Royale

API Documentation

Version 4.9.0.161

Royale 4.9.0.161

API Documentation

4.9.0.161
Created: July 15, 2021

pmdtechnologies ag
Martinshardt 19
57074 Siegen | Germany

phone +49 271 238712-800
fax +49 271 238712-809
info@pmdtec.com
www.pmdtec.com

Chapter 1

Introduction

The **Royale** software package provides a light-weight camera framework for time-of-flight (ToF) cameras. While being tailored to PMD cameras, the framework enables partners and customers to evaluate and/or integrate 3D TOF technology on/in their target platform. This reduces time to first demo and time to market.

Royale contains all the logic which is required to operate a ToF based camera. The user need not care about setting registers, but can conveniently control the camera via a high-level interface. The Royale framework is completely designed in C++ using the C++11 standard.

1.1 Operating Systems

Royale supports the following operating systems:

- Windows 10
- Linux (tested on Ubuntu 18.04)
- OS X (tested on El Capitan 10.11)
- Android (tested on Android 8)
- Linux ARM (32Bit version tested on Raspbian GNU/Linux 10 (Buster) Raspberry Pi reference 2020-08-20 64Bit version tested on the Odroid C2 with Ubuntu Mate 16.04 ARM 64)

1.2 Hardware Requirements

Royale is tested on the following hardware configurations:

- PC, Intel i7-3770, 3.4 GHz, 4 cores (64 bit)
- MacBook Pro, Intel Core i5, 2.9 Ghz (64 bit)
- Samsung Galaxy S9

1.3 Getting Started

For a detailed guide on how to get started with Royale and your camera please have a look at the corresponding Getting Started Guide that can be found in the top folder of the package you received.

1.4 SDK and Examples

Besides the royaleviewer application, the package also provides a Royale SDK which can be used to develop applications using the PMD camera.

There are multiple samples included in the delivery package which should give you a brief overview about the exposed API. You can find an overview in samples/README_samples.md. The *doc* directory offers a detailed description of the Royale API which is a good starting point as well. You can also find the API documentation by opening the API_Documentation.html in the topmost folder of your platform package.

1.4.1 Debugging in Microsoft Visual Studio

To help debugging royale::Vector, royale::Pair and royale::String we provide a Natvis file for Visual Studio. Please take a look at the natvis.md file in the doc/natvis folder of your installation.

1.5 Matlab

In the delivery package for Windows you will find a Matlab wrapper for the Royale library. After the installation it can be found in the matlab subfolder of your installation directory. To use the wrapper you have to include this folder into your Matlab search paths. We also included some examples to show the usage of the wrapper. They can also be found in the matlab folder of your installation.

1.6 Python

In the package you will also find a wrapper to use Royale with Python. Unfortunately this wrapper will only work with specific Python versions. Please have a look at the README.md file in the Python folder to find out which versions are currently supported. In case you want to use a different Python version you can still use the SWIG interface file from the SWIG folder to compile your own wrapper.

1.7 Reference

FAQ: https://pmdtec.com/picofamily/faq/

1.8 License

See ThirdPartySoftware.txt and royale_license.txt. The source code of the open source software used in the Royale binary installation is available at https://oss.pmdtec.com/.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:


infineon
pmd
can you imagine

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:


infineon
pmd
can you imagine

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Module Documentation

6.1 Royale

Namespaces

- royale

6.1.1 Detailed Description


infineon
pmd
can you imagine

Chapter 7

Namespace Documentation

7.1 royale Namespace Reference

Namespaces

- parameter

Classes

- class CameraManager

  The CameraManager is responsible for detecting and creating instances of `ICameraDevices` one for each connected (supported) camera device.
- struct DepthData

  This structure defines the depth data which is delivered through the callback.
- struct DepthImage

  The depth image represents the depth and confidence for every pixel.
- struct DepthIRImage

  This represents combination of both depth and IR image.
- struct DepthPoint

  Encapsulates a 3D point in object space, with coordinates in meters.
- class ICameraDevice

  This is the main interface for talking to the time-of-flight camera system.
- class IDepthDataListener

  Provides the listener interface for consuming depth data from Royale.
- class IDepthImageListener

  Provides a listener interface for consuming depth images from Royale.
- class IDepthIRImageListener

  Provides a combined listener interface for consuming both depth and IR images from Royale.
- class IEvent

Interface for anything to be passed via IEventListener.

