capacity (PicamAdcQuality_HighCapacity) parameters have been swapped:

- PicamParameter_ActiveLeftMargin and PicamParameter_ActiveRightMargin

By default, these two parameters remain symmetric, so programs will break only when one, or both, of these parameters have been set asymmetrically in High Capacity.

E.14.2 Code Updates to Retain Existing Behavior

Table E-14 lists code changes required to PICam 4.x (and older) code in order to continue working properly with PICam 5.0.

Table E-14: PICam 4.x (and Older) Code Changes: PyLoN: 100B/100BR/400B/400BR/1300B/1300BR, and eXcelon Variants

<table>
<thead>
<tr>
<th>In PICam 4.x and Older Code, if you:</th>
<th>... in PICam 5.0, you need to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Parameter</td>
</tr>
<tr>
<td>ADC Quality: High Capacity</td>
<td>Get PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td></td>
<td>Set PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td></td>
<td>Get PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td></td>
<td>Set PicamParameter_ActiveRightMargin</td>
</tr>
</tbody>
</table>
E.15 PyLoN: 100F/400F/1300F/1300R Cameras

This section provides information necessary to update customer-modified code (PICam 4.x and older) for the following cameras in order for it to work as designed with PICam 5.0:

- PyLoN: 100F;
- PyLoN: 400F;
- PyLoN: 1300F;
- PyLoN: 1300R.

E.15.1 What Changed with PICam 5.0?

In PICam 5.0, the following Low Noise (PicamAdcQuality_LowNoise) parameters have been swapped:

- PicamParameter_ActiveLeftMargin and PicamParameter_ActiveRightMargin

By default, these two parameters remain symmetric, so programs will break only when one, or both, of these parameters have been set asymmetrically in Low Noise.

E.15.2 Code Updates to Retain Existing Behavior

Table E-15 lists code changes required to PICam 4.x (and older) code in order to continue working properly with PICam 5.0.

Table E-15: PICam 4.x (and Older) Code Changes: PyLoN: 100F/400F/1300F/1300R

<table>
<thead>
<tr>
<th>In PICam 4.x and Older Code, if you:</th>
<th>... in PICam 5.0, you need to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Parameter</td>
</tr>
<tr>
<td>Get PicamParameter_ActiveLeftMargin</td>
<td>Get PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Set PicamParameter_ActiveLeftMargin</td>
<td>Set PicamParameter_ActiveRightMargin</td>
</tr>
<tr>
<td>Get PicamParameter_ActiveRightMargin</td>
<td>Get PicamParameter_ActiveLeftMargin</td>
</tr>
<tr>
<td>Set PicamParameter_ActiveRightMargin</td>
<td>Set PicamParameter_ActiveLeftMargin</td>
</tr>
</tbody>
</table>
Warranty and Service

Limited Warranty

Teledyne Princeton Instruments ("us," "we," "our,"”) makes the following limited warranties. These limited warranties extend to the original purchaser ("You," "you,"”) only and no other purchaser or transferee. We have complete control over all warranties and may alter or terminate any or all warranties at any time we deem necessary.

**Basic Limited One (1) Year Warranty**

Teledyne Princeton Instruments warrants this product against substantial defects in materials and/or workmanship for a period of up to one (1) year after shipment. During this period, Teledyne Princeton Instruments will repair the product or, at its sole option, repair or replace any defective part without charge to you. You must deliver the entire product to the Teledyne Princeton Instruments factory or, at our option, to a factory-authorized service center. You are responsible for the shipping costs to return the product. International customers should contact their local Teledyne Princeton Instruments authorized representative/distributor for repair information and assistance, or visit our technical support page at [www.princetoninstruments.com](http://www.princetoninstruments.com).

**Limited One (1) Year Warranty on Refurbished or Discontinued Products**

Teledyne Princeton Instruments warrants, with the exception of the CCD imaging device (which carries NO WARRANTIES EXPRESS OR IMPLIED,) this product against defects in materials or workmanship for a period of up to one (1) year after shipment. During this period, Teledyne Princeton Instruments will repair or replace, at its sole option, any defective parts, without charge to you. You must deliver the entire product to the Teledyne Princeton Instruments factory or, at our option, a factory-authorized service center. You are responsible for the shipping costs to return the product to Teledyne Princeton Instruments. International customers should contact their local Teledyne Princeton Instruments representative/distributor for repair information and assistance or visit our technical support page at [www.princetoninstruments.com](http://www.princetoninstruments.com).

