

## ***PICam™ 5.x Programmer's Manual***

# ***PICam™ 5.x***

## Revision History

Issue	Date	List of Changes
Issue 4	January 2, 2018	Issue 4 of this document incorporates the following changes: <ul style="list-style-type: none"><li>Updated the copyright date.</li></ul>
Issue 3	November 27, 2017	Issue 3 of this document incorporates the following changes: <ul style="list-style-type: none"><li>Updated Linux requirements;</li><li>Updated list of supported devices;</li><li>Updated list of sample codes provided;</li><li>Added <code>PicamCenterWavelengthStatus_Faulted</code> to Section 5.1.1.5, <code>PicamCenterWavelengthStatus</code>.</li></ul>
Issue 2	September 7, 2017	Issue 2 of this document incorporates the following changes: <ul style="list-style-type: none"><li>Added support for Sophia 2048:13.5<math>\mu</math> cameras;</li><li>Added 18-bit support (future enhancement.)</li></ul>
Issue 1	May 9, 2017	This is the initial release of this document.

©Copyright 2017-2018      Princeton Instruments, a division of Roper Scientific, Inc.  
3660 Quakerbridge Rd  
Trenton, NJ 08619  
TEL: 800-874-9789 / 609-587-9797  
FAX: 609-587-1970

All rights reserved. No part of this publication may be reproduced by any means without the written permission of Princeton Instruments, a division of Roper Scientific, Inc. ("Princeton Instruments").

Printed in the United States of America.

PICam is a trademark of Roper Scientific, Inc.

Roper Scientific is a registered trademark of Roper Scientific, Inc.

Windows, Windows 7, Windows 8, and Windows 10 are registered trademarks of Microsoft Corporation in the United States and/or other countries.

Red Hat and Enterprise Linux are registered trademarks of Red Hat, Inc.

The information in this publication is believed to be accurate as of the publication release date. However, Roper Scientific, Inc. does not assume any responsibility for any consequences including any damages resulting from the use thereof. The information contained herein is subject to change without notice. Revision of this publication may be issued to incorporate such change.

# Table of Contents

---

<b>Chapter 1:</b>	<b>About this Manual</b>	<b>9</b>
1.1	Manual Organization	9
1.2	Potential Compatibility Concerns	10
<b>Chapter 2:</b>	<b>Introduction to PICam™</b>	<b>13</b>
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	15
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
<b>Chapter 3:</b>	<b>General Library APIs</b>	<b>23</b>
3.1	Data Type Definitions	23
3.2	Programmers' Reference for General Use Library APIs	24
<b>Chapter 4:</b>	<b>Identification APIs</b>	<b>31</b>
4.1	Data Type Definitions	32
4.2	Structure Definitions	33
4.3	Programmers' API Reference	35
4.3.1	Identification APIs	36
4.3.2	Access APIs	46
4.3.3	Information APIs	60
4.3.4	Demo Camera Identification APIs	64
<b>Chapter 5:</b>	<b>Configuration APIs</b>	<b>69</b>
5.1	Data Type Definitions	70
5.1.1	Hardware Parameter Enumerations	70
5.1.2	Parameter Access Enumerations	105
5.1.3	Parameter Constraint Enumerations	106
5.2	Data Structure Definitions	109
5.2.1	Camera-Specific Parameter Data Structures	109
5.2.2	Shared Camera/Accessory Parameter Data Structures	112
5.3	Parameter Constraints	113
5.3.1	Camera-Specific Parameter Constraints	113
5.3.2	Shared Camera/Accessory Parameter Constraints	117

5.4	Programmers' Reference for Configuration APIs .....	120
5.4.1	Camera-Specific Parameter Value APIs .....	122
5.4.2	Shared Camera/Accessory Parameter Value APIs .....	144
5.4.3	Shared Camera/Accessory Parameter Information APIs .....	164
5.4.4	Camera-Specific Parameter Constraints APIs .....	172
5.4.5	Shared Camera/Accessory Parameter Constraints APIs .....	178
5.4.6	Shared Camera/Accessory Parameter Commitment APIs .....	182

## **Chapter 6: Data Acquisition APIs .....185**

6.1	Data Format .....	186
6.2	Data Type Definitions .....	187
6.2.1	Data Acquisition Enumerations .....	187
6.3	Data Acquisition Data Structures .....	188
6.4	Programmers' Reference for Acquisition Control APIs .....	190

## **Chapter 7: Advanced Function APIs .....197**

7.1	Data Type Definitions .....	198
7.1.1	Shared Camera/Accessory Plug and Play Discovery Data Enumerations ..	198
7.1.2	Shared Camera/Accessory Access Enumerations .....	199
7.1.3	Shared Camera/Accessory Parameter Information Enumerations .....	200
7.1.4	Camera-Specific Data Acquisition Enumerations .....	201
7.2	Data Structures .....	202
7.2.1	Camera-Specific Information Data Structures .....	202
7.2.2	Camera-Specific Parameter Validation Data Structures .....	205
7.2.3	Camera-Specific Data Acquisition Data Structures .....	209
7.3	Callback Functions .....	210
7.3.1	Camera-Specific Discovery Callbacks .....	211
7.3.2	Accessory-Specific Discovery Callbacks .....	211
7.3.3	Camera-Specific Parameter Value Callbacks .....	212
7.3.4	Shared Camera/Accessory Parameter Value Callbacks .....	214
7.3.5	Camera-Specific Parameter Constraints Callbacks .....	217
7.3.6	Shared Camera/Accessory Parameter Constraints Callbacks .....	219
7.3.7	Camera-Specific Data Acquisition Callbacks .....	221
7.4	Programmers' Reference for Advanced APIs .....	222
7.4.1	Camera-Specific Advanced Discovery APIs .....	224
7.4.2	Accessory-Specific Advanced Discovery APIs .....	227
7.4.3	Camera-Specific Advanced Access APIs .....	230
7.4.4	Shared Camera/Accessory Advanced Access APIs .....	234
7.4.5	Camera-Specific Information APIs .....	235
7.4.6	Accessory-Specific Information APIs .....	237
7.4.7	Shared Camera/Accessory Advanced Information APIs .....	238
7.4.8	Camera-Specific Advanced Parameter Value APIs .....	240
7.4.9	Shared Camera/Accessory Advanced Parameter Value APIs .....	248
7.4.10	Shared Camera/Accessory Advanced Parameter Information APIs .....	259
7.4.11	Camera-Specific Advanced Parameter Constraints APIs .....	265
7.4.12	Shared Camera/Accessory Advanced Parameter Constraints APIs .....	274
7.4.13	Camera-Specific Advanced Commitment APIs .....	280
7.4.14	Camera-Specific Advanced Acquisition Setup APIs .....	289
7.4.15	Camera-Specific Advanced Acquisition Notification APIs .....	291
7.4.16	Camera-Specific Advanced Acquisition State Notification APIs .....	293
7.4.17	Camera-Specific Advanced Acquisition Control APIs .....	296

<b>Chapter 8:</b>	<b>EM Calibration APIs</b>	<b>299</b>
8.1	EM Calibration Applications	299
8.2	Structure Definitions	300
8.2.1	EM Calibration Structures	300
8.3	Callback Functions	301
8.3.1	EM Calibration	301
8.4	Programmers' Reference for EM Calibration APIs	302
8.4.1	EM Calibration Access APIs	303
8.4.2	EM Calibration Parameter Value APIs	307
8.4.3	EM Calibration Parameter Constraints APIs	312
8.4.4	EM Calibration APIs	313
<b>Appendix A:</b>	<b>Available Parameters</b>	<b>315</b>
A.1	Camera Parameter Information	315
A.2	Accessory Parameter Information	322
<b>Appendix B:</b>	<b>EM Gain Calibration Code Sample</b>	<b>323</b>
B.1	EM Gain Calibration Procedure	323
<b>Appendix C:</b>	<b>Firmware Upgrade/Restore</b>	<b>325</b>
C.1	Firmware Upgrade Procedure	325
C.2	Restore Firmware	327
C.2.1	Precautions	327
C.2.2	Procedure	328
<b>Appendix D:</b>	<b>Debugging GigE Cameras</b>	<b>331</b>
D.1	Debugging	331
D.1.1	Timeout Period Considerations	332
D.1.2	Following Debugging	332
D.2	Timeout Configuration	332
<b>Appendix E:</b>	<b>PICam 5.0 Compatibility Issues</b>	<b>333</b>
E.1	FERGIE: 256F/FT, FERGIE: 256B/FT, FERGIE: 256BR/FT, and eXcelon Variant Cameras	333
E.1.1	What Changed with PICam 5.0?	333
E.1.2	Code Updates to Retain Existing Behavior	334
E.2	PI-MAX4: 2048B, PI-MAX4: 2048B-RF Cameras	334
E.2.1	What Changed with PICam 5.0?	334
E.2.2	Code Updates to Retain Existing Behavior	334
E.3	PI-MAX4: 512B/EM, PI-MAX4: 1024B/EM	335
E.3.1	What Changed with PICam 5.0?	335
E.3.2	Code Updates to Retain Existing Behavior	335
E.4	PI-MAX4: 512EM/1024EM Cameras	336
E.4.1	What Changed with PICam 5.0?	336
E.4.2	Code Updates to Retain Existing Behavior	336
E.5	PI-MTE: 1300B/1300BR Cameras	337
E.5.1	What Changed with PICam 5.0?	337
E.5.2	Code Updates to Retain Existing Behavior	337
E.6	PI-MTE: 1300R Cameras	338
E.6.1	What Changed with PICam 5.0?	338
E.6.2	Code Updates to Retain Existing Behavior	338

E.7	PIXIS: 100B/100BR/400B/400BR/1300B/1300BR, and XO/XF/XB/ eXcelon Variant Cameras . . . . .	339
E.7.1	What Changed with PICam 5.0? . . . . .	339
E.7.2	Code Updates to Retain Existing Behavior . . . . .	339
E.8	PIXIS: 100F/100R/100C/400F/400R/1300F/1300F-2, and XB Variant Cameras . . . . .	340
E.8.1	What Changed with PICam 5.0? . . . . .	340
E.8.2	Code Updates to Retain Existing Behavior . . . . .	340
E.9	PIXIS: 512F, PIXIS-XO: 512F, PIXIS-XF: 512F Cameras . . . . .	341
E.9.1	What Changed with PICam 5.0? . . . . .	341
E.9.2	Code Updates to Retain Existing Behavior . . . . .	341
E.10	ProEM Cameras (All Models) . . . . .	342
E.10.1	What Changed with PICam 5.0? . . . . .	342
E.10.2	Code Updates to Retain Existing Behavior . . . . .	342
E.11	ProEM-HS: 1KB-10 and eXcelon Variant Cameras . . . . .	343
E.11.1	What Changed with PICam 5.0? . . . . .	343
E.11.2	Code Updates to Retain Existing Behavior . . . . .	343
E.12	ProEM-HS: 512B/512BK/1024B and eXcelon Variant Cameras . . . . .	344
E.12.1	What Changed with PICam 5.0? . . . . .	344
E.12.2	Code Updates to Retain Existing Behavior . . . . .	344
E.13	ProEM+ (All Models) . . . . .	346
E.13.1	What Changed with PICam 5.0? . . . . .	346
E.13.2	Code Updates to Retain Existing Behavior . . . . .	346
E.14	PyLoN: 100B/100BR/400B/400BR/1300B/1300BR, and eXcelon Variant Cameras . . . . .	347
E.14.1	What Changed with PICam 5.0? . . . . .	347
E.14.2	Code Updates to Retain Existing Behavior . . . . .	347
E.15	PyLoN: 100F/400F/1300F/1300R Cameras . . . . .	348
E.15.1	What Changed with PICam 5.0? . . . . .	348
E.15.2	Code Updates to Retain Existing Behavior . . . . .	348

## 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 . . . . .	186
Figure 7-1:	PICam Structure - Advanced . . . . .	197
Figure B-1:	Typical EM Gain Calibration Dialog . . . . .	324
Figure C-1:	Firmware Upgrade: Typical IP Engine Selection Dialog . . . . .	325
Figure C-2:	Firmware Upgrade: Selecting Device to be Upgraded . . . . .	326
Figure C-3:	Firmware Upgrade: Typical Updating Dialog . . . . .	326
Figure C-4:	Firmware Upgrade: Upgrade Complete . . . . .	327
Figure C-5:	Firmware Restore: Typical IP Engine Selection Dialog . . . . .	328
Figure C-6:	Firmware Restore: Selecting Device to be Restored . . . . .	328
Figure C-7:	Firmware Restore: Typical Updating Dialog . . . . .	329
Figure C-8:	Firmware Restore: Complete . . . . .	329

## List of Tables

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 2-3:	Sized Data Type Definitions	19
Table 3-1:	Data Enumeration Definitions for General Library APIs	23
Table 4-1:	Data Type Definitions for Hardware APIs.	32
Table 5-1:	PicamActiveShutter Enumerator Definitions	71
Table 5-2:	PicamAdcAnalogGain Enumerator Definitions	71
Table 5-3:	PicamAdcQuality Enumerator Definitions	72
Table 5-4:	PicamCcdCharacteristicsMask Enumerator Definitions	72
Table 5-5:	PicamCenterWavelengthStatus Enumerator Definitions	73
Table 5-6:	PicamConstraintType Enumerator Definitions	74
Table 5-7:	PicamCoolingFanStatus Enumerator Definitions	74
Table 5-8:	PicamEMICcdGainControlMode Enumerator Definitions	75
Table 5-9:	PicamGateTrackingMask Enumerator Definitions	75
Table 5-10:	PicamGatingMode Enumerator Definitions	76
Table 5-11:	PicamGatingSpeed Enumerator Definitions	76
Table 5-12:	PicamGratingCoating Enumerator Definitions	77
Table 5-13:	PicamGratingType Enumerator Definitions	77
Table 5-14:	PicamIntensifierOptionsMask Enumerator Definitions	78
Table 5-15:	PicamIntensifierStatus Enumerator Definitions	78
Table 5-16:	PicamLaserOutputMode Enumerator Definitions	79
Table 5-17:	PicamLaserStatus Enumerator Definitions	79
Table 5-18:	PicamLightSource Enumerator Definitions	80
Table 5-19:	PicamLightSourceStatus Enumerator Definitions	80
Table 5-20:	PicamModulationTrackingMask Enumerator Definitions	81
Table 5-21:	PicamOrientationMask Enumerator Definitions	81
Table 5-22:	PicamOutputSignal Enumerator Definitions	82
Table 5-23:	PicamParameter Enumerator Definitions	83
Table 5-24:	PicamPhosphorType Enumerator Definitions	93
Table 5-25:	PicamPhotocathodeSensitivity Enumerator Definitions	93
Table 5-26:	PicamPhotonDetectionMode Enumerator Definitions	94
Table 5-27:	PicamPixelFormat Enumerator Definitions	94
Table 5-28:	PicamReadoutControlMode Enumerator Definitions	95
Table 5-29:	PicamSensorTemperatureStatus Enumerator Definitions	96
Table 5-30:	PicamSensorType Enumerator Definitions	96
Table 5-31:	PicamShutterStatus Enumerator Definitions	97
Table 5-32:	PicamShutterTimingMode Enumerator Definitions	98
Table 5-33:	PicamShutterType Enumerator Definitions	99
Table 5-34:	PicamTimeStampsMask Enumerator Definitions	100
Table 5-35:	PicamTriggerCoupling Enumerator Definitions	100
Table 5-36:	PicamTriggerDetermination Enumerator Definitions	101
Table 5-37:	PicamTriggerResponse Enumerator Definitions	101
Table 5-38:	PicamTriggerSource Enumerator Definitions	102
Table 5-39:	PicamTriggerStatus Enumerator Definitions	103
Table 5-40:	PicamTriggerTermination Enumerator Definitions	103
Table 5-41:	PicamValueType Enumerator Definitions	104
Table 5-42:	PicamValueAccess Enumerator Definitions	105
Table 5-43:	PicamConstraintScope Enumerator Definitions	106