- class IEventListener

  This interface allows observers to receive events.
- class IExposureListener

  Provides the listener interface for handling auto-exposure updates in royale.
- class IExposureListener2

  Provides the listener interface for handling auto-exposure updates in royale.
- class IExtendedData

  Interface for getting additional data to the standard depth data.
- class IExtendedDataListener
- class IIRImageListener

  Provides the listener interface for consuming infrared images from Royale.
- struct IntermediateData

  This structure defines the Intermediate depth data which is delivered through the callback if the user has access level 2 for the CameraDevice.
- struct IntermediatePoint

  In addition to the standard depth point, the intermediate point also stores information which is calculated as temporaries in the processing pipeline.
- class IPlaybackStopListener
- class IRecord
- class IRecordStopListener

  This interface needs to be implemented if the client wants to get notified when recording stopped after the specified number of frames.
- class IReplay
- struct IRImage

  Infrared image with 8Bit mono information for every pixel.
- class ISparsePointCloudListener

  Provides the listener interface for consuming sparse point clouds from Royale.
- struct LensParameters

  This container stores the lens parameters from the camera module.
- struct RawData

  This structure defines the raw data which is delivered through the callback only exposed for access LEVEL 2.
- struct SparsePointCloud

  The sparse point cloud gives XYZ and confidence for every valid point.
- class Variant

  Implements a variant type which can take different basic data types, the default type is int and the value is set to zero.

Typedefs

- typedef royale::Vector< royale::Pair< royale::ProcessingFlag, royale::Variant > > ProcessingParameterVector

  This is a map combining a set of flags which can be set/altered in access LEVEL 2 and the set value as Variant type.
- typedef std::map< royale::ProcessingFlag, royale::Variant > ProcessingParameterMap
- typedef std::pair< royale::ProcessingFlag, royale::Variant > ProcessingParameterPair
- using StreamId = uint16_t

  The StreamId uniquely identifies a stream of measurements within a usecase.

Enumerations

- enum TriggerMode { MASTER = 0, SLAVE = 1 }

  Trigger mode used by the camera.
- enum CallbackData : uint16_t { None = 0x00, Raw = 0x01, Depth = 0x02, Intermediate = 0x04 }

  Specifies the type of data which should be captured and returned as callback.
- enum CameraAccessLevel { L1 = 1, L2 = 2, L3 = 3, L4 = 4 }

  This enum defines the access level.
- enum ExposureMode { MANUAL, AUTOMATIC }

  The ExposureMode is used to switch between manual and automatic exposure time handling.
- enum FilterLevel {
  Off = 0, Deprecated1 = 1, Deprecated2 = 2, Deprecated3 = 3,
  Deprecated4 = 4, IR1 = 5, IR2 = 6, AF1 = 7,
  CM1 = 8, Binning_1_Basic = 9, Binning_2_Basic = 10, Binning_3_Basic = 11,
  Binning_4_Basic = 12, Binning_8_Basic = 13, Binning_10_Basic = 14, Binning_1_Efficiency = 15,
  Binning_2_Efficiency = 16, Binning_3_Efficiency = 17, Binning_4_Efficiency = 18, Binning_8_Efficiency = 19,
  Binning_10_Efficiency = 20, Legacy = 200, Full = 255, Custom = 256 }

  Royale allows to set different filter levels.
- enum EventSeverity { ROYALE_INFO = 0, ROYALE_WARNING = 1, ROYALE_ERROR = 2, ROYALE_FATAL = 3 }

  Severity of an IEvent.
- enum EventType {
  ROYALE_CAPTURE_STREAM, ROYALE_DEVICE_DISCONNECTED, ROYALE_OVER_TEMPERATURE, ROYALE_RAW_FRAME_STAT
  ROYALE_EYE_SAFETY, ROYALE_PROCESSING, ROYALE_RECORDING, ROYALE_FRAME_DROP,
  ROYALE_UNKNOWN, ROYALE_ERROR_DESCRIPTION }