**XP Vacuum Chamber Limited Lifetime Warranty**

Teledyne Princeton Instruments warrants that the cooling performance of the system will meet our specifications over the lifetime of an XP style detector (has all metal seals) or Teledyne Princeton Instruments will, at its sole option, repair or replace any vacuum chamber components necessary to restore the cooling performance back to the original specifications at no cost to the original purchaser. Any failure to "cool to spec" beyond our [Basic (1) year limited warranty from date of shipment, due to a non-vacuum-related component failure (e.g., any components that are electrical/electronic) is NOT covered and carries NO WARRANTIES EXPRESSED OR IMPLIED. Responsibility for shipping charges is as described above under our Basic Limited One (1) Year Warranty.
Sealed Chamber Integrity Limited 12 Month Warranty

Teledyne Princeton Instruments warrants the sealed chamber integrity of all our products for a period of twelve (12) months after shipment. If, at anytime within twelve (12) months from the date of delivery, the detector should experience a sealed chamber failure, all parts and labor needed to restore the chamber seal will be covered by us. *Open chamber products carry NO WARRANTY TO THE CCD IMAGING DEVICE, EXPRESSED OR IMPLIED.* Responsibility for shipping charges is as described above under our Basic Limited One (1) Year Warranty.

Vacuum Integrity Limited 12 Month Warranty

Teledyne Princeton Instruments warrants the vacuum integrity of "Non-XP" style detectors (do not have all metal seals) for a period of up to twelve (12) months from the date of shipment. We warrant that the detector head will maintain the factory-set operating temperature without the requirement for customer pumping. Should the detector experience a Vacuum Integrity failure at anytime within twelve (12) months from the date of delivery all parts and labor needed to restore the vacuum integrity will be covered by us. Responsibility for shipping charges is as described above under our Basic Limited One (1) Year Warranty.

Image Intensifier Detector Limited One Year Warranty

All image intensifier products are inherently susceptible to Phosphor and/or Photocathode burn (physical damage) when exposed to high intensity light. Teledyne Princeton Instruments warrants, with the exception of image intensifier products that are found to have Phosphor and/or Photocathode burn damage (which carry NO WARRANTIES EXPRESSED OR IMPLIED,) all image intensifier products for a period of one (1) year after shipment. *Refer to additional Limited One (1) year Warranty terms and conditions above, which apply to this warranty.* Responsibility for shipping charges is as described above under our Basic Limited One (1) Year Warranty.

X-Ray Detector Limited One Year Warranty

Teledyne Princeton Instruments warrants, with the exception of CCD imaging device and fiber optic assembly damage due to X-rays (which carry NO WARRANTIES EXPRESSED OR IMPLIED,) all X-ray products for one (1) year after shipment. *Refer to additional Basic Limited One (1) year Warranty terms and conditions above, which apply to this warranty.* Responsibility for shipping charges is as described above under our Basic Limited One (1) Year Warranty.

Software Limited Warranty

Teledyne Princeton Instruments warrants all of our manufactured software discs to be free from substantial defects in materials and/or workmanship under normal use for a period of one (1) year from shipment. Teledyne Princeton Instruments does not warrant that the function of the software will meet your requirements or that operation will be uninterrupted or error free. You assume responsibility for selecting the software to achieve your intended results and for the use and results obtained from the software. In addition, during the one (1) year limited warranty. The original purchaser is entitled to receive free version upgrades. Version upgrades supplied free of charge will be in the form of a download from the Internet. Those customers who do not have access to the Internet may obtain the version upgrades on a CDROM from our factory for an incidental shipping and handling charge. *Refer to Item 12 in Your Responsibility of this warranty for more information.*
Owner’s Manual and Troubleshooting

You should read the owner’s manual thoroughly before operating this product. In the unlikely event that you should encounter difficulty operating this product, the owner’s manual should be consulted before contacting the Teledyne Princeton Instruments technical support staff or authorized service representative for assistance. If you have consulted the owner’s manual and the problem still persists, please contact the Teledyne Princeton Instruments technical support staff or our authorized service representative. Refer to Item 12 in Your Responsibility of this warranty for more information.