Table 5-44:	PicamConstraintSeverity Enumerator Definitions .....	107
Table 5-45:	PicamConstraintCategory Enumerator Definitions .....	107
Table 5-46:	PicamRoisConstraintRulesMask Enumerator Definitions .....	108
Table 6-1:	PicamAcquisitionErrorsMask Enumerator Definitions.....	187
Table 7-1:	PicamDiscoveryAction Enumerator Definitions .....	198
Table 7-2:	PicamHandleType Enumerator Definitions.....	199
Table 7-3:	PicamDynamicsMask Enumerator Definitions .....	200
Table 7-4:	PicamAcquisitionState Enumerator Definitions .....	201
Table 7-5:	PicamAcquisitionStateErrorsMask Enumerator Definitions ...	201
Table A-1:	Symbol Key for <a href="#">Table A-2</a> and <a href="#">Table A-3</a> .....	315
Table A-2:	Parameter Information and Camera Support .....	315
Table A-3:	Parameter Information and Accessory Support .....	322
Table E-1:	PICam 4.x (and Older) Code Changes: FERGIE: 256F/FT, 256B/FT, 256BR/FT, and eXcelon Variants.....	334
Table E-2:	PICam 4.x (and Older) Code Changes: PI-MAX4: 2048B 2048B-RF .....	334
Table E-3:	PICam 4.x (and Older) Code Changes: PI-MAX4: 512B/EM, 1024B/EM .....	335
Table E-4:	PICam 4.x (and Older) Code Changes: PI-MAX4: 512EM/1024EM ..	336
Table E-5:	PICam 4.x (and Older) Code Changes: PI-MTE: 1300B/1300BR.....	337
Table E-6:	PICam 4.x (and Older) Code Changes: PI-MTE: 1300R .....	338
Table E-7:	PICam 4.x (and Older) Code Changes: PIXIS: 100B/100BR/400B/ 400BR/1300B/1300BR, and XO/XF/XB/eXcelon Variants .....	339
Table E-8:	PICam 4.x (and Older) Code Changes: PIXIS: 100F/100R/100C/ 400F/400R/1300F/1300F-2, and XB Variants .....	340
Table E-9:	PICam 4.x (and Older) Code Changes: PIXIS: 512F, PIXIS-XO: 512F, and PIXIS-XF: 512F .....	341
Table E-10:	PICam 4.x (and Older) Code Changes: ProEM (All Models) .....	342
Table E-11:	PICam 4.x (and Older) Code Changes: ProEM-HS: 1KB-10 and eXcelon Variants .....	343
Table E-12:	PICam 4.x (and Older) Code Changes: ProEM-HS: 512B/512BK/ 1024B and eXcelon Variants.....	344
Table E-13:	PICam 4.x (and Older) Code Changes: ProEM+ (All Models) .....	346
Table E-14:	PICam 4.x (and Older) Code Changes: PyLoN: 100B/100BR/400B/ 400BR/1300B/1300BR, and eXcelon Variants.....	347
Table E-15:	PICam 4.x (and Older) Code Changes: PyLoN: 100F/400F/1300F/ 1300R.....	348



# 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 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., `pic_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 333. 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)**

Camera/Camera Family	Section and Page #
FERGIE: 256F/FT, FERGIE: 256B/FT, FERGIE: 256BR/FT, and eXcelon Variant Cameras	<a href="#">Section E.1 on page 333</a>
PI-MAX4: 2048B, PI-MAX4: 2048B-RF Cameras	<a href="#">Section E.2 on page 334</a>
PI-MAX4: 512B/EM, PI-MAX4: 1024B/EM	<a href="#">Section E.3 on page 335</a>
PI-MAX4: 512EM/1024EM Cameras	<a href="#">Section E.4 on page 336</a>
PI-MTE: 1300B/1300BR Cameras	<a href="#">Section E.5 on page 337</a>
PI-MTE: 1300R Cameras	<a href="#">Section E.6 on page 338</a>

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

Camera/Camera Family	Section and Page #
PIXIS: 100B/100BR/400B/400BR/1300B/1300BR, and XO/XF/XB/ eXcelon Variant Cameras	Section E.7 on page 339
PIXIS: 100F/100R/100C/400F/400R/1300F/1300F-2, and XB Variant Cameras	Section E.8 on page 340
PIXIS: 512F, PIXIS-XO: 512F, PIXIS-XF: 512F Cameras	Section E.9 on page 341
ProEM Cameras (All Models)	Section E.10 on page 342
ProEM-HS: 1KB-10 and eXcelon Variant Cameras	Section E.11 on page 343
ProEM-HS: 512B/512BK/1024B and eXcelon Variant Cameras	Section E.12 on page 344
ProEM+ (All Models)	Section E.13 on page 346
PyLoN: 100B/100BR/400B/400BR/1300B/1300BR, and eXcelon Variant Cameras	Section E.14 on page 347
PyLoN: 100F/400F/1300F/1300R Cameras	Section E.15 on page 348

*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 Accessories
- KURO
- PI-MAX3
- PI-MAX4
- PI-MAX4:RF
- PI-MAX4:EM
- PI-MTE
- PloNIR/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 325.
- For information about restoring firmware, refer to [Section C.2, Restore Firmware](#), on page 327.

## 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:

- 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 2-1: List of Sample Code Files Provided (Sheet 1 of 2)**

Sample Name	Description
Accessory	This sample demonstrates control of hardware accessories.
Acquire	This is the basic data acquisition sample. It calls <code>Picam_Acquire()</code> 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.
AcquisitionState	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.)

**Table 2-1: List of Sample Code Files Provided (Sheet 2 of 2)**

Sample Name	Description
Advanced	This sample illustrates features of <code>picam_advanced.h</code> .
Configure	This sample illustrates how to change settings during camera setup as well as online while polling for data.
EMGainCalibration	This sample illustrates how to set up EM Gain Calibration. For additional information about incorporating this sample into production code, refer to <a href="#">Appendix B, EM Gain Calibration Code Sample</a> .
Gating	This sample illustrates how to set up repetitive and sequential gating. Also demonstrates RF features on cameras which support RF functionality.
Kinetics	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.
Metadata	This sample enables metadata (i.e., Time Stamp(s) and Frame Tracking.) It illustrates how to extract metadata from the data stream.
MultiCam	This example opens multiple (i.e., 2,) cameras and collects data from all simultaneously.
ParamInfo	This sample accesses all parameter information for all hardware parameters, and then prints them to the screen.
Poll	This sample illustrates how to use the polling method for collecting data by using <code>Picam_WaitForAcquisitionUpdate()</code> .
Rois	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).
SaveData	This sample acquires data synchronously and writes the returned data buffer to disk.
Spectrograph	This sample moves the center wavelength asynchronously and waits for it to complete.
WaitForTrig	This sample waits for an external trigger to start data acquisition.

## 2.5 Naming Conventions

The following naming conventions are used in PICam:

- All primitive types have a typedef with a `pi` prefix (e.g., `piint`, `pi64s`.)
- All functions defined by PICam are prefixed with `Picam_` and return an error code of `PicamError` (e.g., `Picam_GetParameterIntegerValue`, `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:
  - `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 `const pichar**`.
  - `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 `const PicamCollectionConstraint**`.

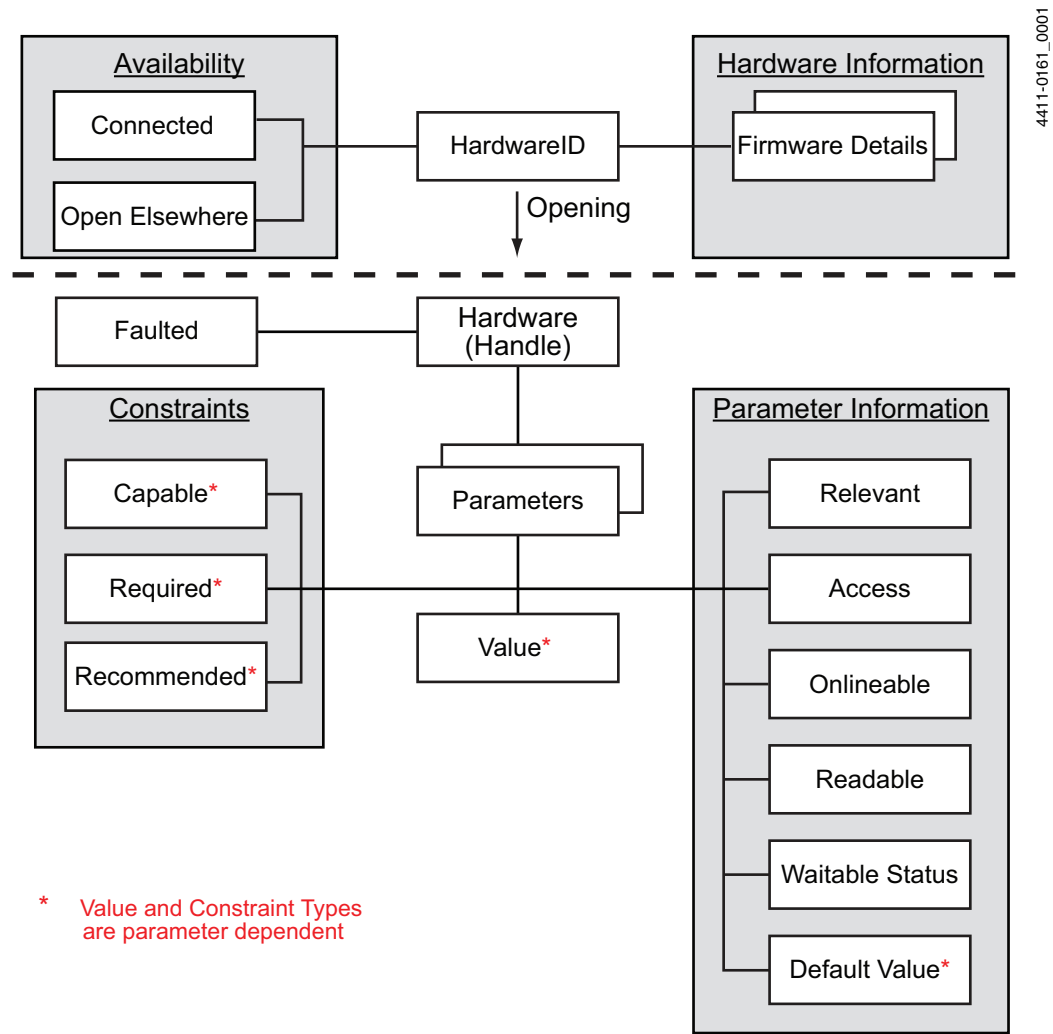
- All functions that allocate an array of memory to store the results of the function call return a pointer to a constant array allocation of the appropriate type as well as the number of items in the array.  
For example, `Picam_GetParameters` returns an array of parameters by taking the addresses of a pointer to a constant parameter array and a count. In other words, two arguments to the function are `const PicamParameter**` and `piint*`.
- All functions that free memory allocated by PICam have a `Picam_Destroy` prefix (e.g., `Picam_DestroyString`, `Picam_Destroy_CollectionConstraints`, `Picam_DestroyRois`.)
- All types defined by PICam are prefixed with `Picam` and have a typedef to `<TypeName>` (e.g., `PicamParameter`, `PicamRoi`.)
- All enum type members defined by PICam are prefixed with `<EnumName>_` (e.g., `PicamValueType` enum has a `PicamValueType_Integer` constant.)
- All enum types that represent multiple values with bitmasks have a `Mask` suffix (e.g., `PicamCcdCharacteristicsMask`, `PicamTimeStampsMask`.)

## 2.6 Concepts

[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.



Figure 2-1: Basic PICam Structure



## 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`.

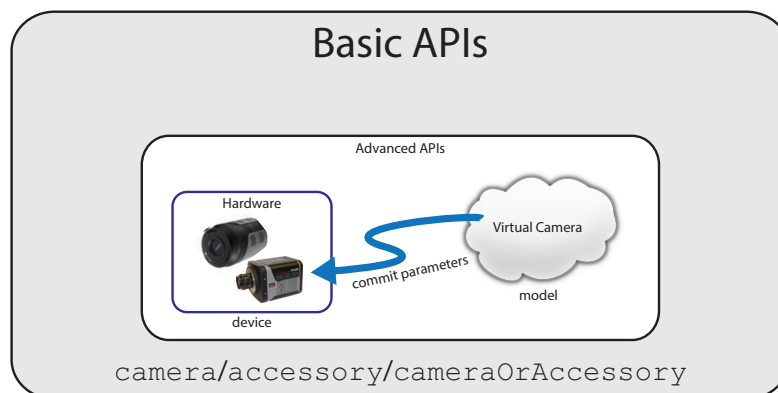
- `model`  
Identifies a specific VIRTUAL camera within application memory.

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 `pic_platform.h`.

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

Type	Definition
piint	Integer native to platform
piflt	Floating point native to platform
pibln	Boolean native to platform
pichar	Character native to platform
pibyte	Byte native to platform
pibool	C++ Boolean native to platform

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

Type	Definition
pi8s	8-bit signed integer
pi8u	8-bit unsigned integer
pi16s	16-bit signed integer
pi16u	16-bit unsigned integer
pi32s	32-bit signed integer
pi32u	32-bit unsigned integer
pi64s	64-bit signed integer
pi64u	64-bit unsigned integer
pi32f	32-bit floating point
pi64f	64-bit floating point

## 2.8 Include Files

Any program using PICam must include the following header files:

- `pic_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.

*This page is intentionally blank.*



# 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 3-1: Data Enumeration Definitions for General Library APIs**

Name	Type	Description
<code>PicamError</code>	enum	The set of error codes returned from all APIs declared as <code>PICAM_API</code> .
<code>PicamEnumeratedType</code>	enum	The set of all PICam enumeration types.

## 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:

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

*continued on next page*



*continued from previous 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.



#### NOTE:

`Picam_UninitializeLibrary()` **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:

```
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:

```
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**

Name	Type	Description
PicamModel	enum	The hardware model.  Series models represent a model family and may be used to represent older hardware whose exact model is not known.
PicamComputerInterface	enum	The interface used to communicate with the hardware.
PicamStringSize	enum	Fixed sizes limiting the maximum size of some picam strings.
PicamHandle	void*	A PICam allocated resource.



## 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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
PICAM_API Picam_GetAvailableCameraIDs(  
    const PicamCameraID** id_array,  
    piint* 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()`.

#### 4.3.1.4 *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:

```
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_GetAvailableCameraIDs()` is:

```
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:

```
PICAM_API PicamAccessory_GetUnavailableAccessoryIDs(  
    const PicamAccessoryID** id_array,  
    piint* 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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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,  
    piint* 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,  
    piint*  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:

```
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:

```
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:

```
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_DestroyFirmwareDetails()`

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:

```
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:

```
PICAM_API PicamAccessory_GetFirmwareDetails(  
    const PicamAccessoryID* id,  
    const PicamFirmwareDetail** firmware_array,  
    piint* 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:

```
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:

```
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 its 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`
- `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

Enumerator	Description
<code>PicamActiveShutter_External</code>	The shutter external to the hardware.
<code>PicamActiveShutter_Internal</code>	The shutter internal to the hardware.
<code>PicamActiveShutter_None</code>	There is no shutter installed.

### 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

Enumerator	Description
<code>PicamAdcAnalogGain_High</code>	Large amplification. Refer to the user manual for the specific hardware being used for complete information.
<code>PicamAdcAnalogGain_Low</code>	Small amplification. Refer to the user manual for the specific hardware being used for complete information.
<code>PicamAdcAnalogGain_Medium</code>	Average amplification. Refer to the user manual for the specific hardware being used for complete information.

### 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**

Enumerator	Description
<code>PicamAdcQuality_ElectronMultiplied</code>	Provides electron multiplication.
<code>PicamAdcQuality_HighCapacity</code>	Optimized for sensing high levels of radiation.
<code>PicamAdcQuality_HighSpeed</code>	Provides faster readout speeds.
<code>PicamAdcQuality_LowNoise</code>	Optimized for the lowest noise.

### 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 (Sheet 1 of 2)**

Enumerator	Description
<code>PicamCcdCharacteristicsMask_AdvancedInverted Mode</code>	The CCD has reduced dark current.
<code>PicamCcdCharacteristicsMask_BackIlluminated</code>	Indicates the type of illumination used. Valid values are: <ul style="list-style-type: none"> <li>1 CCD is back-illuminated</li> <li>0 CCD is front-illuminated</li> </ul>
<code>PicamCcdCharacteristicsMask_DeepDepleted</code>	The CCD is deep depleted.
<code>PicamCcdCharacteristicsMask_ExcelonEnabled</code>	The CCD is enhanced with eXcelon technology.



**Table 5-4:** `PicamCcdCharacteristicsMask` Enumerator Definitions (Sheet 2 of 2)

Enumerator	Description
<code>PicamCcdCharacteristicsMask_HighResistivity</code>	The CCD is enhanced for sensing infrared radiation.
<code>PicamCcdCharacteristicsMask_Multiport</code>	The CCD has multiple readout ports that can be used simultaneously.
<code>PicamCcdCharacteristicsMask_None</code>	No additional characteristics.
<code>PicamCcdCharacteristicsMask_OpenElectrode</code>	The CCD is open electrode.
<code>PicamCcdCharacteristicsMask_SecondaryMask</code>	The CCD has an additional masked area.
<code>PicamCcdCharacteristicsMask_UVEnhanced</code>	The CCD is enhanced for sensing ultraviolet radiation.

### 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

Enumerator	Description
<code>PicamCenterWavelengthStatus_Faulted</code>	The grating drive has malfunctioned.
<code>PicamCenterWavelengthStatus_Moving</code>	The center wavelength is moving.
<code>PicamCenterWavelengthStatus_Stationary</code>	The center wavelength is stationary.

### 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

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

### 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

Enumerator	Description
<i>PicamCoolingFanStatus_ForcedOn</i>	The cooling fan has been forced on to prevent overheating.
<i>PicamCoolingFanStatus_Off</i>	The cooling fan is off.
<i>PicamCoolingFanStatus_On</i>	The cooling fan is on.

### 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

Enumerator	Description
<code>PicamEMICcdGainControlMode_Manual</code>	Allows each gain to be controlled independently.
<code>PicamEMICcdGainControlMode_Optimal</code>	Controls both gains simultaneously as a single emICCD gain.