  Type of an IEvent.
- enum ProcessingFlag {
  ConsistencyTolerance_Float = 0, FlyingPixelsF0_Float, FlyingPixelsF1_Float, FlyingPixelsFarDist_Float,
  FlyingPixelsNearDist_Float, LowerSaturationThreshold_Int, UpperSaturationThreshold_Int, MPIAmpThreshold_Float,
  MPIDistThreshold_Float, MPINoiseDistance_Float, NoiseThreshold_Float, AdaptiveNoiseFilterType_Int,
  AutoExposureRefAmplitude_Float, UseAdaptiveNoiseFilter_Bool, UseAutoExposure_Bool, UseRemoveFlyingPixel_Bool,
  UseMPIFlagAverage_Bool, UseMPIFlag_Amp_Bool, UseMPIFlag_Dist_Bool, UseValidateImage_Bool,
  UseRemoveStrayLight_Bool, UseSparsePointCloud_Bool, UseFilter2Freq_Bool, GlobalBinning_Int,
  UseAdaptiveBinning_Bool, AutoExposureRefValue_Float, UseSmoothingFilter_Bool, SmoothingAlpha_Float,
  SmoothingFilterType_Int, UseFlagSBI_Bool, UseHoleFilling_Bool, Reserved1,
  Reserved2, Reserved3, Reserved4, Reserved5,
  Reserved6, Reserved7, Reserved8, AutoExpoMin_Int,
  AutoExpoMax_Int, SpectreProcessingType_Int, UseGrayImageFallbackAmplitude_Bool, GrayImageMeanMap_Int,
  NoiseFilterSigmaD_Float, NoiseFilterIterations_Int, FlyingPixelAngleLimit_Float, FlyingPixelAmpThreshold_Float,
  FlyingPixelMinNeighbors_Int, FlyingPixelMaxNeighbors_Int, FlyingPixelNoiseRatioThresh_Float, SmoothingFilterResetThreshold_Float,
  CCThresh_Int, PhaseNoiseThresh_Float, StraylightThreshold_Float, NoiseFilterSigmaA_Float,
  TwoFreqCombinationType_Int, UseCorrectMPI_Bool, NUM_FLAGS }

  This is a list of flags which can be set/altered in access LEVEL 2 in order to control the processing pipeline.
- enum SpectreProcessingType {
  AUTO = 1, CB_BINNED_WS = 2, NG = 3, AF = 4,
  CB_BINNED_NG = 5, GRAY_IMAGE = 6, CM_FI = 7, NUM_TYPES }

  This is a list of pipelines that can be set in Spectre.
- enum CameraStatus {
  SUCCESS = 0, RUNTIME_ERROR = 1024, DISCONNECTED = 1026, INVALID_VALUE = 1027,
  TIMEOUT = 1028, LOGIC_ERROR = 2048, NOT_IMPLEMENTED = 2049, OUT_OF_BOUNDS = 2050,
  RESOURCE_ERROR = 4096, FILE_NOT_FOUND = 4097, COULD_NOT_OPEN = 4098, DATA_NOT_FOUND = 4099,
  DEVICE_IS_BUSY = 4100, WRONG_DATA_FORMAT_FOUND = 4101, USECASE_NOT_SUPPORTED = 5001,

FRAMERATE_NOT_SUPPORTED = 5002,
EXPOSURE_TIME_NOT_SUPPORTED = 5003, DEVICE_NOT_INITIALIZED = 5004, CALIBRATION_DATA_ERROR = 5005, INSUFFICIENT_PRIVILEGES = 5006,
DEVICE_ALREADY_INITIALIZED = 5007, EXPOSURE_MODE_INVALID = 5008, NO_CALIBRATION_DATA = 5009, INSUFFICIENT_BANDWIDTH = 5010,
DUTYCYCLE_NOT_SUPPORTED = 5011, SPECTRE_NOT_INITIALIZED = 5012, NO_USE_CASES = 5013, NO_USE_CASES_FOR_LEVEL = 5014,
FSM_INVALID_TRANSITION = 8096, UNKNOWN = 0x7fffff01 }

- enum VariantType { Int, Float, Bool, Enum }

Functions

- ROYALE_API royale::String getFilterLevelName (royale::FilterLevel level)
- ROYALE_API royale::FilterLevel getFilterLevelFromName (royale::String name)
- ROYALE_API royale::String getProcessingFlagName (royale::ProcessingFlag mode)

  For debugging, printable strings corresponding to the ProcessingFlag enumeration.
- ROYALE_API bool parseProcessingFlagName (const royale::String &modeName, royale::ProcessingFlag &processingFlag)

  Convert a string received from getProcessingFlagName back into its ProcessingFlag.
- ROYALE_API ProcessingParameterMap combineProcessingMaps (const ProcessingParameterMap &a, const ProcessingParameterMap &b)

  Takes ProcessingParameterMaps a and b and returns a combination of both.
- ROYALE_API royale::String getSpectreProcessingTypeName (royale::SpectreProcessingType mode)

  Converts the given processing type into a readable string.
- ROYALE_API bool getSpectreProcessingTypeFromName (const royale::String &modeName, royale::SpectreProcessingType &processingType)

  Converts the name of a processing type into an enum value.
- ROYALE_API royale::String getStatusString (royale::CameraStatus status)

  Get a human-readable description for a given error message.
- inline ::std::ostream & operator<< (::std::ostream &os, royale::CameraStatus status)

7.1.1 Typedef Documentation

7.1.1.1 ProcessingParameterMap

```
typedef std::map< royale::ProcessingFlag, royale::Variant > ProcessingParameterMap
```

7.1.1.2 ProcessingParameterPair

```
typedef std::pair< royale::ProcessingFlag, royale::Variant > ProcessingParameterPair
```

7.1.1.3 ProcessingParameterVector

```
typedef royale::Vector< royale::Pair<royale::ProcessingFlag, royale::Variant> > ProcessingParameterVector
```

This is a map combining a set of flags which can be set/altered in access LEVEL 2 and the set value as Variant type.