Your Responsibility

The above Limited Warranties are subject to the following terms and conditions:

1. You must retain your bill of sale (invoice) and present it upon request for service and repairs or provide other proof of purchase satisfactory to Teledyne Princeton Instruments.

2. You must notify the Teledyne Princeton Instruments factory service center within (30) days after you have taken delivery of a product or part that you believe to be defective. With the exception of customers who claim a “technical issue” with the operation of the product or part, all invoices must be paid in full in accordance with the terms of sale. Failure to pay invoices when due may result in the interruption and/or cancellation of your one (1) year limited warranty and/or any other warranty, expressed or implied.

3. All warranty service must be made by the Teledyne Princeton Instruments factory or, at our option, an authorized service center.

4. Before products or parts can be returned for service you must contact the Teledyne Princeton Instruments factory and receive a return authorization number (RMA.) Products or parts returned for service without a return authorization evidenced by an RMA will be sent back freight collect.

5. These warranties are effective only if purchased from the Teledyne Princeton Instruments factory or one of our authorized manufacturer’s representatives or distributors.

6. Unless specified in the original purchase agreement, Teledyne Princeton Instruments is not responsible for installation, setup, or disassembly at the customer’s location.

7. Warranties extend only to defects in materials or workmanship as limited above and do not extend to any product or part which:
   - has been lost or discarded by you;
   - has been damaged as a result of misuse, improper installation, faulty or inadequate maintenance, or failure to follow instructions furnished by us;
   - has had serial numbers removed, altered, defaced, or rendered illegible;
   - has been subjected to improper or unauthorized repair;
   - has been damaged due to fire, flood, radiation, or other “acts of God,” or other contingencies beyond the control of Teledyne Princeton Instruments; or
   - is a shutter which is a normal wear item and as such carries a onetime only replacement due to a failure within the original 1 year Manufacturer warranty.

8. After the warranty period has expired, you may contact the Teledyne Princeton Instruments factory or a Teledyne Princeton Instruments-authorized representative for repair information and/or extended warranty plans.

9. Physically damaged units or units that have been modified are not acceptable for repair in or out of warranty and will be returned as received.
10. All warranties implied by state law or non-U.S. laws, including the implied warranties of merchantability and fitness for a particular purpose, are expressly limited to the duration of the limited warranties set forth above. With the exception of any warranties implied by state law or non-U.S. laws, as hereby limited, the foregoing warranty is exclusive and in lieu of all other warranties, guarantees, agreements, and similar obligations of manufacturer or seller with respect to the repair or replacement of any parts. In no event shall Teledyne Princeton Instruments’ liability exceed the cost of the repair or replacement of the defective product or part.

11. This limited warranty gives you specific legal rights and you may also have other rights that may vary from state to state and from country to country. Some states and countries do not allow limitations on how long an implied warranty lasts, when an action may be brought, or the exclusion or limitation of incidental or consequential damages, so the above provisions may not apply to you.

12. When contacting us for technical support or service assistance, please refer to the Teledyne Princeton Instruments factory of purchase, contact your authorized Teledyne Princeton Instruments representative or reseller, or visit our technical support page at www.princetoninstruments.com.

Contact Information

Teledyne Princeton Instruments’ manufacturing facility for this product is located at the following address:

Teledyne Princeton Instruments
3660 Quakerbridge Road
Trenton, NJ 08619 (USA)
Tel: 1-800-874-9789 / 1-609-587-9797
Customer Support E-mail: techsupport@princetoninstruments.com

Refer to http://www.princetoninstruments.com/support for complete support and contact information, including:

- Up-to-date addresses and telephone numbers;
- Software downloads;
- Product manuals;
- Support topics for Teledyne Princeton Instruments’ product lines.
This page is intentionally blank.