### 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

Enumerator	Description
<code>PicamGateTrackingMask_Delay</code>	The delay of the gate pulse is tracked.
<code>PicamGateTrackingMask_None</code>	No components are tracked.
<code>PicamGateTrackingMask_Width</code>	The width of the gate pulse is tracked.

### 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

Enumerator	Description
<i>PicamGatingMode_Custom</i>	Custom gate timing.
<i>PicamGatingMode_Disabled</i>	Gating is disabled.
<i>PicamGatingMode_Repetitive</i>	The same gate timing is repeated for each frame.
<i>PicamGatingMode_Sequential</i>	Get timing varies for each frame.

### 5.1.1.11 *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

Enumerator	Description
<i>PicamGatingSpeed_Fast</i>	The gate pulse can be very narrow.
<i>PicamGatingSpeed_Slow</i>	The gate pulse width is limited by the intensifier.

### 5.1.1.12 *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

Enumerator	Description
<code>PicamGratingCoating_Al</code>	Aluminium coated.
<code>PicamGratingCoating_AlMgF2</code>	Aluminium and magnesium fluoride coated.
<code>PicamGratingCoating_Ag</code>	Silver coated.
<code>PicamGratingCoating_Au</code>	Gold coated.

### 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

Enumerator	Description
<code>PicamGratingType_Ruled</code>	Ruled grating.
<code>PicamGratingType_HolographicVisible</code>	Holographic grating for the visible range.
<code>PicamGratingType_HolographicNir</code>	Holographic grating for the near infrared range.
<code>PicamGratingType_HolographicUV</code>	Holographic grating for the ultraviolet range.
<code>PicamGratingType_Mirror</code>	Grating is a mirror.

### 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

Enumerator	Description
<code>PicamIntensifierOptionsMask_Modulation</code>	The intensifier can be modulated.
<code>PicamIntensifierOptionsMask_SubNanosecondGating</code>	The pulse can be gated narrower than a nanosecond.
<code>PicamIntensifierOptionsMask_McpGating</code>	The microchannel plate is gated instead of the photocathode.
<code>PicamIntensifierOptionsMask_None</code>	No additional options.

### 5.1.1.15 *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

Enumerator	Description
<code>PicamIntensifierStatus_PoweredOff</code>	The physical switch is in the off position.
<code>PicamIntensifierStatus_PoweredOn</code>	The physical switch is in the on position.

### 5.1.1.16 *PicamLaserOutputMode*

#### Data Type

`PicamLaserOutputMode` is defined as enum.

#### Description

`PicamLaserOutputMode` is when the laser produces light.

#### Enumerator Definitions

Refer to [Table 5-16](#) for enumerator definitions.

**Table 5-16:** `PicamLaserOutputMode` Enumerator Definitions

Enumerator	Description
<code>PicamLaserOutputMode_Disabled</code>	The laser is off.
<code>PicamLaserOutputMode_ContinuousWave</code>	Output is constant.
<code>PicamLaserOutputMode_Pulsed</code>	output is gated on and off.

### 5.1.1.17 *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

Enumerator	Description
<code>PicamLaserStatus_Disarmed</code>	The hardware key is missing.
<code>PicamLaserStatus_Unarmed</code>	The laser is off.
<code>PicamLaserStatus_Arming</code>	The laser will be enabled momentarily.
<code>PicamLaserStatus_Armed</code>	The laser is ready.

### 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

Enumerator	Description
<i>PicamLightSource_Disabled</i>	No Light Source
<i>PicamLightSource_Hg</i>	Mercury Light Source
<i>PicamLightSource_NeAr</i>	Neon and Argon mixed light source
<i>PicamLightSource_Qth</i>	Quartz Tungsten Halogen light source.

### 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

Enumerator	Description
<i>PicamLightSourceStatus_Unstable</i>	The light source is unstable.
<i>PicamLightSourceStatus_Stable</i>	The light source is stable.



### 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 [Table 5-20](#) for enumerator definitions.

**Table 5-20:** `PicamModulationTrackingMask` Enumerator Definitions

Enumerator	Description
<code>PicamModulationTrackingMask_Duration</code>	The modulation duration is tracked.
<code>PicamModulationTrackingMask_Frequency</code>	The modulation frequency is tracked.
<code>PicamModulationTrackingMask_None</code>	No components are tracked.
<code>PicamModulationTrackingMask_Output SignalFrequency</code>	The modulation output signal frequency is tracked.
<code>PicamModulationTrackingMask_Phase</code>	The modulation phase is tracked.

### 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

Enumerator	Description
<code>PicamOrientationMask_Flipped Horizontally</code>	The data is flipped about the centered, vertical axis relative to normal.
<code>PicamOrientationMask_Flipped Vertically</code>	The data is flipped about the centered, horizontal axis relative to normal.
<code>PicamOrientationMask_Normal</code>	This defines a standard orientation.

### 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

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

### 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 10)

Enumerator	Description
<i>PicamParameter_Accumulations</i>	Controls the number of on-sensor accumulations
<i>PicamParameter_ActiveBottomMargin</i>	Controls the inactive number of rows on the bottom.
<i>PicamParameter_ActiveExtendedHeight</i>	Controls the number of additional active rows that can be used for storage. <b>NOTE:</b> These rows cannot be imaged directly.
<i>PicamParameter_ActiveHeight</i>	Controls the active number of rows.
<i>PicamParameter_ActiveLeftMargin</i>	Controls the inactive number of columns on the left.
<i>PicamParameter_ActiveRightMargin</i>	Controls the inactive number of columns on the right.
<i>PicamParameter_ActiveShutter</i>	Selects the shutter via the <i>PicamActiveShutter</i> data enumeration. Refer to <a href="#">Section 5.1.1.1, <i>PicamActiveShutter</i></a> , on page 71 for additional information.
<i>PicamParameter_ActiveTopMargin</i>	Controls the inactive number of rows on the top.
<i>PicamParameter_ActiveWidth</i>	Controls the active number of columns.
<i>PicamParameter_AdcAnalogGain</i>	Controls the electronic gain of the pixel digitization via the <i>PicamAdcAnalogGain</i> data enumeration. Refer to <a href="#">Section 5.1.1.2, <i>PicamAdcAnalogGain</i></a> , on page 71 for additional information.
<i>PicamParameter_AdcBitDepth</i>	Controls the resolution of the pixel digitization in bits-per-pixel.
<i>PicamParameter_AdcEMGain</i>	Controls the electromagnetic gain in terms of multiples.
<i>PicamParameter_AdcQuality</i>	Controls the nature of pixel digitization via the <i>PicamAdcQuality</i> data enumeration. Refer to <a href="#">Section 5.1.1.3, <i>PicamAdcQuality</i></a> , on page 72 for additional information.
<i>PicamParameter_AdcSpeed</i>	Controls the rate pixels are digitized, in MHz.
<i>PicamParameter_Age</i>	Reports the age measured in minutes.
<i>PicamParameter_AnticiapteTrigger</i>	Uses an external pre-trigger to anticipate an external trigger.
<i>PicamParameter_AuxOutput</i>	Controls the auxiliary output gate pulse.

**Table 5-23: PicamParameter Enumerator Definitions (Sheet 2 of 10)**

Enumerator	Description
PicamParameter_BracketGating	Enables bracket pulsing for intensified hardware.
PicamParameter_CcdCharacteristics	Reports characteristics of a CCD sensor via the <a href="#">PicamCcdCharacteristicsMask</a> data enumeration. Refer to <a href="#">Section 5.1.1.4, PicamCcdCharacteristicsMask</a> , on page 72 for additional information.
PicamParameter_CenterWavelengthReading	Reports the actual position of the center wavelength in nanometers (nm).
PicamParameter_CenterWavelengthSetPoint	Controls the target position of the center wavelength in nanometers (nm). Refer to <a href="#">Section 5.1.1.5, PicamCenterWavelengthStatus</a> , on page 73 for additional information.
PicamParameter_CenterWavelengthStatus	Reports if the center wavelength is moving.
PicamParameter_CleanBeforeExposure	Controls cleaning before each exposure.
PicamParameter_CleanCycleCount	Controls the number of clean cycles to run before acquisition begins.
PicamParameter_CleanCycleHeight	Controls the number of rows in a clean cycle.
PicamParameter_CleanSectionFinalHeight	Controls the final height rows for exponential decomposition cleaning.
PicamParameter_CleanSectionFinalHeightCount	Controls the final height iterations for exponential decomposition cleaning.
PicamParameter_CleanSerialRegister	Controls the cleaning of the serial register itself.
PicamParameter_CleanUntilTrigger	Controls the nature of cleaning while waiting for an external trigger.
PicamParameter_CoolingFanStatus	Reports the status of the cooling fan via the <a href="#">PicamCoolingFanStatus</a> data enumeration. Refer to <a href="#">Section 5.1.1.7, PicamCoolingFanStatus</a> , on page 74 for additional information.
PicamParameter_CorrectPixelBias	Enables pixel bias correction.
PicamParameter_CustomModulationSequence	Customizes a modulation sequence.
PicamParameter_DelayFromPreTrigger	Specifies the delay from pre-trigger to trigger in microseconds (μS).
PicamParameter_DifEndingGate	Controls the second gate pulse in DIF readout in nanoseconds (nS).
PicamParameter_DifStartingGate	Controls the initial gate pulse in DIF readout in nanoseconds (nS).
PicamParameter_DisableCoolingFan	Enables/disables the thermoelectric cooling fan.
PicamParameter_DisabledDataFormatting	Controls the basic processing necessary to receive data in the expected format.
PicamParameter_EMIccdGain	Optimally controls the intensifier gain and electron multiplication gain in emICCD hardware in terms of multiples.
PicamParameter_EMIccdGainControlMode	Determines how the intensifier gain and electron multiplication gain are controlled in emICCD hardware via the <a href="#">PicamEMIccdGainControlMode</a> data enumeration. Refer to <a href="#">Section 5.1.1.8, PicamEMIccdGainControlMode</a> , on page 75, for additional information.

**Table 5-23: PicamParameter Enumerator Definitions (Sheet 3 of 10)**

Enumerator	Description
PicamParameter_EnableAuxOutput	Enables the <b>AUX</b> output pulse.
PicamParameter_EnableIntensifier	Enables the intensifier. <b>NOTE:</b> The intensifier must be enabled and powered on for it to function.
PicamParameter_EnableModulation	Enables RF modulation for intensified hardware.
PicamParameter_EnableModulationOutput Signal	Enables an RF output signal from intensified hardware to be used as the user sees fit.
PicamParameter_EnableNondestructive Readout	Allows the hardware to periodically readout while exposing.
PicamParameter_EnableSensorWindowHeater	Enables the sensor window to heat up in an effort to prevent condensation.
PicamParameter_EnableSyncMaster	Enables <b>SyncMASTER1</b> and <b>SyncMASTER2</b> gate pulses.
PicamParameter_ExactReadoutCountMaximum	Reports the maximum number of readouts the hardware can acquire. <b>NOTE:</b> This does not include non-destructive readouts from hardware that supports the feature.
PicamParameter_ExposureTime	Controls the time the sensor is exposed in milliseconds (mS).
PicamParameter_ExternalShutterStatus	Reports the status of the shutter that is external to the hardware via the <a href="#">PicamShutterStatus</a> data enumeration. Refer to <a href="#">Section 5.1.1.31, PicamShutterStatus</a> , on page 97 for additional information.
PicamParameter_ExternalShutterType	Reports the type of shutter that is external to, and can be driven by, the hardware via the <a href="#">PicamShutterType</a> data enumeration. Refer to <a href="#">Section 5.1.1.33, PicamShutterType</a> , on page 99 for additional information.
PicamParameter_FocalLength	Reports the optical path length from the focusing mirror to the sensor in millimeters (mm).
PicamParameter_FrameRateCalculation	Reports the estimated frame rate in frames-per-second. <b>NOTE:</b> If there is more than one frame-per-readout, this represents the burst frame rate within the readout. <b>NOTE:</b> If the hardware is being externally triggered, this represents the fastest possible rate.
PicamParameter_FrameSize	Reports the size, in bytes, of a data frame.
PicamParameter_FramesPerReadout	Reports the number of frames contained in one readout.
PicamParameter_FrameStride	Reports the length, in bytes, necessary to traverse to the next frame.
PicamParameter_FrameTrackingBitDepth	Controls the frame tracking number size in bits-per-pixel.
PicamParameter_GateTracking	Controls the tracking of a sequential gate pulse in metadata via the <a href="#">PicamGateTrackingMask</a> data enumeration. Refer to <a href="#">Section 5.1.1.9, PicamGateTrackingMask</a> , on page 75 for additional information.

**Table 5-23:** `PicamParameter` Enumerator Definitions (Sheet 4 of 10)

Enumerator	Description
<code>PicamParameter_GateTrackingBitDepth</code>	Controls the size of one component in a varying sequential gate pulse. <b>NOTE:</b> This metadata is floating point.
<code>PicamParameter_GatingMode</code>	Controls the nature of gate pulse timing via the <code>PicamGatingMode</code> data enumeration. Refer to <a href="#">Section 5.1.1.10, <code>PicamGatingMode</code></a> , on page 76 for additional information.
<code>PicamParameter_GatingSpeed</code>	Classifies the narrowest gate pulse. Refer to <a href="#">Section 5.1.1.11, <code>PicamGatingSpeed</code></a> , on page 76 for additional information.
<code>PicamParameter_GratingBlazingWavelength</code>	Reports the blaze at a particular wavelength.
<code>PicamParameter_GratingCoating</code>	Reports the coating on the grating. Refer to <a href="#">Section 5.1.1.12, <code>PicamGratingCoating</code></a> , on page 77 for additional information.
<code>PicamParameter_GratingGrooveDensity</code>	Reports the groove density of the grating in grooves per millimeter.
<code>PicamParameter_GratingType</code>	Reports the type of grating. Refer to <a href="#">Section 5.1.1.13, <code>PicamGratingType</code></a> , on page 77 for additional information.
<code>PicamParameter_InactiveShutterTimingModeResult</code>	Reports the state of the inactive shutter via the <code>PicamShutterTimingMode</code> data enumeration. Refer to <a href="#">Section 5.1.1.32, <code>PicamShutterTimingMode</code></a> , on page 98 for additional information.
<code>PicamParameter_InclusionAngle</code>	Reports the sum of the incident and diffracted ray angles relative to the grating normal vector in degrees.
<code>PicamParameter_InputTriggerStatus</code>	Reports if an external trigger source is connected. Refer to <a href="#">Section 5.1.1.39, <code>PicamTriggerStatus</code></a> , on page 103 for additional information.
<code>PicamParameter_IntensifierDiameter</code>	Reports the diameter of the intensifier in millimeters (mm).
<code>PicamParameter_IntensifierGain</code>	Controls the gain of the intensifier in terms of multiples.
<code>PicamParameter_IntensifierOptions</code>	Reports additional features of intensified hardware via the <code>PicamIntensifierOptionsMask</code> data enumeration. Refer to <a href="#">Section 5.1.1.14, <code>PicamIntensifierOptionsMask</code></a> , on page 78 for additional information.
<code>PicamParameter_IntensifierStatus</code>	Reports the status of the intensifier power via the <code>PicamIntensifierStatus</code> data enumeration. Refer to <a href="#">Section 5.1.1.15, <code>PicamIntensifierStatus</code></a> , on page 78 for additional information.
<code>PicamParameter_InternalShutterStatus</code>	Reports the status of the shutter that is internal to the hardware via the <code>PicamShutterStatus</code> data enumeration. Refer to <a href="#">Section 5.1.1.31, <code>PicamShutterStatus</code></a> , on page 97 for additional information.