The proposed minimum and maximum limits are recommendations for reasonable results. Values beyond these boundaries are permitted, but are currently neither evaluated nor verified.

7.1.1.4 StreamId

```
using StreamId = uint16_t
```

The StreamId uniquely identifies a stream of measurements within a usecase.

A stream is a sequence of periodic measurements having the same depth range, exposure times and other settings. Most usecases will only produce a single stream, but with mixed-mode usecases, there may be more than one. A typical mixed-mode usecase may for example include one stream designed for hand tracking (which needs a high frame rate but can work with reduced depth range) and another one for environment scanning (full depth range at a lower frame rate).

StreamId 0 is not a valid StreamId, but can (for backward compatibility) be used as referring to the single stream contained in non mixed mode usecases in most API functions that expect a StreamId. This allows applications that don't make use of mixed mode to work without having to deal with StreamIds. Applications that need to use mixed mode usecases will have to provide the correct IDs in the API as 0 is not accepted as StreamId if a mixed mode usecase is active.

7.1.2 Enumeration Type Documentation

7.1.2.1 CallbackData

```
enum CallbackData :  uint16_t  [strong]
```

Specifies the type of data which should be captured and returned as callback.

Enumerator

None	only get the callback but no data delivery
Raw	raw frames, if exclusively used no processing pipe is executed (no calibration data is needed)
Depth	one depth and grayscale image will be delivered for the complete sequence
Intermediate	all intermediate data will be delivered which are generated in the processing pipeline

7.1.2.2 CameraAccessLevel

`enum CameraAccessLevel [strong]`

This enum defines the access level.

For Royale >= 3.5 this can be directly cast to unsigned ints (Level 1 equals 1, Level 2 equals 2, ...).

Enumerator

L1	Level 1 access provides depth data using standard, known-working configurations.
L2	Level 2 access provides raw data, e.g. for custom processing pipelines
L3	Level 3 access enables you to overwrite exposure limits.
L4	Level 4 access is for bringing up new camera modules.

7.1.2.3 CameraStatus

`enum CameraStatus [strong]`

Enumerator

SUCCESS	Indicates that there isn't an error.
RUNTIME_ERROR	Something unexpected happened.
DISCONNECTED	Camera device is disconnected.
INVALID_VALUE	The value provided is invalid.
TIMEOUT	The connection got a timeout.
LOGIC_ERROR	This does not make any sense here.
NOT_IMPLEMENTED	This feature is not implemented yet.
OUT_OF_BOUNDS	Setting/parameter is out of specified range.
RESOURCE_ERROR	Cannot access resource.
FILE_NOT_FOUND	Specified file was not found.
COULD_NOT_OPEN	Cannot open resource.
DATA_NOT_FOUND	No data available where expected.
DEVICE_IS_BUSY	Another action is in progress.
WRONG_DATA_FORMAT_FOUND	A resource was expected to be in one data format, but was in a different (recognisable) format.
USECASE_NOT_SUPPORTED	This use case is not supported.

Enumerator

FRAMERATE_NOT_SUPPORTED	The specified frame rate is not supported.
EXPOSURE_TIME_NOT_SUPPORTED	The exposure time is not supported.
DEVICE_NOT_INITIALIZED	The device seems to be uninitialized.
CALIBRATION_DATA_ERROR	The calibration data is not readable.
INSUFFICIENT_PRIVILEGES	The camera access level does not allow to call this operation.
DEVICE_ALREADY_INITIALIZED	The camera was already initialized.
EXPOSURE_MODE_INVALID	The current set exposure mode does not support this operation.
NO_CALIBRATION_DATA	The method cannot be called since no calibration data is available.
INSUFFICIENT_BANDWIDTH	The interface to the camera module does not provide a sufficient bandwidth.
DUTYCYCLE_NOT_SUPPORTED	The duty cycle is not supported.
SPECTRE_NOT_INITIALIZED	Spectre was not initialized properly.
NO_USE_CASES	The camera offers no use cases.
NO_USE_CASES_FOR_LEVEL	The camera offers no use cases for the current access level.
FSM_INVALID_TRANSITION	Camera module state machine does not support current transition.
UNKNOWN	Catch-all failure.

7.1.2.4 EventSeverity

`enum EventSeverity [strong]`

Severity of an IEvent.

Enumerator

ROYALE_INFO	Information only event.
ROYALE_WARNING	Potential issue detected (e.g. soft overtemperature limit reached).
ROYALE_ERROR	Errors occurred during operation. The operation (e.g. recording or stream capture) has failed and was stopped.
ROYALE_FATAL	A severe error was detected. The corresponding ICameraDevice is no longer in a usable state.

7.1.2.5 EventType

`enum EventType [strong]`

Type of an IEvent.

Enumerator

ROYALE_CAPTURE_STREAM	The event was detected as part of the mechanism to receive image data. For events of this class, ROYALE_WARNING is likely to indicate dropped frames, and ROYALE_ERROR is likely to indicate that no more frames will be captured until after the next call to ICameraDevice::startCapture().
ROYALE_DEVICE_DISCONNECTED	Royale is no longer able to talk to the camera; this is always severity ROYALE_FATAL.
ROYALE_OVER_TEMPERATURE	The camera has become hot, likely because of the illumination. For events of this class, ROYALE_WARNING indicates that the device is still functioning but is near to the temperature at which it will be shut down for safety, and ROYALE_ERROR indicates that the safety mechanism has been triggered.
ROYALE_RAW_FRAME_STATS	This event is sent regularly during capturing. The trigger for sending this event is implementation defined and varies for different use cases, but the timing is normally around one per second. If all frames were successfully received then it will be ROYALE_INFO, if any were dropped then it will be ROYALE_WARNING.
ROYALE_EYE_SAFETY	This event indicates that the camera's internal monitor of the power used by the illumination has been triggered, which is never expected to happen with the use cases in production devices. The capturing should already been stopped when receiving this event, as the illumination will stay turned off and most of the received data will be corrupted. This is always severity ROYALE_FATAL.
ROYALE_PROCESSING	This event is sent if, for example, the backend of the processing is changed.
ROYALE_RECORDING	This event is sent if something happens during the recording.
ROYALE_FRAME_DROP	This event is sent when a frame drop occurs.
ROYALE_UNKNOWN	The event type is for any event for which there is no official API specification.
ROYALE_ERROR_DESCRIPTION	This event type can be used to get more information about errors that are returned by Royale.

7.1.2.6 ExposureMode

```
enum ExposureMode  [strong]
```

The ExposureMode is used to switch between manual and automatic exposure time handling.

Enumerator

MANUAL	Camera exposure mode set to manual.
AUTOMATIC	Camera exposure mode set to automatic.

7.1.2.7 FilterLevel

`enum FilterLevel [strong]`

Royale allows to set different filter levels.

Internally these represent different configurations of the processing pipeline. Which filter levels are available depends on the currently selected pipeline.

Enumerator

Off	Turn off all filtering of the data (validation will still be enabled) (WS pipeline)
Deprecated1	Not available anymore.
Deprecated2	Not available anymore.
Deprecated3	Not available anymore.
Deprecated4	Not available anymore.
IR1	Only available for the IR/FaceID pipeline : IR_IlluOn-FPN.
IR2	Only available for the IR/FaceID pipeline : IR_IlluOn-IlluOff.
AF1	Standard setting for the auto focus pipeline.
CM1	Standard setting for the coded modulation pipeline.
Binning_1_Basic	NG pipeline : basic kernels with binning size 1.
Binning_2_Basic	NG pipeline : basic kernels with binning size 2.
Binning_3_Basic	NG pipeline : basic kernels with binning size 3.
Binning_4_Basic	NG pipeline : basic kernels with binning size 4.
Binning_8_Basic	NG pipeline : basic kernels with binning size 8.
Binning_10_Basic	NG pipeline : basic kernels with binning size 10.
Binning_1_Efficiency	NG pipeline : efficiency kernels with binning size 1.
Binning_2_Efficiency	NG pipeline : efficiency kernels with binning size 2.
Binning_3_Efficiency	NG pipeline : efficiency kernels with binning size 3.
Binning_4_Efficiency	NG pipeline : efficiency kernels with binning size 4.
Binning_8_Efficiency	NG pipeline : efficiency kernels with binning size 8.
Binning_10_Efficiency	NG pipeline : efficiency kernels with binning size 10.
Legacy	Standard settings for older cameras (WS pipeline)
Full	Enable all filters that are available for this camera (WS pipeline)
Custom	Value returned by getFilterLevel if the processing parameters differ from all of the presets.

7.1.2.8 ProcessingFlag

`enum ProcessingFlag [strong]`

This is a list of flags which can be set/altered in access LEVEL 2 in order to control the processing pipeline.