**Table 5-23:** `PicamParameter` Enumerator Definitions (Sheet 5 of 10)

Enumerator	Description
<code>PicamParameter_InternalShutterType</code>	Reports the type of shutter that is internal to, and can be driven by, the hardware via the <code>PicamShutterType</code> data enumeration.  Refer to <a href="#">Section 5.1.1.33, <code>PicamShutterType</code></a> , on page 99 for additional information.
<code>PicamParameter_InvertOutputSignal</code>	Controls if the timing signal is inverted when viewed from the hardware monitor.
<code>PicamParameter_InvertOutputSignal2</code>	Controls if the timing signal is inverted when viewed from the second hardware monitor.
<code>PicamParameter_KineticsWindowHeight</code>	Controls the number of rows used for the sensing window in a kinetics readout.
<code>PicamParameter_LaserOutputMode</code>	Controls when the laser produces light.  Refer to <a href="#">Section 5.1.1.16, <code>PicamLaserOutputMode</code></a> , on page 79 for additional information.
<code>PicamParameter_LaserPower</code>	Controls the laser power as a multiplier.
<code>PicamParameter_LaserStatus</code>	Reports if the laser is ready.  Refer to <a href="#">Section 5.1.1.17, <code>PicamLaserStatus</code></a> , on page 79 for additional information.
<code>PicamParameter_LifeExpectancy</code>	Reports the expected lifetime measured in minutes.
<code>PicamParameter_LightSource</code>	Controls the light source on the lamp.  Refer to <a href="#">Section 5.1.1.18, <code>PicamLightSource</code></a> , on page 80 for additional information.
<code>PicamParameter_LightSourceStatus</code>	Reports if the light source has stabilized.  Refer to <a href="#">Section 5.1.1.19, <code>PicamLightSourceStatus</code></a> , on page 80 for additional information.
<code>PicamParameter_MaskedBottomMargin</code>	Controls the number of masked rows akin to active bottom margin.
<code>PicamParameter_MaskedHeight</code>	Controls the number of masked rows akin to active height.
<code>PicamParameter_MaskedTopMargin</code>	Controls the number of masked rows akin to active top margin.
<code>PicamParameter_ModulationDuration</code>	Controls the time the intensifier is modulating in milliseconds (mS).
<code>PicamParameter_ModulationFrequency</code>	Controls the frequency of the intensifier modulation in MHz.
<code>PicamParameter_ModulationOutputSignalAmplitude</code>	Controls the peak-to-peak amplitude of the user RF output signal in volts (V).
<code>PicamParameter_ModulationOutputSignalFrequency</code>	Controls the frequency of the user RF output signal in MHz
<code>PicamParameter_ModulationTracking</code>	Controls the tracking of a sequential phase or custom modulation sequence in metadata via the <code>PicamModulationTrackingMask</code> data enumeration.  Refer to <a href="#">Section 5.1.1.20, <code>PicamModulationTrackingMask</code></a> , on page 81 for additional information.

**Table 5-23:** PicamParameter Enumerator Definitions (Sheet 6 of 10)

Enumerator	Description
PicamParameter_ModulationTrackingBitDepth	Controls the size of one component in a varying sequential modulation phase or custom modulation sequence. <b>NOTE:</b> This metadata is floating point.
PicamParameter_NondestructiveReadoutPeriod	Controls the rate at which the hardware will non-destructively readout during exposure in seconds (S). <b>NOTE:</b> 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_OnlineReadoutRateCalculation	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 <a href="#">PicamOrientationMask</a> data enumeration. Refer to <a href="#">Section 5.1.1.21, PicamOrientationMask</a> , on page 81 for additional information.
PicamParameter_OutputSignal	Controls what timing signal is issued from the hardware monitor via the <a href="#">PicamOutputSignal</a> data enumeration. Refer to <a href="#">Section 5.1.1.22, PicamOutputSignal</a> , on page 82 for additional information.
PicamParameter_OutputSignal2	Controls what timing signal is issued from the second hardware monitor via the <a href="#">PicamOutputSignal</a> data enumeration. Refer to <a href="#">Section 5.1.1.22, PicamOutputSignal</a> , on page 82 for additional information.
PicamParameter_PhosphorDecayDelay	Controls the length of time the hardware waits for the phosphor to decay before reading out. The time unit depends on <a href="#">PicamParameter_PhosphorDecayDelay Resolution</a> .
PicamParameter_PhosphorDecayDelayResolution	Controls the time unit used for phosphor decay delay. This value is in microseconds ( $\mu$ S). <b>Example:</b> A resolution of: <ul style="list-style-type: none"> <li>• 1 signifies delay is in microseconds (<math>\mu</math>S);</li> <li>• 1000 for milliseconds (mS);</li> <li>• 0.01 for tens-of-nanoseconds (nS).</li> </ul>
PicamParameter_PhosphorType	Reports the type of phosphor used in intensified hardware. Refer to <a href="#">Section 5.1.1.24, PicamPhosphorType</a> , on page 93 for additional information.
PicamParameter_PhotocathodeSensitivity	Classifies the wavelength sensitivity of the photocathode. Refer to <a href="#">Section 5.1.1.25, PicamPhotocathodeSensitivity</a> , on page 93 for additional information.
PicamParameter_PhotonDetectionMode	Enables/disables photon detection and controls how it is done via the <a href="#">PicamPhotonDetectionMode</a> data enumeration. Refer to <a href="#">Section 5.1.1.26, PicamPhotonDetectionMode</a> , on page 94 for additional information.



**Table 5-23:** `PicamParameter` Enumerator Definitions (Sheet 7 of 10)

Enumerator	Description
<code>PicamParameter_PhotonDetectionThreshold</code>	The threshold, in counts, used to distinguish photons from background for each pixel.
<code>PicamParameter_PixelBitDepth</code>	Reports the size of a data pixel in bits-per-pixel.
<code>PicamParameter_PixelFormat</code>	Controls the format of a data pixel via the <code>PicamPixelFormat</code> data enumeration. Refer to <a href="#">Section 5.1.1.27, <code>PicamPixelFormat</code></a> , on page 94 for additional information.
<code>PicamParameter_PixelGapHeight</code>	Reports the vertical distance between pixels, in microns.
<code>PicamParameter_PixelGapWidth</code>	Reports the horizontal distance between pixels, in microns.
<code>PicamParameter_PixelHeight</code>	Reports the pixel height, in microns.
<code>PicamParameter_PixelWidth</code>	Reports the pixel width, in microns.
<code>PicamParameter_ReadoutControlMode</code>	Controls how the sensor is read out via the <code>PicamReadoutControlMode</code> data enumeration. Refer to <a href="#">Section 5.1.1.28, <code>PicamReadoutControlMode</code></a> , on page 95 for additional information.
<code>PicamParameter_ReadoutCount</code>	Controls the number of readouts to acquire before stopping the hardware. <b>NOTE:</b> The hardware may acquire more than the readouts requested for large requests (i.e., more readouts than those specified by <code>PicamParameter_ExactReadoutCountMaximum</code> ). <b>NOTE:</b> This does not include non-destructive readouts from hardware that supports such a feature. <b>NOTE: [Advanced-API Usage Only]</b> The value 0 indicates the hardware will run forever until explicitly stopped or an error occurs.
<code>PicamParameter_ReadoutOrientation</code>	Reports the orientation of the data due to readout port location via the <code>PicamOrientationMask</code> data enumeration. Refer to <a href="#">Section 5.1.1.21, <code>PicamOrientationMask</code></a> , on page 81 for additional information.
<code>PicamParameter_ReadoutPortCount</code>	Controls the number of readout ports from which the hardware should simultaneously read data.
<code>PicamParameter_ReadoutRateCalculation</code>	Reports the estimated rate of data in readouts-per-second. <b>NOTE:</b> If the hardware is being externally triggered, this represents the fastest possible rate.
<code>PicamParameter_ReadoutStride</code>	Reports the length, in bytes, necessary to traverse to the next readout.
<code>PicamParameter_ReadoutTimeCalculation</code>	Reports the duration of time it takes for the hardware to read out one readout in milliseconds (mS).
<code>PicamParameter_RepetitiveGate</code>	Controls the constant gate pulse in nanoseconds (nS).
<code>PicamParameter_RepetitiveModulationPhase</code>	Controls the constant phase of the intensifier with respect to the modulation output signal in degrees.

**Table 5-23: PicamParameter Enumerator Definitions (Sheet 8 of 10)**

Enumerator	Description
PicamParameter_Rois	Controls the area of the sensor to be digitized via the <a href="#">PicamRois</a> structure.  Refer to <a href="#">Section 5.2.1.2, PicamRois</a> , on page 110 for additional information.
PicamParameter_SecondaryActiveHeight	Controls the number of secondary active rows.
PicamParameter_SecondaryMaskedHeight	Controls the number of secondary masked rows.
PicamParameter_SensorActiveBottomMargin	Reports the inactive rows on the bottom.
PicamParameter_SensorActiveExtendedHeight	Reports the number of additional active rows that can be used for storage. <b>NOTE:</b> These rows cannot be imaged directly.
PicamParameter_SensorActiveHeight	Reports the active number of rows.
PicamParameter_SensorActiveLeftMargin	Reports the inactive columns on the left.
PicamParameter_SensorActiveRightMargin	Reports the inactive columns on the right.
PicamParameter_SensorActiveTopMargin	Reports the inactive rows on the top.
PicamParameter_SensorActiveWidth	Reports the active number of columns.
PicamParameter_SensorAngle	Reports the angle between rays striking the sensor relative to the sensor normal vector in degrees.
PicamParameter_SensorMaskedBottomMargin	Reports the number of masked rows akin to active bottom margin.
PicamParameter_SensorMaskedHeight	Reports the number of masked rows akin to active height.
PicamParameter_SensorMaskedTopMargin	Reports the number of masked rows akin to active top margin.
PicamParameter_SensorSecondaryActiveHeight	Reports the number of secondary active rows.
PicamParameter_SensorSecondaryMaskedHeight	Reports the number of secondary masked rows.
PicamParameter_SensorTemperatureReading	Reports the temperature of the sensor in degrees C.
PicamParameter_SensorTemperatureSetPoint	Controls the target temperature for the sensor in degrees C.
PicamParameter_SensorTemperatureStatus	Reports the status of the sensor temperature via the <a href="#">PicamSensorTemperatureStatus</a> data enumeration.  Refer to <a href="#">Section 5.1.1.29, PicamSensorTemperatureStatus</a> , on page 96 for additional information.
PicamParameter_SensorType	Reports the kind of sensor being used via the <a href="#">PicamSensorType</a> data enumeration.  Refer to <a href="#">Section 5.1.1.30, PicamSensorType</a> , on page 96 for additional information.
PicamParameter_SeNsRWindowHeight	Controls the height of the unmasked area used in SeNsR readout.
PicamParameter_SequentialEndingGate	Controls the last gate pulse in a sequence in nanoseconds (nS).
PicamParameter_SequentialEndingModulationPhase	Controls the last modulation phase of the intensifier with respect to the modulation output signal in a sequence in degrees.

**Table 5-23:** `PicamParameter` Enumerator Definitions (Sheet 9 of 10)

Enumerator	Description
<code>PicamParameter_SequentialGateStepCount</code>	Controls the number of gate pulse steps in a sequence.
<code>PicamParameter_SequentialGateStepIterations</code>	Controls the number of gate pulses at each step in a sequence.
<code>PicamParameter_SequentialStartingGate</code>	Controls the first gate pulse in a sequence in nanoseconds (nS).
<code>PicamParameter_SequentialStartingModulationPhase</code>	Controls the first modulation phase of the intensifier with respect to the modulation output signal in a sequence in degrees.
<code>PicamParameter_ShutterClosingDelay</code>	Controls the duration of time the hardware waits for the shutter to close before reading out.  The time unit depends on <a href="#">PicamParameter_ShutterDelayResolution</a> .
<code>PicamParameter_ShutterDelayResolution</code>	Controls the time unit used for shutter opening/closing delay. This value is in microseconds.  <b>Example:</b> A resolution of: <ul style="list-style-type: none"> <li>• 1 signifies delay is in microseconds (<math>\mu</math>S);</li> <li>• 1000 for milliseconds (mS);</li> <li>• 0.01 for tens-of-nanoseconds (nS).</li> </ul>
<code>PicamParameter_ShutterOpeningDelay</code>	Controls the duration of time the hardware waits for the shutter to open before exposing.  The time unit depends on <a href="#">PicamParameter_ShutterDelayResolution</a> .
<code>PicamParameter_ShutterTimingMode</code>	Controls the behavior of the shutter during acquisition.  Refer to <a href="#">Section 5.1.1.32, PicamShutterTimingMode</a> , on page 98 for additional information.
<code>PicamParameter_StopCleaningOnPreTrigger</code>	Stops sensor cleaning when an external pre-trigger is acknowledged.
<code>PicamParameter_SyncMaster2Delay</code>	Controls the delay of <b>SyncMASTER2</b> relative to <b>SyncMASTER1</b> in microseconds ( $\mu$ S).
<code>PicamParameter_TimeStampBitDepth</code>	Controls the time stamp size in bits-per-pixel. <b>NOTE:</b> Because time stamps may be negative one bit is reserved for sign.
<code>PicamParameter_TimeStampResolution</code>	Controls the time stamp resolution in ticks-per-second. <b>NOTE:</b> This value is computer-dependent when time stamps are software generated.
<code>PicamParameter_TimeStamps</code>	Controls time stamp metadata via the <a href="#">PicamTimeStampsMask</a> data enumeration.  Refer to <a href="#">Section 5.1.1.34, PicamTimeStampsMask</a> , on page 100 for additional information.
<code>PicamParameter_TrackFrames</code>	Controls frame tracking metadata.
<code>PicamParameter_TriggerCoupling</code>	Controls the coupling between an external trigger source and the camera input via the <a href="#">PicamTriggerCoupling</a> data enumeration.  Refer to <a href="#">Section 5.1.1.35, PicamTriggerCoupling</a> , on page 100 for additional information.

**Table 5-23:** `PicamParameter` Enumerator Definitions (Sheet 10 of 10)

Enumerator	Description
<code>PicamParameter_TriggerDelay</code>	Controls the delay between an external trigger and the camera's response in nanoseconds.
<code>PicamParameter_TriggerDetermination</code>	Controls what the hardware recognizes as an external trigger via the <a href="#">PicamTriggerDetermination</a> data enumeration. Refer to <a href="#">Section 5.1.1.36, PicamTriggerDetermination</a> , on page 101 for additional information.
<code>PicamParameter_TriggerFrequency</code>	Controls the internal trigger and <b>SyncMASTER</b> frequency in Hz.
<code>PicamParameter_TriggerResponse</code>	Controls the hardware's behavior in response to a trigger via the <a href="#">PicamTriggerResponse</a> data enumeration. Refer to <a href="#">Section 5.1.1.37, PicamTriggerResponse</a> , on page 101 for additional information.
<code>PicamParameter_TriggerResponse</code>	Controls the hardware's behavior in response to an external trigger via the <a href="#">PicamTriggerResponse</a> data enumeration. Refer to <a href="#">Section 5.1.1.37, PicamTriggerResponse</a> , on page 101 for additional information.
<code>PicamParameter_TriggerSource</code>	Controls the source of a trigger via the <a href="#">PicamTriggerSource</a> data enumeration. Refer to <a href="#">Section 5.1.1.38, PicamTriggerSource</a> , on page 102 for additional information.
<code>PicamParameter_TriggerTermination</code>	Controls the termination of an external trigger source at the hardware input via the <a href="#">PicamTriggerTermination</a> data enumeration. Refer to <a href="#">Section 5.1.1.40, PicamTriggerTermination</a> , on page 103 for additional information.
<code>PicamParameter_TriggerThreshold</code>	Controls the voltage threshold necessary for the hardware to recognize a trigger in volts (V).
<code>PicamParameter_VerticalShiftRate</code>	Controls the rate to shift one row towards the serial register in a CCD in microseconds ( $\mu$ S).

### 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

Enumerator	Description
<code>PicamPhosphorType_P43</code>	The phosphor is P43.
<code>PicamPhosphorType_P46</code>	The phosphor is P46.

### 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

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

### 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

Enumerator	Description
<code>PicamPhotonDetectionMode_Clipping</code>	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.
<code>PicamPhotonDetectionMode_Disabled</code>	Photon detection is disabled.
<code>PicamPhotonDetectionMode_Thresholding</code>	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.

### 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

Enumerator	Description
<code>PicamPixelFormat_Monochrome16Bit</code>	16 bits of monochrome data
<code>PicamPixelFormat_Monochrome32Bit</code>	32 bits of monochrome data

### 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

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

### 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

Enumerator	Description
<code>PicamSensorTemperatureStatus_Faulted</code>	Sensor cooling has malfunctioned.
<code>PicamSensorTemperatureStatus_Locked</code>	The temperature has stabilized at the set point.
<code>PicamSensorTemperatureStatus_Unlocked</code>	The temperature has not stabilized at the set point.

### 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

Enumerator	Description
<code>PicamSensorType_Ccd</code>	The sensor is a CCD.
<code>PicamSensorType_Cmos</code>	The sensor is a CMOS.
<code>PicamSensorType_InGaAs</code>	The sensor is an InGaAs.



### 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

Enumerator	Description
<code>PicamShutterStatus_Connected</code>	A shutter is connected.
<code>PicamShutterStatus_NotConnected</code>	No shutter is connected.
<code>PicamShutterStatus_Overheated</code>	<p>A connected shutter has overheated and is temporarily disabled.</p> <p><b>NOTE:</b> If a shutter becomes overheated, data acquisition will stop and cannot be started again until the shutter is no longer overheated.</p>

### 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

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

### 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

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

### 5.1.1.34 *PicamTimestampsMask*

#### Data Type

*PicamTimestampsMask* is defined as enum.

#### Description

*PicamTimestampsMask* is the set of timestamp metadata.

#### Enumerator Definitions

Refer to [Table 5-34](#) for enumerator definitions.

**Table 5-34:** *PicamTimestampsMask* Enumerator Definitions

Enumerator	Description
<i>PicamTimestampsMask_None</i>	No time stamps are generated during acquisition.
<i>PicamTimestampsMask_ExposureStarted</i>	The time will be stamped when exposure starts.
<i>PicamTimestampsMask_ExposureEnded</i>	The time will be stamped when exposure ends.

### 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

Enumerator	Description
<i>PicamTriggerCoupling_AC</i>	The components are AC-coupled.
<i>PicamTriggerCoupling_DC</i>	The components are DC-coupled.