The suffix type indicates the expected Variant type. For a more detailed description of the different parameters please refer to the documentation you will receive after getting LEVEL 2 access.

Make sure to retrieve and update all the flags when SpectreProcessingType_Int is altered.

Keep in mind, that if this list is changed, the map with names has to be adapted!

Enumerator

ConsistencyTolerance_Float	Consistency limit for asymmetry validation.
FlyingPixelsF0_Float	Scaling factor for lower depth value normalization.
FlyingPixelsF1_Float	Scaling factor for upper depth value normalization.
FlyingPixelsFarDist_Float	Upper normalized threshold value for flying pixel detection.
FlyingPixelsNearDist_Float	Lower normalized threshold value for flying pixel detection.
LowerSaturationThreshold_Int	Lower limit for valid raw data values.
UpperSaturationThreshold_Int	Upper limit for valid raw data values.
MPIAmpThreshold_Float	Threshold for MPI flags triggered by amplitude discrepancy.
MPIDistThreshold_Float	Threshold for MPI flags triggered by distance discrepancy.
MPINoiseDistance_Float	Threshold for MPI flags triggered by noise.
NoiseThreshold_Float	Upper threshold for final distance noise.
AdaptiveNoiseFilterType_Int	Kernel type of the adaptive noise filter.
AutoExposureRefAmplitude_Float	DEPRECATED : The reference amplitude for the new exposure estimate.
UseAdaptiveNoiseFilter_Bool	Activate spatial filter reducing the distance noise.
UseAutoExposure_Bool	DEPRECATED : Activate dynamic control of the exposure time.
UseRemoveFlyingPixel_Bool	Activate FlyingPixel flag.
UseMPIFlagAverage_Bool	Activate spatial averaging MPI value before thresholding.
UseMPIFlag_Amp_Bool	Activates MPI-amplitude flag.
UseMPIFlag_Dist_Bool	Activates MPI-distance flag.
UseValidateImage_Bool	Activates output image validation.
UseRemoveStrayLight_Bool	Activates the removal of stray light.
UseSparsePointCloud_Bool	DEPRECATED : Creates a sparse-point cloud in Spectre.
UseFilter2Freq_Bool	Activates 2 frequency filtering.
GlobalBinning_Int	Sets the size of the global binning kernel.
UseAdaptiveBinning_Bool	DEPRECATED : Activates adaptive binning.
AutoExposureRefValue_Float	The reference value for the new exposure estimate.
UseSmoothingFilter_Bool	Enable/Disable the smoothing filter.
SmoothingAlpha_Float	The alpha value used for the smoothing filter.
SmoothingFilterType_Int	Determines the type of smoothing that is used.

Enumerator

UseFlagSBI_Bool	Enable/Disable the flagging of pixels where the SBI was active.
UseHoleFilling_Bool	Enable/Disable the hole filling algorithm.
Reserved1	
Reserved2	
Reserved3	
Reserved4	
Reserved5	
Reserved6	
Reserved7	
Reserved8	
AutoExpoMin_Int	The minimum value for the auto exposure algorithm (new values will be bound by use case limits)
AutoExpoMax_Int	The maximum value for the auto exposure algorithm (new values will be bound by use case limits)
SpectreProcessingType_Int	The type of processing used by Spectre.
UseGrayImageFallbackAmplitude_Bool	Uses the fallback image in the gray image pipeline as amplitude image.
GrayImageMeanMap_Int	Value where the mean of the gray image is mapped to.
NoiseFilterSigmaD_Float	SigmaD.
NoiseFilterIterations_Int	Iterations of the noise filter.
FlyingPixelAngleLimit_Float	Angle limit of the flying pixel algorithm.
FlyingPixelAmpThreshold_Float	Amplitude threshold of the flying pixel algorithm.
FlyingPixelMinNeighbors_Int	Minimum neighbors for the flying pixel algorithm.
FlyingPixelMaxNeighbors_Int	Maximum neighbors for the flying pixel algorithm.
FlyingPixelNoiseRatioThresh_Float	Noiseratio threshold.
SmoothingFilterResetThreshold_Float	Reset value for the smoothing.
CCThresh_Int	Connected components threshold.
PhaseNoiseThresh_Float	PhaseNoise threshold.
StraylightThreshold_Float	Straylight threshold.
NoiseFilterSigmaA_Float	SigmaA.
TwoFreqCombinationType_Int	Determines which algorithm will be used for combining the two frequencies.
UseCorrectMPI_Bool	Turn on/off the MPI correction of the spot processing algorithm.
NUM_FLAGS	

7.1.2.9 SpectreProcessingType

`enum SpectreProcessingType [strong]`

This is a list of pipelines that can be set in Spectre.

Which pipelines are available depends on the module and the currently selected use case.

Enumerator

AUTO	
CB_BINNED_WS	
NG	
AF	
CB_BINNED_NG	
GRAY_IMAGE	
CM_FI	
NUM_TYPES	

7.1.2.10 TriggerMode

`enum TriggerMode [strong]`

Trigger mode used by the camera.

Enumerator

MASTER	The camera acts as a master.
SLAVE	The camera acts as a slave.

7.1.2.11 VariantType

`enum VariantType [strong]`

Enumerator

Int	
Float	
Bool	
Enum	

7.1.3 Function Documentation

7.1.3.1 combineProcessingMaps()

```
ROYALE_API ProcessingParameterMap royale::combineProcessingMaps (
             const ProcessingParameterMap & a,
             const ProcessingParameterMap & b )
```

Takes ProcessingParameterMaps a and b and returns a combination of both.

Keys that exist in both maps will take the value of map b.

7.1.3.2 getFilterLevelFromName()

```
ROYALE_API royale::FilterLevel royale::getFilterLevelFromName (
             royale::String name )
```

7.1.3.3 getFilterLevelName()

```
ROYALE_API royale::String royale::getFilterLevelName (
             royale::FilterLevel level )
```

7.1.3.4 getProcessingFlagName()

```
ROYALE_API royale::String royale::getProcessingFlagName (
             royale::ProcessingFlag mode )
```

For debugging, printable strings corresponding to the ProcessingFlag enumeration.

The returned value is copy of the processing flag name. If the processing flag is not found an empty string will be returned.

These strings will not be localized.

7.1.3.5 getSpectreProcessingTypeFromName()

```
ROYALE_API bool royale::getSpectreProcessingTypeFromName (
             const royale::String & modeName,
             royale::SpectreProcessingType & processingType )
```

Converts the name of a processing type into an enum value.

7.1.3.6 getSpectreProcessingTypeName()

```
ROYALE_API royale::String royale::getSpectreProcessingTypeName (
            royale::SpectreProcessingType mode )
```

Converts the given processing type into a readable string.

7.1.3.7 getStatusString()

```
ROYALE_API royale::String royale::getStatusString (
            royale::CameraStatus status )
```

Get a human-readable description for a given error message.

Note: These descriptions are in English and are intended to help developers, they're not translated to the current locale.

Examples

sampleRecordRRF.cpp.

7.1.3.8 operator<<()

```
inline ::std::ostream& royale::operator<< (
            ::std::ostream & os,
            royale::CameraStatus status )
```

7.1.3.9 parseProcessingFlagName()

```
ROYALE_API bool royale::parseProcessingFlagName (
            const royale::String & modeName,
            royale::ProcessingFlag & processingFlag )
```

Convert a string received from getProcessingFlagName back into its ProcessingFlag.

If the processing flag name is not found the method returns false. Else the method will return true.

7.2 royale::parameter Namespace Reference

Chapter 8

Class Documentation

8.1 CameraManager Class Reference

The CameraManager is responsible for detecting and creating instances of `ICameraDevices` one for each connected (supported) camera device.

```
#include <CameraManager.hpp>
```

Public Member Functions

- ROYALE_API CameraManager (const royale::String &activationCode="")
  Constructor of the CameraManager.
- ROYALE_API ~CameraManager ()
  Destructor of the CameraManager.
- ROYALE_API royale::Vector< royale::String > getConnectedCameraList ()
  Returns the list of connected camera modules identified by a unique ID (serial number).
- ROYALE_API std::unique_ptr< royale::ICameraDevice > createCamera (const royale::String &cameraId, const royale::TriggerMode mode=TriggerMode::MASTER)
  Creates a master or slave camera object ICameraDevice identified by its ID.
- ROYALE_API royale::Vector< royale::String > getConnectedCameraNames ()
  This function has to be called after getConnectedCameraList().
- ROYALE_API royale::CameraStatus registerEventListener (royale::IEventListener *listener)
  Register an event listener with this camera manager.
- ROYALE_API royale::CameraStatus unregisterEventListener ()
  Unregister the current event listener of this camera manager.
- ROYALE_API void setCacheFolder (const royale::String &path)
  Sets the folder that will be used for caching e.g.

Static Public Member Functions

- static ROYALE_API royale::CameraAccessLevel getAccessLevel (const royale::String &activationCode="")
  Retrieves the access level to the given activation code.

8.1.1 Detailed Description

The CameraManager is responsible for detecting and creating instances of `ICameraDevices` one for each connected (supported) camera device.