### 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

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

### 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 (Sheet 1 of 2)

Enumerator	Description
<code>PicamTriggerResponse_ExposeDuringTriggerPulse</code>	Each trigger controls when exposure begins and ends.
<code>PicamTriggerResponse_GatePerTrigger</code>	The hardware generates a gate pulse after the trigger.
<code>PicamTriggerResponse_NoResponse</code>	The hardware does not respond to triggering.

**Table 5-37: PicamTriggerResponse Enumerator Definitions (Sheet 2 of 2)**

Enumerator	Description
PicamTriggerResponse_ReadoutPerTrigger	The hardware reads out the sensor after each trigger. <b>NOTE:</b> 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.
PicamTriggerResponse_ShiftPerTrigger	Each trigger moves to the next frame on the sensor.
PicamTriggerResponse_StartOnSingleTrigger	The hardware begins acquisition after a single trigger.

### 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**

Enumerator	Description
PicamTriggerSource_External	Triggers originate from an external source.
PicamTriggerSource_Internal	Triggers originate from the hardware itself.
PicamTriggerSource_None	No triggers.

### 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

Enumerator	Description
<code>PicamTriggerStatus_NotConnected</code>	No trigger source is connected.
<code>PicamTriggerStatus_Connected</code>	A trigger source is connected.

### 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

Enumerator	Description
<code>PicamTriggerTermination_FiftyOhms</code>	The trigger terminates into 50 ohms.
<code>PicamTriggerTermination_HighImpedance</code>	The trigger terminates into very high impedance.

### 5.1.1.41 *PicamValueType*

#### Data Type

*PicamValueType* is defined as `enum`.

#### Description

*PicamValueType* is the set of parameter value data types.

#### Enumerator Definitions

Refer to [Table 5-41](#) for enumerator definitions.

**Table 5-41:** *PicamValueType* Enumerator Definitions

Enumerator	Description
<i>PicamValueType_Boolean</i>	Accessed as <code>piint</code> . <ul style="list-style-type: none"><li>• <code>FALSE</code> = 0</li><li>• <code>TRUE</code> = 1</li></ul>
<i>PicamValueType_Enumeration</i>	Any enum accessed as <code>piint</code> .
<i>PicamValueType_FloatingPoint</i>	Accessed as <code>piflt</code> .
<i>PicamValueType_Integer</i>	Accessed as <code>piint</code> .
<i>PicamValueType_LargeInteger</i>	Accessed as <code>pi64s</code> .
<i>PicamValueType_Modulations</i>	Accessed as <code>PicamModulations</code>
<i>PicamValueType_Pulse</i>	Accessed as <code>PicamPulse</code> .
<i>PicamValueType_Rois</i>	Accessed as <code>PicamRois</code> .



## 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 5-42](#) for enumerator definitions.

**Table 5-42:** `PicamValueAccess` Enumerator Definitions

Enumerator	Description
<code>PicamValueAccess_ReadOnly</code>	The stored parameter value can only be read.
<code>PicamValueAccess_ReadWriteTrivial</code>	The stored parameter value can be read and/or overwritten, but there is only one value for this parameter.
<code>PicamValueAccess_ReadWrite</code>	The stored parameter value can be read and/or overwritten.

### 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 5-43](#) for enumerator definitions.

**Table 5-43:** [PicamConstraintScope](#) Enumerator Definitions

Enumerator	Description
<a href="#">PicamConstraintScope_Independent</a>	The constraint has no dependencies and is therefore constant.
<a href="#">PicamConstraintScope_Dependent</a>	The constraint has dependencies and therefore is variable.

### 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 5-44](#) for enumerator definitions.

**Table 5-44:** `PicamConstraintSeverity` Enumerator Definitions

Enumerator	Description
<code>PicamConstraintSeverity_Error</code>	Failure indicates the value is in error.
<code>PicamConstraintSeverity_Warning</code>	Failure indicates the value is in warning and notice should be taken.

### 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 5-45](#) for enumerator definitions.

**Table 5-45:** `PicamConstraintCategory` Enumerator Definitions

Enumerator	Description
<code>PicamConstraintCategory_Capable</code>	Which set of values are ultimately possible.
<code>PicamConstraintCategory_Required</code>	Which set of values are currently permissible.
<code>PicamConstraintCategory_Recommended</code>	Which set of values fall within a recommended range for most scenarios.

### 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 5-46](#) for enumerator definitions.

**Table 5-46:** `PicamRoisConstraintRulesMask` Enumerator Definitions

Enumerator	Description
<code>PicamRoisConstraintRulesMask_None</code>	No additional rules.
<code>PicamRoisConstraintRulesMask_XBinningAlignment</code>	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.
<code>PicamRoisConstraintRulesMask_YBinningAlignment</code>	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.
<code>PicamRoisConstraintRulesMask_HorizontalSymmetry</code>	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.
<code>PicamRoisConstraintRulesMask_VerticalSymmetry</code>	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.
<code>PicamRoisConstraintRulesMask_SymmetryBoundsBinning</code>	A region required to bisect a line of symmetry may not bin pixels together that fall on both sides of the line.

## 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:

```
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:

```
typedef struct PicamRois
{
    PicamRoi*   roi_array;
    piint      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:

```
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:

```
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:

```
typedef struct PicamModulations
{
    PicamModulation* modulation_array;
    pint 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:

```
typedef struct PicamStatusPurview
{
    const piint*  values_array;
    piint  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/Accessory 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:

```
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*

*continued from previous page*

## Variable Definitions

The variables required by `PicamRoisConstraint` are:

<code>scope:</code>	The scope of the constraint.
<code>severity:</code>	The severity of the constraint.
<code>empty_set:</code>	Indicates when there are no valid Regions of Interest defined. Valid values are: <ul style="list-style-type: none"> <li>• <code>TRUE</code> There are no valid ROIs defined. When <code>TRUE</code>, only <code>scope</code> and <code>severity</code> are relevant.</li> <li>• <code>FALSE</code> There is at least one valid ROI defined.</li> </ul>
<code>rules:</code>	Complex set of rules to which a parameter of this type must adhere.
<code>maximum_roi_count:</code>	The maximum number of ROIs permitted.
<code>x_constraint:</code>	The constraint governing the value of <code>PicamRoi.x</code> .
<code>width_constraint:</code>	The constraint governing the value of <code>PicamRoi.width</code> .
<code>x_binning_limits_array:</code>	The list of valid values for <code>PicamRoi.x_binning</code> . <b>NOTE:</b> An additional requirement is that <code>PicamRoi.x_binning</code> must always divide evenly into <code>PicamRoi.width</code> . This is null when no additional limits are required.
<code>x_binning_limits_count:</code>	The number of items in <code>x_binning_limits_array</code> . This is 0 when no additional limits are required.
<code>y_constraint:</code>	The constraint governing the value of <code>PicamRoi.y</code> .
<code>height_constraint:</code>	The constraint governing the value of <code>PicamRoi.height</code> .
<code>y_binning_limits_array:</code>	The list of valid values for <code>PicamRoi.y_binning</code> . <b>NOTE:</b> An additional requirement is that <code>PicamRoi.y_binning</code> must always divide evenly into <code>PicamRoi.height</code> . This is null when no additional limits are required.
<code>y_binning_limits_count:</code>	The number of items in <code>y_binning_limits_array</code> . 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:

```
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:

```
typedef struct PicamModulationsConstraint
{
    PicamConstraintScope    scope;
    PicamConstraintSeverity severity;
    pibln                   empty_set;
    piint                   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:

```
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;
    pibln    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.

*continued on next page*

---

*continued from previous page*

`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()`
  - `Picam_SetParameterPulseValueOnline()`
- 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()`

*continued on next page*



*continued from previous page*

- **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:

```
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`, or
- `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:

```
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:

```
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()*

### 5.4.1.6 *Picam\_CanSetParameterRoisValue()*

#### Description

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

#### Syntax

The syntax for `Picam_CanSetParameterRoisValue()` is:

```
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:

```
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()`



### 5.4.1.8 *Picam\_DestroyPulses()*

#### Description

`Picam_DestroyPulses()` releases memory that has been allocated by PICam for use by `pulses`.

If `pulses` is null, calling `Picam_DestroyPulses()` has no effect.

#### Syntax

The syntax for `Picam_DestroyPulses()` is:

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

#### Input Parameters

Input parameters for `Picam_DestroyPulses()` are:

`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:

```
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()`

### 5.4.1.10 *Picam\_CanSetParameterPulseValue()*

#### Description

`Picam_CanSetParameterPulseValue()` determines if a value is valid for a specified Pulse parameter.

#### Syntax

The syntax for `Picam_CanSetParameterPulseValue()` is:

```
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 the:

- `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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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.

Valid parameters are those of type `PicamValueType_Pulse`.

`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:

```
PICAM_API Picam_CanSetParameterIntegerValue(  
    PicamHandle camera_or_accessory,  
    PicamParameter parameter,  
    piint 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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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 `PicamValueType_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:

```
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`, or
- `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:

```
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:

```
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:

```
PICAM_API Picam_ReadParameterIntegerValue(
                                PicamHandle camera_or_accessory,
                                PicamParameter parameter,
                                piint* 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.

*continued on next page*

*continued from previous page*

### **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.

*continued on next page*

---

*continued from previous page*

### **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:

```
PICAM_API Picam_CanWaitForStatusParameter(  
    PicamHandlecamera_or_accessory,  
    PicamParameterparameter,  
    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:

```
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:

```
PICAM_API Picam_EstimateTimeToStatusParameterValue(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    piint value,
    piint* 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,  
    piint value,  
    piint 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:

```
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 PICam 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:

```
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:

```
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`, or
- `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.

#### Output Parameters

Output parameters for `Picam_GetParameterValueType()` are:

`type`: Pointer to the memory location in which the data type of the specified parameter's 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`.

#### 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:

```
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:

```
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:

```
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:

```
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 Rois 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 Rois 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:

```
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:

```
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:

```
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:

```
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.

##### 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.

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

##### 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:

```
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:

```
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:

```
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:

```
PICAM_API Picam_CommitParameters(
    PicamHandle camera_or_accessory,
    const PicamParameter** failed_parameter_array,
    piint* 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_ExactReadoutCountMaximum`.
  - 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

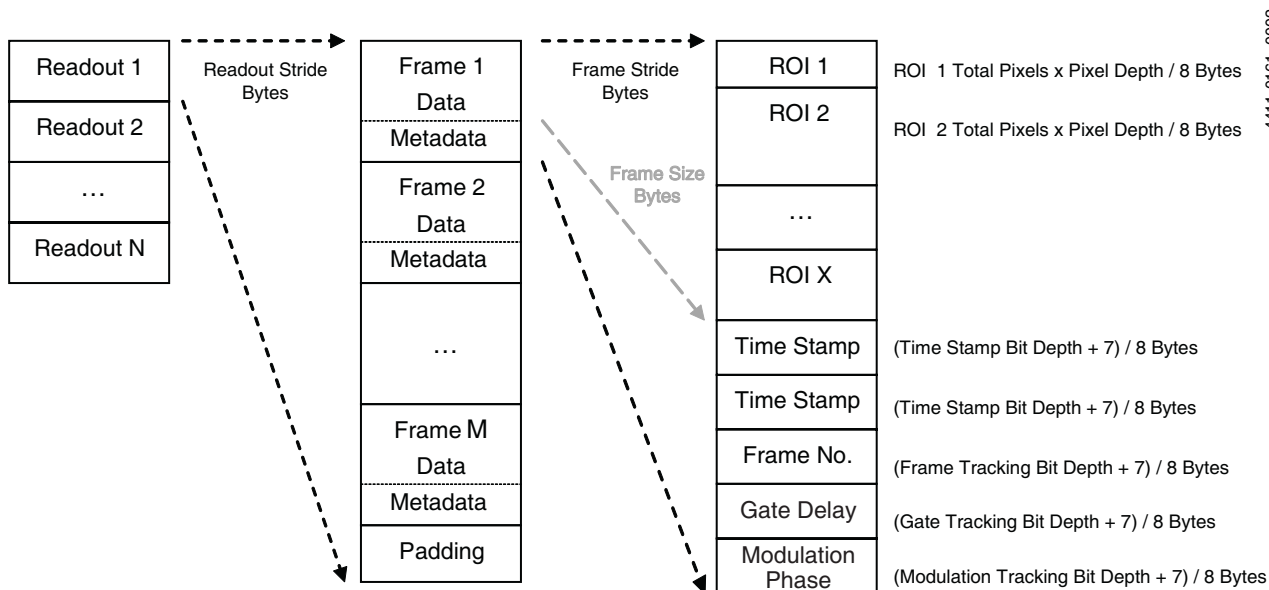
Acquired data is structured as shown in [Figure 6-1](#).



### 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

Enumerator	Description
<code>PicamAcquisitionErrorsMask_CameraFaulted</code>	The hardware has critically malfunctioned and is in need of service. Further acquisitions are not possible until the hardware has been serviced.
<code>PicamAcquisitionErrorsMask_ConnectionLost</code>	The hardware was disconnected.
<code>PicamAcquisitionErrorsMask_DataLost</code>	Data has been lost.
<code>PicamAcquisitionErrorsMask_DataNotArriving</code>	Data is no longer arriving from the hardware.
<code>PicamAcquisitionErrorsMask_None</code>	No errors have occurred.
<code>PicamAcquisitionErrorsMask_ShutterOverheated</code>	A connected shutter has overheated and is temporarily disabled. Further acquisitions are not possible until the shutter is no longer overheated.

## 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:

```
typedef struct PicamAvailableData
{
    void*   initial_readout;
    pi64s   readout_count;
} PicamAvailableData;
```

#### Variable Definitions

The variables required by `PicamAvailableData` are:

<code>initial_readout</code> :	Pointer to the start of the first available readout.
<code>readout_count</code> :	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:

```
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 183 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*

*continued from previous 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...`PicamParameter_ExactReadoutCountMaximum`]  
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 183 for information.

Data acquisition 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 190 for additional information); or
- `Picam_StopAcquisition()` is called.



#### 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.

### 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()` halts an in-progress data acquisition.



#### 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.

### Syntax

The syntax for `Picam_IsAcquisitionRunning()` is:

```
PICAM_API Picam_IsAcquisitionRunning(  
                                PicamHandle camera,  
                                pibln* 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 a 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 `PicamAcquisitionStatus.errors` 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:

```
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.  
When specifying an infinite length of time, configure this parameter to **-1**.

*continued on next page*

*continued from previous page*

## 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.

## 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.`

# 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.

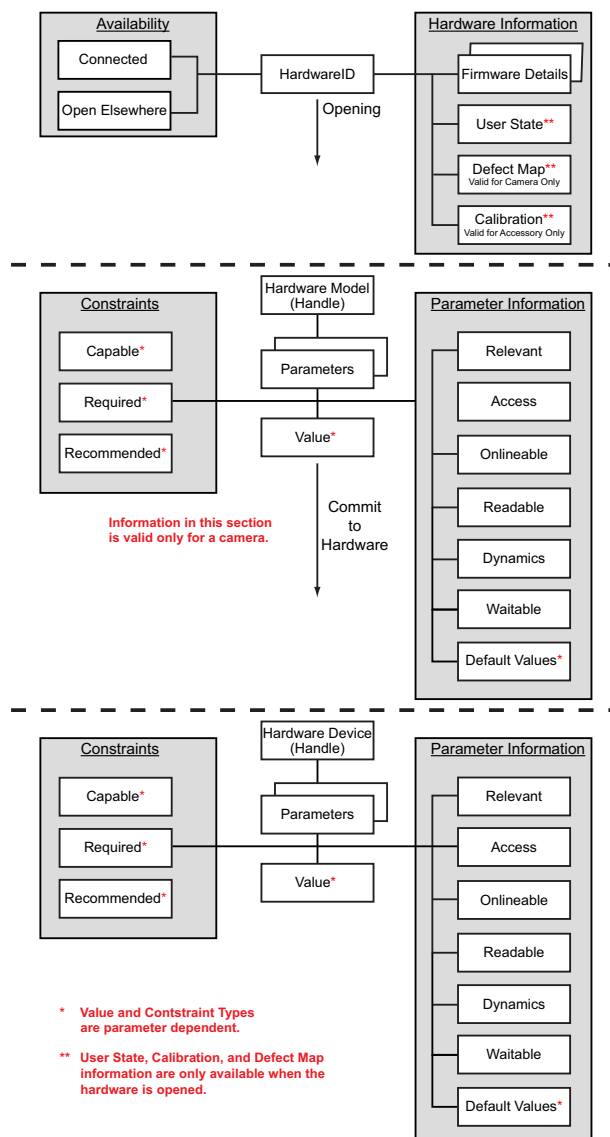
Figure 7-1 illustrates a block diagram of the PICam Advanced Function structure.



## REFERENCES:

Refer to [Section 2.6.1, Handles](#), for information about the handles used within PICam.

Figure 7-1: PICam Structure - Advanced



4411-0161\_0003

## 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

Enumerator	Description
<a href="#">PicamDiscoveryAction_Faulted</a>	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.
<a href="#">PicamDiscoveryAction_Found</a>	Hardware is now available for use.
<a href="#">PicamDiscoveryAction_Lost</a>	Hardware is no longer available for use.

## 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

Enumerator	Description
<code>PicamHandleType_Accessory</code>	This handle refers to a hardware accessory.
<code>PicamHandleType_CameraDevice</code>	The handle refers to a camera device.
<code>PicamHandleType_CameraModel</code>	The handle refers to a camera model.
<code>PicamHandleType_EMCalibration</code>	The handle refers to a camera opened for EM calibration.

## 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

Enumerator	Description
<i>PicamDynamicsMask_None</i>	No parameter attributes may change.
<i>PicamDynamicsMask_Value</i>	The parameter value may change.
<i>PicamDynamicsMask_ValueAccess</i>	The parameter value access may change.
<i>PicamDynamicsMask_IsRelevant</i>	The parameter relevance may change.
<i>PicamDynamicsMask_Constraint</i>	The parameter dependent constraints may change.



## 7.1.4 Camera-Specific Data Acquisition Enumerations

This section provides programming information about camera-specific data acquisition enumerations.

### 7.1.4.1 *PicamAcquisitionState*

#### Data Type

`PicamAcquisitionState` is defined as `enum`.

#### Description

`PicamAcquisitionState` is the set of camera states that can be detected during an acquisition.

#### Enumerator Definitions

Refer to [Table 7-4](#) for enumerator definitions.

**Table 7-4:** `PicamAcquisitionState` Enumerator Definitions

Enumerator	Description
<code>PicamAcquisitionState_ReadoutStarted</code>	The camera has begun to readout data.
<code>PicamAcquisitionState_ReadoutEnded</code>	The camera has finished reading out data.

### 7.1.4.2 *PicamAcquisitionStateErrorsMask*

#### Data Type

`PicamAcquisitionStateErrorsMask` is defined as `enum`.

#### Description

`PicamAcquisitionStateErrorsMask` is the set of errors that can occur while detecting acquisition states.

#### Enumerator Definitions

Refer to [Table 7-5](#) for enumerator definitions.

**Table 7-5:** `PicamAcquisitionStateErrorsMask` Enumerator Definitions

Enumerator	Description
<code>PicamAcquisitionStateErrorsMask_None</code>	No error has occurred.
<code>PicamAcquisitionStateErrorsMask_LostCount</code>	One or more state transitions have been missed.

## 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:

```
typedef struct PicamPixelLocation
{
    pi16s x;
    pi16s 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:

```
typedef struct PicamColumnDefect
{
    PicamPixelLocation start;
    piint 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:

```
typedef struct PicamRowDefect
{
    PicamPixelLocation start;
    piint 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:

```
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:

<code>is_valid:</code>	Indicates the validation status for a single parameter. Valid values are: <ul style="list-style-type: none"> <li>• <code>TRUE</code> Indicates the parameter validation has succeeded.</li> <li>• <code>FALSE</code> Indicates the parameter validation has failed.</li> </ul>
<code>failed_parameter:</code>	The parameter that has failed validation. This is null when validation has succeeded.
<code>failed_error_constraint_scope:</code>	The scope of the error constraint that has failed. This is null when: <ul style="list-style-type: none"> <li>• Validation has succeeded, or</li> <li>• Only a warning constraint has failed validation.</li> </ul>
<code>failed_warning_constraint_scope:</code>	The scope of the warning constraint that has failed. This is null when: <ul style="list-style-type: none"> <li>• Validation has succeeded, or</li> <li>• Only an error constraint has failed validation.</li> </ul>
<code>error_constraining_parameter_array:</code>	An array of parameters involved in constraining the failed parameter when a dependent error failed. This is null otherwise.

*continued on next page*

*continued from previous page*

<code>error_constraining_parameter_count:</code>	The number of items in the array of parameters involved in constraining the failed parameter if a dependent error failed. This is 0 otherwise.
<code>warning_constraining_parameter_array:</code>	An array of parameters involved in constraining the failed parameter if a dependent warning failed. This is null otherwise.
<code>warning_constraining_parameter_count:</code>	The number of items in the array of parameters involved in constraining the failed parameter if a dependent warning failed. This is 0 otherwise.

### 7.2.2.2 *PicamValidationResults*

#### Description

`PicamValidationResults` provides information about the validation status for multiple parameters.

#### Structure Definition

The structure definition for `PicamValidationResults` is:

```
typedef struct PicamValidationResults
{
    pibln is_valid;
    const PicamValidationResult* validation_result_array;
    piint validation_result_count;
} PicamValidationResults;
```

#### Variable Definitions

The variables required by `PicamValidationResults` are:

<code>is_valid:</code>	Indicates the validation status for multiple tested parameters. Valid values are: <ul style="list-style-type: none"> <li>• TRUE Indicates all parameter validations have succeeded.</li> <li>• FALSE Indicates one or more parameter validations has failed.</li> </ul>
<code>validation_result_array:</code>	An array containing a result for each parameter that failed validation; null if validation succeeded.
<code>validation_result_count:</code>	The number of failed parameter results; 0 if validation succeeded.

### 7.2.2.3 *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:

```
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:

<code>failed_parameter:</code>	The parameter whose validation failed and is constrained by another.
<code>failed_error_constraint_scope:</code>	The scope of the error constraint that failed. This is null when only a warning constraint failed.
<code>failed_warning_constraint_scope:</code>	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:

```
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:

```
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:

```
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()`
- Shared Camera/Accessory Parameter Value Callbacks
  - `PicamIntegerValueChangedCallback()`
  - `PicamFloatingPointValueChangedCallback()`
  - `PicamWhenStatusParameterValueCallback()`
  - `PicamIsRelevantChangedCallback()`
  - `PicamValueAccessChangedCallback()`
- Camera-Specific Parameter Constraints Callbacks
  - `PicamDependentRoisConstraintChangedCallback()`
  - `PicamDependentPulseConstraintChangedCallback()`
  - `PicamDependentModulationsConstraintChangedCallback()`
- 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:

```
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:

```
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:

```
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:

```
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.

### 7.3.3.3 *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:

```
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.

### 7.3.3.4 *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:

```
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:

```
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:

```
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:

```
typedef PicamError (PIL_CALL* PicamWhenStatusParameterValueCallback)
(
    PicamHandle device_or_accessory,
    PicamParameter parameter,
    pint 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:

```
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:

```
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:

```
typedef PicamError (PIL_CALL* PicamDependentRoisConstraintChanged  
    Callback)  
(  
    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:

```
typedef PicamError (PIL_CALL* PicamDependentPulseConstraintChanged
    Callback)
(
    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:

```
typedef PicamError (PIL_CALL* PicamDependentModulationsConstraint
    ChangedCallback)
(
    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:

```
typedef PicamError (PIL_CALL*PicamDependentCollectionConstraint  
    ChangedCallback)  
(  
    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:

```
typedef PicamError (PIL_CALL*PicamDependentRangeConstraint  
    ChangedCallback)  
(  
    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:

```
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:

```
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 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*

*continued from previous 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:

```
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:

```
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:

```
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:

```
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:

`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:

```
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:

```
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:

```
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();` or
- `PicamAdvanced_CloseCameraDevice().`

#### Syntax

The syntax for `PicamAdvanced_OpenCameraDevice()` is:

```
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:

```
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,  
    piint* 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.

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

*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 *PicamAdvanced\_GetCameraDevice()*

##### Description

`PicamAdvanced_GetCameraDevice()` returns the handle for a specified camera device.

##### Syntax

The syntax for `PicamAdvanced_GetCameraDevice()` is:

```
PICAM_API PicamAdvanced_GetCameraDevice(  
    PicamHandle camera,  
    PicamHandle* device );
```

##### Input Parameters

Input parameters for `PicamAdvanced_GetCameraDevice()` are:

`camera`: Specifies the camera device or camera model for which the handle is to be returned.

##### Output Parameters

Output parameters for `PicamAdvanced_GetCameraDevice()` are:

`device`: Pointer to the memory location in which the handle for the camera device is stored.

## 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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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.

*continued on next page*



---

*continued from previous page*

### **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:

```
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:

```
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 change 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:

```
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:

```
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:

```
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:

```
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:

```
PICAM_API PicamAdvanced_NotifyWhenStatusParameterValue(
    PicamHandle device_or_accessory,
    PicamParameter parameter,
    piint 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:

```
PICAM_API PicamAdvanced_CancelNotifyWhenStatusParameterValue(  
    PicamHandle  device_or_accessory,  
    PicamParameter  parameter,  
    piint  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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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 no 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:

```
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:

```
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 not longer required.

#### Syntax

The syntax for

`PicamAdvanced_RegisterForDependentModulationsConstraintChanged()` is:

```
PICAM_API PicamAdvanced_RegisterForDependentModulationsConstraint
Changed (
                                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:

```
PICAM_API PicamAdvanced_UnregisterForDependentModulations
          ConstraintChanged(
                                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:

```
PICAM_API PicamAdvanced_GetParameterCollectionConstraints(
    PicamHandle camera_or_accessory,
    PicamParameter parameter,
    const PicamCollectionConstraint** constraint_array,
    piint* 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 *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 not 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:

```
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:

```
PICAM_API PicamAdvanced_GetParameterRangeConstraints(  
    PicamHandle camera_or_accessory,  
    PicamParameter parameter,  
    const PicamRangeConstraint** constraint_array,  
    piint* 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 no longer required.

#### Syntax

The syntax for `PicamAdvanced_RegisterForDependentRangeConstraintChanged()` is:

```
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:

```
PICAM_API PicamAdvanced_UnregisterForDependentRangeConstraintChanged(  
    PicamHandle camera_or_accessory,  
    PicamParameter parameter,  
    PicamDependentRangeConstraintChangedCallback 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:

```
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:

```
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 not 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:

```
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\_ValidateParameters()*

#### Description

`PicamAdvanced_ValidateParameters()` validates all parameters against all associated constraints and returns the results.

#### Syntax

The syntax for `PicamAdvanced_ValidateParameters()` is:

```
PICAM_API PicamAdvanced_ValidateParameters(  
                                PicamHandle model,  
                                const PicamValidationResults** results );
```

#### Input Parameters

Input parameters for `PicamAdvanced_ValidateParameters()` are:

`model`: Handle for the model for which all parameters are being validated.

#### Output Parameters

Output parameters for `PicamAdvanced_ValidateParameters()` 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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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:

```
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 advance 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:

```
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 recalibrate 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:

```
typedef struct PicamEMCalibrationDate
{
    piint  year;
    piint  month;
    piint  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:

```
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:

```
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,  
    piint* 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.  
**NOTE:** This memory is allocated by PICam and must be released by calling `Picam_DestroyHandles()`
- `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.

#### 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:

```
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 setpoint 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 setpoint is to be determined.

##### Output Parameters

Output parameters for `PicamEMCalibration_GetSensorTemperatureSetPoint()` are:

`value`: Pointer to the memory location in which the setpoint 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 setpoint 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 setpoint is to be programmed.

`value`: The desired temperature setpoint, 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 setpoints, in degrees Celsius, for a specified camera is stored.

#### Syntax

The syntax for *PicamEMCalibration\_GetSensorTemperatureSetPointConstraint()* is:

```
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 setpoints 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 setpoints is stored.

**NOTE:** This memory is allocated by PICam and must be released by calling *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 cancelled (via the use of the callback function `PicamEMCalibrationCallback()`) this function returns `PicamError_OperationCanceled`.

#### Syntax

The syntax for `PicamEMCalibration_Calibrate()` is:

```
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:

- 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`.

- An asterisk indicates that the parameter does not apply to all members of a camera family.

**Table A-1: Symbol Key for Table A-2 and Table A-3**

Value Types		Constraint Types	
F = Floating Point	M = Modulations	R = Range	M = Modulation
E = Enumeration	R = Region of Interest	C = Collection	P = Pulse
B = Boolean	P = Pulse	Ri = Region of Interest	
I = Integer	L = Large Integer		

## A.1 Camera Parameter Information

Refer to [Table A-2](#) for the list of available Camera parameters

**Table A-2: Parameter Information and Camera Support (Sheet 1 of 8)**

Parameter Name	Read Only	Value Type	Constraint Type	BLAZE	FERGIE	KURO	NIRvana LN	PI-MAX 3/4	PI-MTE	PIoNIR/NIRvana/ST	PIXIS	ProEM/ + / -HS	PyLoN	PyLoN-IR	SOPHIA	Quad-RO
<b>Shutter Timing</b>																
ActiveShutter		E	C	✓	✓										✓	
ExposureTime		F	R	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
ExternalShutterStatus	✓	E		✓											✓	
ExternalShutterType	✓	E		✓	✓										✓	
InactiveShutterTiming ModeResult	✓	E													✓*	
InternalShutterStatus	✓	E			✓										✓	

**Table A-2: Parameter Information and Camera Support (Sheet 2 of 8)**

Parameter Name	Read Only	Value Type	Constraint Type	BLAZE	FERGIE	KURO	NIRvana LN	PI-MAX 3/4	PI-MTE	PIoNIR/NIRvana/ST	PIXIS	ProEM/ + / -HS	PyLoN	PyLoN-IR	SOPHIA	Quad-RO
InternalShutterType	✓	E		✓	✓										✓	
ShutterClosingDelay		F	R	✓	✓		✓		✓	✓	✓	✓	✓	✓	✓	✓
ShutterDelay Resolution		F	C	✓	✓		✓		✓	✓	✓	✓	✓	✓	✓	✓
ShutterOpeningDelay		F	R	✓	✓		✓			✓		✓	✓	✓	✓	
ShutterTimingMode		E	C	✓	✓		✓		✓	✓	✓	✓	✓	✓	✓	✓
<b>Gating</b>																
DifEndingGate		P	P					✓*								
DifStartingGate		P	P					✓*								
GatingMode		E	C		✓			✓								
RepetitiveGate		P	P		✓			✓								
SequentialEndingGate		P	P		✓			✓								
SequentialGateStep Count		L	R		✓			✓								
SequentialGateStep Iterations		L	R		✓			✓								
SequentialStarting Gate		P	P		✓			✓								
<b>Intensifier</b>																
BracketGating		B	C					✓*								
CustomModulation Sequence		M	M					✓*								
EMICcdGain		I	R					✓*								
EMICcdGainControlMode		E	C					✓*								
EnableIntensifier		B	C					✓								
EnableModulation		B	C					✓*								
GatingSpeed	✓	E						✓								
IntensifierDiameter	✓	F						✓								
IntensifierGain		I	R					✓								
IntensifierOptions	✓	E						✓								
IntensifierStatus	✓	E						✓								
ModulationDuration		F	R					✓*								

**Table A-2: Parameter Information and Camera Support (Sheet 3 of 8)**

Parameter Name	Read Only	Value Type	Constraint Type	BLAZE	FERGIE	KURO	NIRvana LN	PI-MAX 3/4	PI-MTE	PIoNIR/NIRvana/ST	PIXIS	ProEM/ + / -HS	PyLoN	PyLoN-IR	SOPHIA	Quad-RO
ModulationFrequency		F	R					✓*								
PhosphorDecayDelay		F	R					✓								
PhosphorDecayDelay Resolution		F	C					✓								
PhosphorType	✓	E						✓								
Photocathode Sensitivity	✓	E						✓								
RepetitiveModulation Phase		F	R					✓*								
SequentialEnding ModulationPhase		F	R					✓*								
SequentialStarting ModulationPhase		F	R					✓*								
<b>Analog to Digital Conversion</b>																
AdcAnalogGain		E	C	✓	✓	✓		✓		✓	✓	✓	✓	✓	✓	✓
AdcBitDepth		I	C	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AdcEMGain		I	R					✓*				✓				
AdcQuality		E	C	✓				✓*	✓*		✓*	✓*	✓		✓*	
AdcSpeed		F	C	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CorrectPixelBias		B	C	✓	✓			✓*				✓	✓		✓	
<b>Hardware I/O</b>																
AnticipateTrigger		B	C					✓*								
AuxOutput		P	P		✓			✓								
DelayFromPreTrigger		F	R					✓*								
EnableAuxOutput		B	C		✓											
EnableModulation OutputSignal		B	C					✓*								
EnableSyncMaster		B	C					✓								
InvertOutputSignal		B	C	✓	✓		✓	✓		✓		✓	✓	✓	✓	
InvertOutputSignal2		B	C	✓	✓										✓	
ModulationOutput SignalAmplitude		F	R					✓*								



**Table A-2: Parameter Information and Camera Support (Sheet 5 of 8)**

[illegible]

**Table A-2: Parameter Information and Camera Support (Sheet 6 of 8)**

Parameter Name	Read Only	Value Type	Constraint Type	BLAZE	FERGIE	KURO	NIRvana LN	PI-MAX 3/4	PI-MTE	PIoNIR/NIRvana/ST	PIXIS	ProEM/ + / -HS	PyLoN	PyLoN-IR	SOPHIA	Quad-RO
PixelWidth	✓	F		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SensorActiveBottom Margin	✓	I		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SensorActiveExtended Height	✓	I		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SensorActiveHeight	✓	I		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SensorActiveLeft Margin	✓	I		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SensorActiveRight Margin	✓	I		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SensorActiveTopMargin	✓	I		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SensorActiveWidth	✓	I		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SensorMaskedBottom Margin	✓	I		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SensorMaskedHeight	✓	I		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SensorMaskedTopMargin	✓	I		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SensorSecondaryActive Height	✓	I		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SensorSecondaryMasked Height	✓	I		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SensorType	✓	E		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
<b>Sensor Layout</b>																
ActiveBottomMargin		I	R	✓	✓			✓	✓		✓	✓	✓		✓	
ActiveExtendedHeight					✓*										✓	
ActiveHeight		I	R	✓	✓			✓	✓		✓	✓	✓		✓	
ActiveLeftMargin		I	R	✓	✓			✓	✓		✓	✓	✓		✓	
ActiveRightMargin		I	R	✓	✓			✓	✓		✓	✓	✓		✓	
ActiveTopMargin		I	R	✓	✓			✓	✓		✓	✓	✓		✓	
ActiveWidth		I	R	✓	✓			✓	✓		✓	✓	✓		✓	
MaskedBottomMargin		I	R		✓*			✓*	✓*			✓*				
MaskedHeight		I	R		✓*			✓*	✓*			✓*				
MaskedTopMargin		I	R		✓*			✓*	✓*			✓*				
SecondaryActiveHeight		I	R									✓*				



**Table A-2: Parameter Information and Camera Support (Sheet 7 of 8)**

**Table A-2: Parameter Information and Camera Support (Sheet 8 of 8)**

Parameter Name	Read Only	Value Type	Constraint Type	BLAZE	FERGIE	KURO	NIRvana LN	PI-MAX 3/4	PI-MTE	PIoNIR/NIRvana/ST	PIXIS	ProEM/ + / -HS	PyLoN	PyLoN-IR	SOPHIA	Quad-RO
GratingGrooveDensity	✓	F			✓											
GratingType	✓	E			✓											
InclusionAngle	✓	F			✓											
SensorAngle	✓	F			✓											

## 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**

Parameter Name	Read Only	Value Type	Constraint Type	FERGIE AEL	FERGIE QTH	FERGIE Laser 785
<b>Accessory – Lamp</b>						
Age	✓	F			✓	
LifeExpectancy	✓	F			✓	
LightSource		E	C	✓	✓	
LightSourceStatus	✓	E			✓	
<b>Accessory – Laser</b>						
InputTriggerStatus	✓	E				✓
LaserOutputMode		E	C			✓
LaserPower		F	R			✓
LaserStatus	✓	E				✓

# 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.

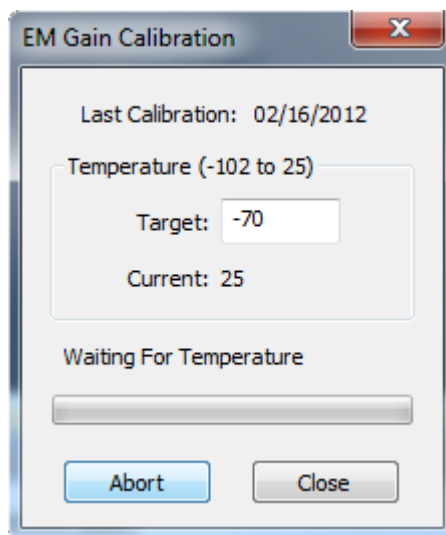
4. Launch `EMGainCalibration.exe`.

**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**



4411-0161\_0004

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:

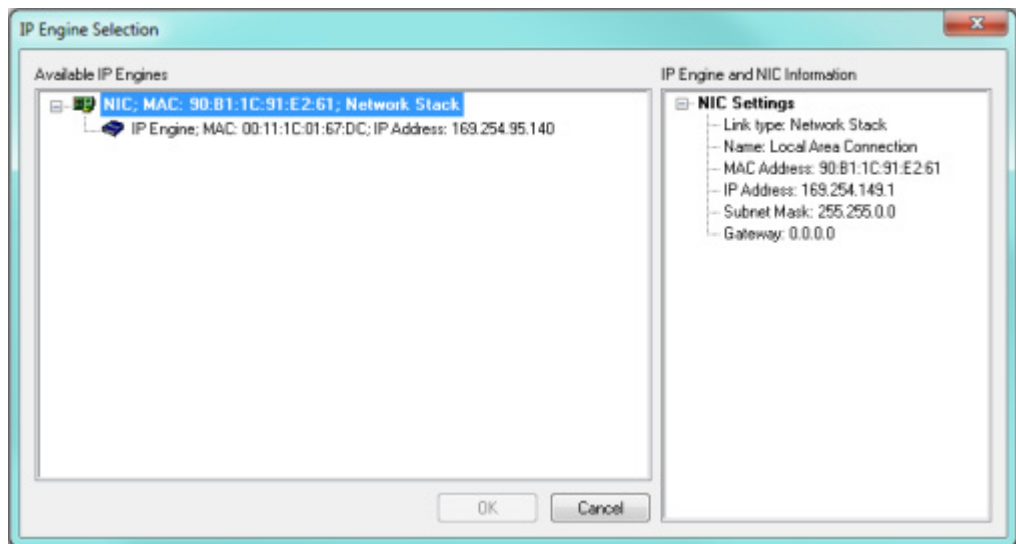
1. On the Host Computer, navigate to the following directory:

`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**



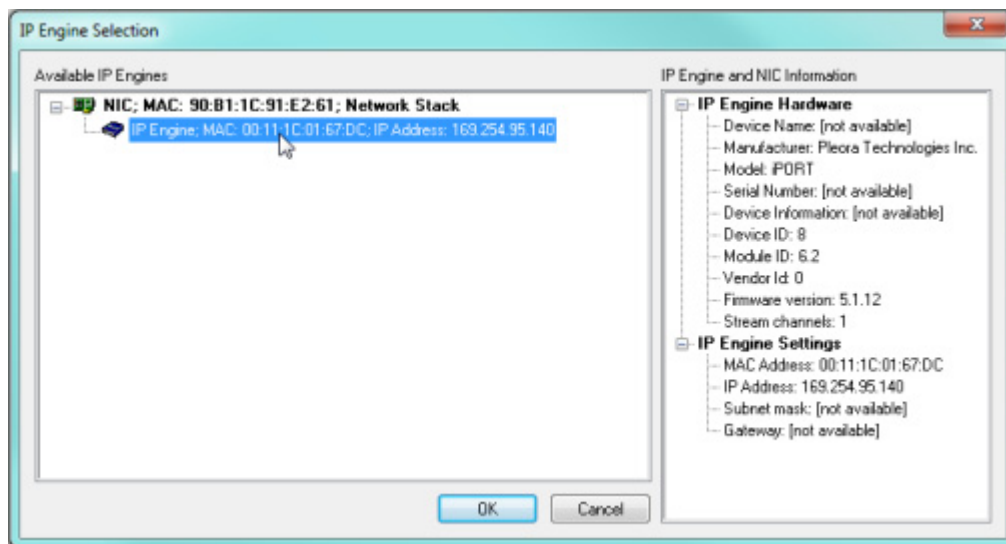
3. Within the **Available IP Engines** field, select the desired IP Engine from the list of available devices.

**NOTE:**


Each IP Engine listed represents one camera.

See [Figure C-2](#).

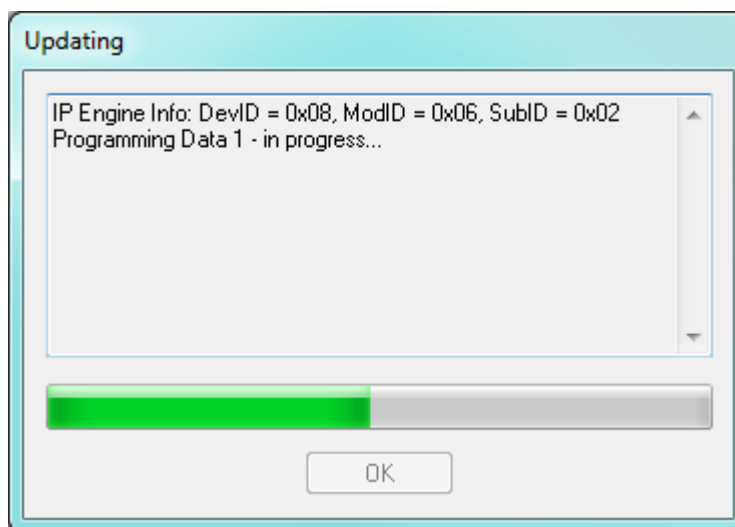
**Figure C-2: Firmware Upgrade: Selecting Device to be Upgraded**



4411-0161\_0007

4. Once selected, click  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**

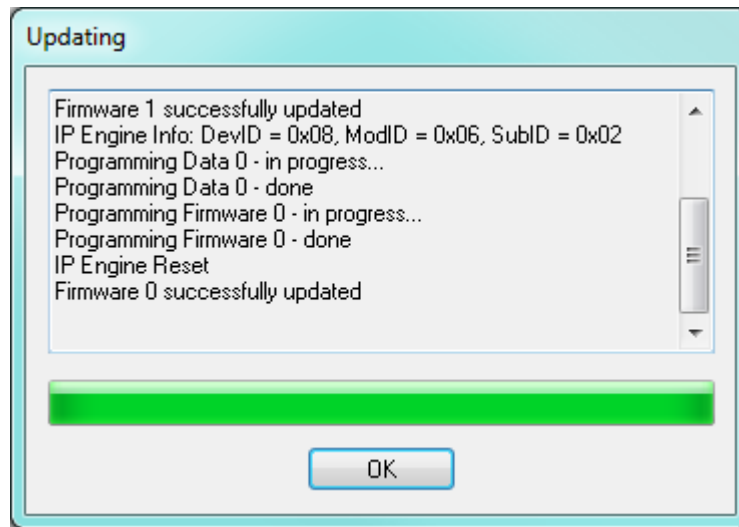


4411-0161\_0008

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**



4411-0161\_0009

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 unistalling PICam 5.x.

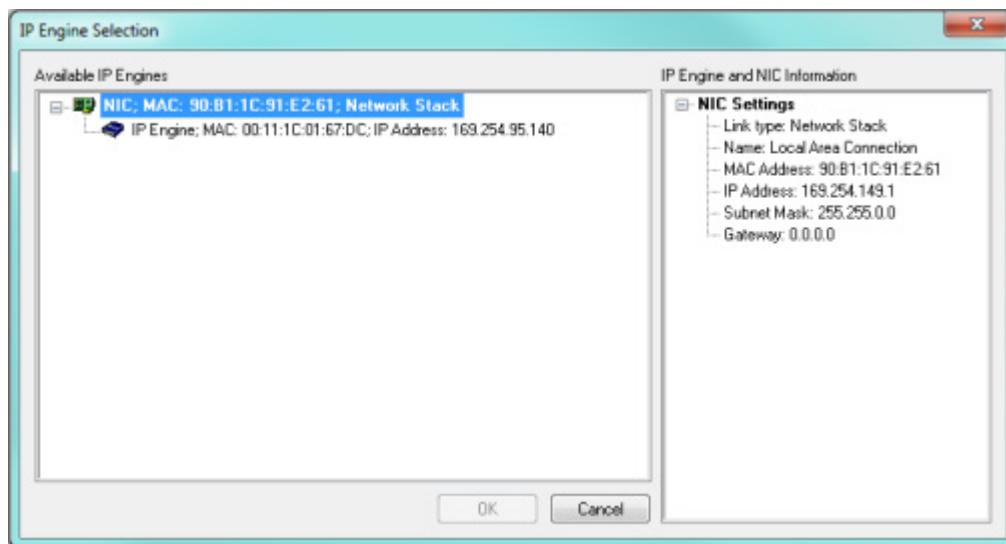
Additionally, Pleora EBUS Support is required in order to run the Firmware Restore Tool. Do not uninstall Pleora EBUS Support.

## 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**



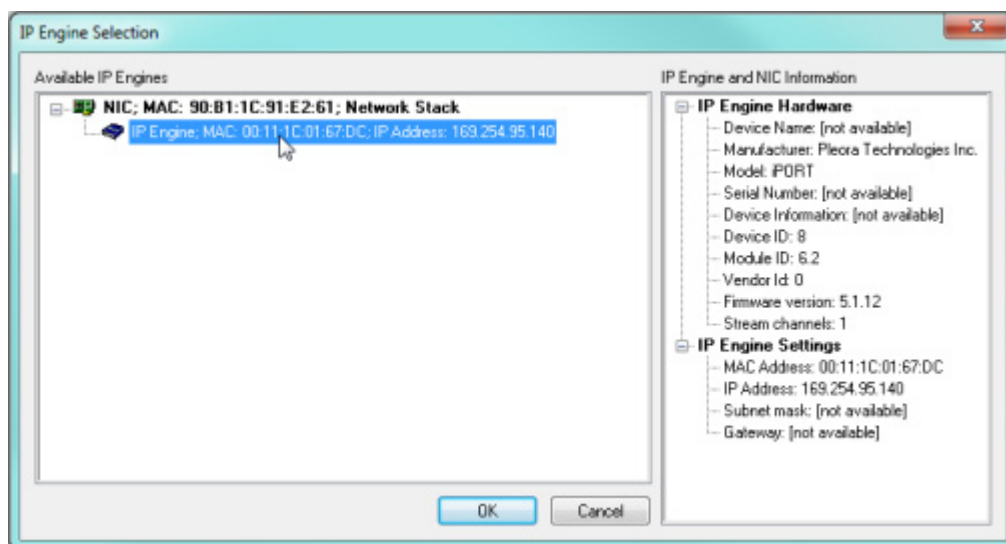
3. Within the **Available IP Engines** field, select the desired IP Engine from the list of available devices. See [Figure C-6](#).




### NOTE:

Each IP Engine listed represents one camera.

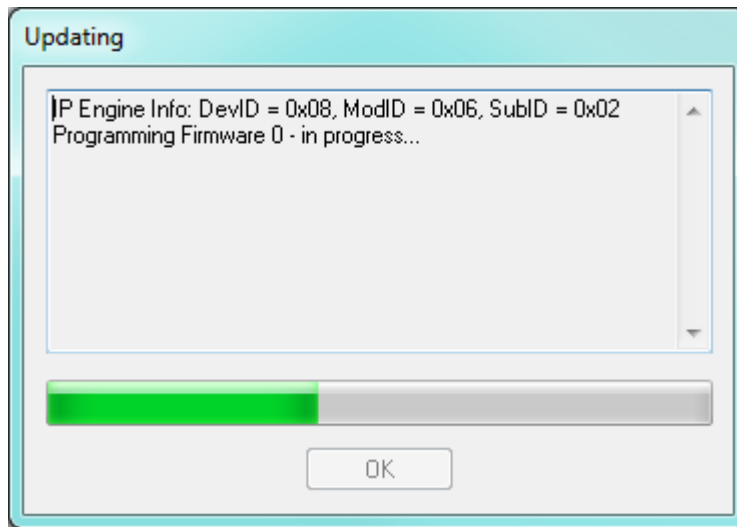
**Figure C-6: Firmware Restore: Selecting Device to be Restored**





4. Once selected, click  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**

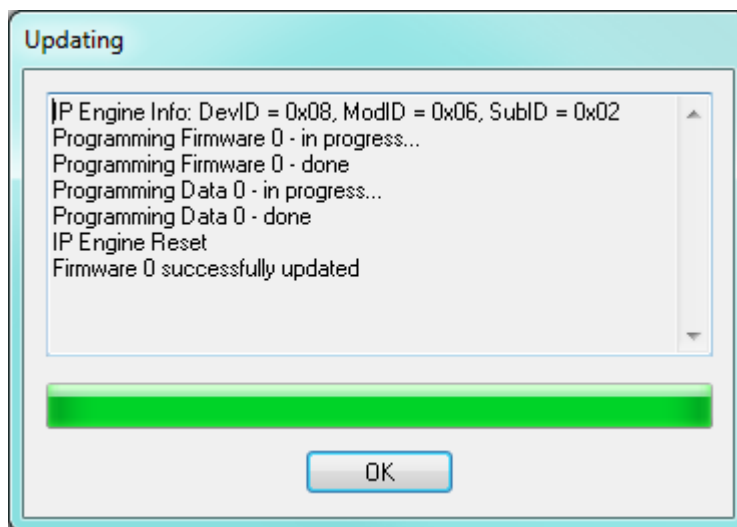


4411-0161\_0010

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**



4411-0161\_0011

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**

In PICam 4.x and Older Code, if you:		... in PICam 5.0, you need to:	
Action	Parameter	Action	Parameter
Get	<code>PicamParameter_ActiveLeftMargin</code>	Get	<code>PicamParameter_ActiveRightMargin</code>
Set	<code>PicamParameter_ActiveLeftMargin</code>	Set	<code>PicamParameter_ActiveRightMargin</code>
Get	<code>PicamParameter_ActiveRightMargin</code>	Get	<code>PicamParameter_ActiveLeftMargin</code>
Set	<code>PicamParameter_ActiveRightMargin</code>	Set	<code>PicamParameter_ActiveLeftMargin</code>

## 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**

In PICam 4.x and Older Code, if you:		... in PICam 5.0, you need to:	
Action	Parameter	Action	Parameter
Get	<code>PicamParameter_ActiveLeftMargin</code>	Get	<code>PicamParameter_ActiveRightMargin</code>
Set	<code>PicamParameter_ActiveLeftMargin</code>	Set	<code>PicamParameter_ActiveRightMargin</code>
Get	<code>PicamParameter_ActiveRightMargin</code>	Get	<code>PicamParameter_ActiveLeftMargin</code>
Set	<code>PicamParameter_ActiveRightMargin</code>	Set	<code>PicamParameter_ActiveLeftMargin</code>

## 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**

In PICam 4.x and Older Code, if you:		... in PICam 5.0, you need to:	
Action	Parameter	Action	Parameter
<b>ADC Quality: Electron Multiplied</b>			
Get	<code>PicamParameter_ActiveLeftMargin</code>	Get	<code>PicamParameter_ActiveRightMargin</code>
Set	<code>PicamParameter_ActiveLeftMargin</code>	Set	<code>PicamParameter_ActiveRightMargin</code>
Get	<code>PicamParameter_ActiveRightMargin</code>	Get Subtract	<code>PicamParameter_ActiveLeftMargin</code> <b>16</b> from the value
Set	<code>PicamParameter_ActiveRightMargin</code>	Add Set	<b>16</b> to the value <code>PicamParameter_ActiveLeftMargin</code>
Get	<code>PicamParameter_SensorActiveLeftMargin</code>	Get	<code>PicamParameter_SensorActiveRightMargin</code>
Get	<code>PicamParameter_SensorActiveRightMargin</code>	Get Subtract	<code>PicamParameter_SensorActiveLeftMargin</code> <b>16</b> from the value
<b>ADC Quality: Low Noise</b>			
Get	<code>PicamParameter_ActiveRightMargin</code>	Get Subtract	<code>PicamParameter_ActiveRightMargin</code> <b>16</b> from the value
Set	<code>PicamParameter_ActiveRightMargin</code>	Add Set	<b>16</b> to the value <code>PicamParameter_ActiveRightMargin</code>
Get	<code>PicamParameter_SensorActiveRightMargin</code>	Get Subtract	<code>PicamParameter_SensorActiveRightMargin</code> <b>16</b> from the value

## 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**

In PICam 4.x and Older Code, if you:		... in PICam 5.0, you need to:	
Action	Parameter	Action	Parameter
<b>ADC Quality: Electron Multiplied</b>			
Get	<code>PicamParameter_ActiveRightMargin</code>	Get Subtract	<code>PicamParameter_ActiveRightMargin</code> <b>16</b> from the value
Set	<code>PicamParameter_ActiveRightMargin</code>	Add Set	<b>16</b> to the value <code>PicamParameter_ActiveRightMargin</code>
Get	<code>PicamParameter_SensorActiveRightMargin</code>	Get Subtract	<code>PicamParameter_SensorActiveRightMargin</code> <b>16</b> from the value
<b>ADC Quality: Low Noise</b>			
Get	<code>PicamParameter_ActiveLeftMargin</code>	Get	<code>PicamParameter_ActiveRightMargin</code>
Set	<code>PicamParameter_ActiveLeftMargin</code>	Set	<code>PicamParameter_ActiveRightMargin</code>
Get	<code>PicamParameter_ActiveRightMargin</code>	Get Subtract	<code>PicamParameter_ActiveLeftMargin</code> <b>16</b> from the value
Set	<code>PicamParameter_ActiveRightMargin</code>	Add Set	<b>16</b> to the value <code>PicamParameter_ActiveLeftMargin</code>
Get	<code>PicamParameter_SensorActiveLeftMargin</code>	Get	<code>PicamParameter_SensorActiveRightMargin</code>
Get	<code>PicamParameter_SensorActiveRightMargin</code>	Get Subtract	<code>PicamParameter_SensorActiveLeftMargin</code> <b>16</b> from the value



## 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**

In PICam 4.x and Older Code, if you:		... in PICam 5.0, you need to:	
Action	Parameter	Action	Parameter
<b>ADC Quality: High Capacity</b>			
Get	<a href="#">PicamParameter_ActiveLeftMargin</a>	Get	<a href="#">PicamParameter_ActiveRightMargin</a>
Set	<a href="#">PicamParameter_ActiveLeftMargin</a>	Set	<a href="#">PicamParameter_ActiveRightMargin</a>
Get	<a href="#">PicamParameter_ActiveRightMargin</a>	Get	<a href="#">PicamParameter_ActiveLeftMargin</a>
Set	<a href="#">PicamParameter_ActiveRightMargin</a>	Set	<a href="#">PicamParameter_ActiveLeftMargin</a>

## 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**

In PICam 4.x and Older Code, if you:		... in PICam 5.0, you need to:	
Action	Parameter	Action	Parameter
<b>ADC Quality: Low Noise</b>			
Get	<code>PicamParameter_ActiveLeftMargin</code>	Get	<code>PicamParameter_ActiveRightMargin</code>
Set	<code>PicamParameter_ActiveLeftMargin</code>	Set	<code>PicamParameter_ActiveRightMargin</code>
Get	<code>PicamParameter_ActiveRightMargin</code>	Get	<code>PicamParameter_ActiveLeftMargin</code>
Set	<code>PicamParameter_ActiveRightMargin</code>	Set	<code>PicamParameter_ActiveLeftMargin</code>

## 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 E-7: PICam 4.x (and Older) Code Changes: PIXIS: 100B/100BR/400B/400BR/1300B/1300BR, and XO/XF/XB/eXcelon Variants**

In PICam 4.x and Older Code, if you:		... in PICam 5.0, you need to:	
Action	Parameter	Action	Parameter
<b>ADC Quality: High Capacity</b>			
Get	<code>PicamParameter_ActiveLeftMargin</code>	Get	<code>PicamParameter_ActiveRightMargin</code>
Set	<code>PicamParameter_ActiveLeftMargin</code>	Set	<code>PicamParameter_ActiveRightMargin</code>
Get	<code>PicamParameter_ActiveRightMargin</code>	Get	<code>PicamParameter_ActiveLeftMargin</code>
Set	<code>PicamParameter_ActiveRightMargin</code>	Set	<code>PicamParameter_ActiveLeftMargin</code>

## 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**

In PICam 4.x and Older Code, if you:		... in PICam 5.0, you need to:	
Action	Parameter	Action	Parameter
<b>ADC Quality: Low Noise</b>			
Get	<code>PicamParameter_ActiveLeftMargin</code>	Get	<code>PicamParameter_ActiveRightMargin</code>
Set	<code>PicamParameter_ActiveLeftMargin</code>	Set	<code>PicamParameter_ActiveRightMargin</code>
Get	<code>PicamParameter_ActiveRightMargin</code>	Get	<code>PicamParameter_ActiveLeftMargin</code>
Set	<code>PicamParameter_ActiveRightMargin</code>	Set	<code>PicamParameter_ActiveLeftMargin</code>

## 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**

In PICam 4.x and Older Code, if you:		... in PICam 5.0, you need to:	
Action	Parameter	Action	Parameter
Get	<code>PicamParameter_ActiveLeftMargin</code>	Get	<code>PicamParameter_ActiveRightMargin</code>
Set	<code>PicamParameter_ActiveLeftMargin</code>	Set	<code>PicamParameter_ActiveRightMargin</code>
Get	<code>PicamParameter_ActiveRightMargin</code>	Get	<code>PicamParameter_ActiveLeftMargin</code>
Set	<code>PicamParameter_ActiveRightMargin</code>	Set	<code>PicamParameter_ActiveLeftMargin</code>
Get	<code>PicamParameter_SensorActiveLeftMargin</code>	Get	<code>PicamParameter_SensorActiveRightMargin</code>
Get	<code>PicamParameter_SensorActiveRightMargin</code>	Get	<code>PicamParameter_SensorActiveLeftMargin</code>

## 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)**

In PICam 4.x and Older Code, if you:		... in PICam 5.0, you need to:	
Action	Parameter	Action	Parameter
<b>ADC Quality: Electron Multiplied</b>			
Get	<code>PicamParameter_ActiveLeftMargin</code>	Get	<code>PicamParameter_ActiveRightMargin</code>
Set	<code>PicamParameter_ActiveLeftMargin</code>	Set	<code>PicamParameter_ActiveRightMargin</code>
Get	<code>PicamParameter_ActiveRightMargin</code>	Get Subtract	<code>PicamParameter_ActiveLeftMargin</code> <b>16</b> from the value
Set	<code>PicamParameter_ActiveRightMargin</code>	Add Set	<b>16</b> to the value <code>PicamParameter_ActiveLeftMargin</code>
Get	<code>PicamParameter_SensorActiveLeftMargin</code>	Get	<code>PicamParameter_SensorActiveRightMargin</code>
Get	<code>PicamParameter_SensorActiveRightMargin</code>	Get Subtract	<code>PicamParameter_SensorActiveLeftMargin</code> <b>16</b> from the value
<b>ADC Quality: Low Noise</b>			
Get	<code>PicamParameter_ActiveRightMargin</code>	Get Subtract	<code>PicamParameter_ActiveRightMargin</code> <b>16</b> from the value
Set	<code>PicamParameter_ActiveRightMargin</code>	Add Set	<b>16</b> to the value <code>PicamParameter_ActiveRightMargin</code>
Get	<code>PicamParameter_SensorActiveRightMargin</code>	Get Subtract	<code>PicamParameter_SensorActiveRightMargin</code> <b>16</b> from the value

## 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**

In PICam 4.x and Older Code, if you:		... in PICam 5.0, you need to:	
Action	Parameter	Action	Parameter
Get	<code>PicamParameter_ActiveLeftMargin</code>	Get	<code>PicamParameter_ActiveRightMargin</code>
Set	<code>PicamParameter_ActiveLeftMargin</code>	Set	<code>PicamParameter_ActiveRightMargin</code>
Get	<code>PicamParameter_ActiveRightMargin</code>	Get Subtract	<code>PicamParameter_ActiveLeftMargin</code> <b>24</b> from this value
Set	<code>PicamParameter_ActiveRightMargin</code>	Add Set	<b>24</b> to the value <code>PicamParameter_ActiveLeftMargin</code>
Get	<code>PicamParameter_SensorActiveLeftMargin</code>	Get	<code>PicamParameter_SensorActiveRightMargin</code>
Get	<code>PicamParameter_SensorActiveRightMargin</code>	Get Subtract	<code>PicamParameter_SensorActiveLeftMargin</code> <b>24</b> from this value

## 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)**

In PICam 4.x and Older Code, if you:		... in PICam 5.0, you need to:	
Action	Parameter	Action	Parameter
<b>ADC Quality: Electron Multiplied</b>			
Get	<code>PicamParameter_ActiveLeftMargin</code>	Get	<code>PicamParameter_ActiveRightMargin</code>
Set	<code>PicamParameter_ActiveLeftMargin</code>	Set	<code>PicamParameter_ActiveRightMargin</code>
Get	<code>PicamParameter_ActiveRightMargin</code>	Get Subtract	<code>PicamParameter_ActiveLeftMargin</code> <b>16</b> from the value
Set	<code>PicamParameter_ActiveRightMargin</code>	Add Set	<b>16</b> to the value <code>PicamParameter_ActiveLeftMargin</code>
Get	<code>PicamParameter_SensorActiveLeftMargin</code>	Get	<code>PicamParameter_SensorActiveRightMargin</code>
Get	<code>PicamParameter_SensorActiveRightMargin</code>	Get Subtract	<code>PicamParameter_SensorActiveLeftMargin</code> <b>16</b> from the value
<b>ADC Quality: Low Noise</b>			
Get	<code>PicamParameter_ActiveRightMargin</code>	Get Subtract	<code>PicamParameter_ActiveRightMargin</code> <b>16</b> from the value
Set	<code>PicamParameter_ActiveRightMargin</code>	Add Set	<b>16</b> to the value <code>PicamParameter_ActiveRightMargin</code>



**Table E-12: PICam 4.x (and Older) Code Changes: ProEM-HS: 512B/512BK/1024B and eXcelon Variants (Sheet 2 of 2)**

In PICam 4.x and Older Code, if you:		... in PICam 5.0, you need to:	
Action	Parameter	Action	Parameter
Get	PicamParameter_SensorActiveRightMargin	Get	PicamParameter_SensorActiveRightMargin
		Subtract	<b>16</b> from the value

## 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)**

In PICam 4.x and Older Code, if you:		... in PICam 5.0, you need to:	
Action	Parameter	Action	Parameter
<b>ADC Quality: Electron Multiplied</b>			
Get	<code>PicamParameter_ActiveLeftMargin</code>	Get	<code>PicamParameter_ActiveRightMargin</code>
Set	<code>PicamParameter_ActiveLeftMargin</code>	Set	<code>PicamParameter_ActiveRightMargin</code>
Get	<code>PicamParameter_ActiveRightMargin</code>	Get Subtract	<code>PicamParameter_ActiveLeftMargin</code> <b>16</b> from the value
Set	<code>PicamParameter_ActiveRightMargin</code>	Add Set	<b>16</b> to the value <code>PicamParameter_ActiveLeftMargin</code>
Get	<code>PicamParameter_SensorActiveLeftMargin</code>	Get	<code>PicamParameter_SensorActiveRightMargin</code>
Get	<code>PicamParameter_SensorActiveRightMargin</code>	Get Subtract	<code>PicamParameter_SensorActiveLeftMargin</code> <b>16</b> from the value
<b>ADC Quality: Low Noise</b>			
Get	<code>PicamParameter_ActiveRightMargin</code>	Get Subtract	<code>PicamParameter_ActiveRightMargin</code> <b>16</b> from the value
Set	<code>PicamParameter_ActiveRightMargin</code>	Add Set	<b>16</b> to the value <code>PicamParameter_ActiveRightMargin</code>
Get	<code>PicamParameter_SensorActiveRightMargin</code>	Get Subtract	<code>PicamParameter_SensorActiveRightMargin</code> <b>16</b> 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**

In PICam 4.x and Older Code, if you:		... in PICam 5.0, you need to:	
Action	Parameter	Action	Parameter
<b>ADC Quality: High Capacity</b>			
Get	<a href="#">PicamParameter_ActiveLeftMargin</a>	Get	<a href="#">PicamParameter_ActiveRightMargin</a>
Set	<a href="#">PicamParameter_ActiveLeftMargin</a>	Set	<a href="#">PicamParameter_ActiveRightMargin</a>
Get	<a href="#">PicamParameter_ActiveRightMargin</a>	Get	<a href="#">PicamParameter_ActiveLeftMargin</a>
Set	<a href="#">PicamParameter_ActiveRightMargin</a>	Set	<a href="#">PicamParameter_ActiveLeftMargin</a>

## 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**

In PICam 4.x and Older Code, if you:		... in PICam 5.0, you need to:	
Action	Parameter	Action	Parameter
<b>ADC Quality: Low Noise</b>			
Get	<code>PicamParameter_ActiveLeftMargin</code>	Get	<code>PicamParameter_ActiveRightMargin</code>
Set	<code>PicamParameter_ActiveLeftMargin</code>	Set	<code>PicamParameter_ActiveRightMargin</code>
Get	<code>PicamParameter_ActiveRightMargin</code>	Get	<code>PicamParameter_ActiveLeftMargin</code>
Set	<code>PicamParameter_ActiveRightMargin</code>	Set	<code>PicamParameter_ActiveLeftMargin</code>

# Warranty and Service

---

## Limited Warranty

Princeton Instruments, a division of Roper Scientific, Inc. (“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**

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, 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 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 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**

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, 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 Princeton Instruments factory or, at our option, a factory-authorized service center. You are responsible for the shipping costs to return the product to Princeton Instruments. International customers should contact their local 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**

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 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***

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***

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. 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***

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***

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. 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 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 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 Princeton Instruments.
2. You must notify the 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 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 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 Princeton Instruments factory or one of our authorized manufacturer's representatives or distributors.
6. Unless specified in the original purchase agreement, 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 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 Princeton Instruments factory or a 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 forgoing 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 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 Princeton Instruments factory of purchase, contact your authorized Princeton Instruments representative or reseller, or visit our technical support page at [www.princetoninstruments.com](http://www.princetoninstruments.com).

## Contact Information

Roper Scientific's manufacturing facility for this product is located at the following address:

Princeton Instruments  
3660 Quakerbridge Road  
Trenton, NJ 08619 (USA)

Tel: 1-800-874-9789 / 1-609-587-9797

Fax: 1-609-587-1970

Customer Support E-mail: [techsupport@princetoninstruments.com](mailto: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 Princeton Instruments' product lines.



*This page is intentionally blank.*



---

[www.princetoninstruments.com](http://www.princetoninstruments.com)

[info@princetoninstruments.com](mailto:info@princetoninstruments.com)

USA +1 877 474 2286 | FRANCE +33 (1) 60 86 03 65 | GERMANY +49 (0) 89 660 7793

UK & IRELAND +44 (0) 1628 472 346 | SINGAPORE +65 6408 6240 | CHINA +86 10 659 16460 | JAPAN +81 (3) 5639 2741