Depending on the provided activation code access `Level 2` or `Level 3` can be created. Due to eye safety reasons, `Level 3` is for internal purposes only. Once a known time-of-flight device is detected, the according communication (e.g. via USB) is established and the camera device is ready.

Examples

sampleCameraInfo.cpp, sampleExportPLY.cpp, sampleIReplay.cpp, sampleOpenCV.cpp, sampleRecordRRF.cpp, and sampleRetrieveData.cpp.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 CameraManager()

```
ROYALE_API CameraManager (
            const royale::String & activationCode = "" )  [explicit]
```

Constructor of the CameraManager.

An empty activationCode only allows to get an ICameraDevice. A valid activation code also allows to gain `Level 2` or `Level 3` access rights.

8.1.2.2 ~CameraManager()

```
ROYALE_API ~CameraManager ( )
```

Destructor of the CameraManager.

8.1.3 Member Function Documentation

8.1.3.1 createCamera()

```
ROYALE_API std::unique_ptr<royale::ICameraDevice> createCamera (
            const royale::String & cameraId,
            const royale::TriggerMode mode = TriggerMode::MASTER )
```

Creates a master or slave camera object ICameraDevice identified by its ID.

The ID can be

- The ID which is returned by getConnectedCameraList (representing a physically connected camera)
- A given filename of a previously recorded stream

If the ID or filename are not correct, a nullptr will be returned. The ownership is transfered to the caller, which means that the ICameraDevice is still valid once the CameraManager is out of scope.

In case of a given filename, the returned ICameraDevice can also be dynamically casted to an IReplay interface which offers more playback functionality.

If the camera is opened as a slave it will not receive a start signal from Royale, but will wait for the external trigger signal. Please have a look at the master/slave example which shows how to deal with multiple cameras.

Parameters

cameraId	Unique ID either the ID returned from getConnectedCameraList of a filename for a recorded stream
mode	Tell Royale to open this camera as master or slave.

Returns

ICameraDevice object if ID was found, nullptr otherwise

Examples

sampleCameraInfo.cpp, sampleExportPLY.cpp, sampleOpenCV.cpp, sampleRecordRRF.cpp, and sampleRetrieveData.cpp.

8.1.3.2 getAccessLevel()

```
static ROYALE_API royale::CameraAccessLevel getAccessLevel (
            const royale::String & activationCode = "" )  [static]
```

Retrieves the access level to the given activation code.

8.1.3.3 getConnectedCameraList()

```
ROYALE_API royale::Vector<royale::String> getConnectedCameraList ( )
```

Returns the list of connected camera modules identified by a unique ID (serial number).

This call tries to connect to each plugged-in camera and queries for its unique serial number. Found cameras need to be fetched by calling createCamera(). This in turn moves the ownership of the CameraDevice to the caller of createCamera(). Calling getConnectedCameraList() twice will automatically close all unused ICameraDevices that were returned from the first call. **Calling this function twice is not the expected usage for this function!** Once the scope of CameraManager ends, all (other) unused ICameraDevices will also be closed automatically. The createCamera() keeps the ICameraDevice beyond the scope of the CameraManager since the ownership is given to the caller.

WARNING: please also only have one instance of CameraManager at the same time! royale does not support multiple instances of CameraManager in parallel. The caller will receive events through the event listener registered with registerEventListener() under the respective conditions:

Event	Condition
EventImagerConfigNotFound	The external configuration file is not found.
EventProbedDevicesNotMatched	There are connected devices which may be cameras but none of them were found suitable for inclusion in the connected camera list.

Returns

list of connected camera IDs

Examples

sampleCameraInfo.cpp, sampleOpenCV.cpp, sampleRecordRRF.cpp, and sampleRetrieveData.cpp.

8.1.3.4 getConnectedCameraNames()

```
ROYALE_API royale::Vector<royale::String> getConnectedCameraNames ( )
```

This function has to be called after getConnectedCameraList().

It returns the list of connected camera names without creating them.

Returns

list of connected camera Names

8.1.3.5 registerEventListener()

```
ROYALE_API royale::CameraStatus registerEventListener (
            royale::IEventListener * listener )
```

Register an event listener with this camera manager.

The listener may receive an event if an error occurs during a following call to one of the camera manager's other methods. For the conditions under which an error event occurs, see the respective method:

- getConnectedCameraList()

Parameters

listener	the listener to be registered.

Returns

SUCCESS if the event listener was successfully registered.

See also

unregisterEventListener().

Examples

sampleCameraInfo.cpp.

8.1.3.6 setCacheFolder()

```
ROYALE_API void setCacheFolder (
            const royale::String & path )
```

Sets the folder that will be used for caching e.g.

calibration files.

8.1.3.7 unregisterEventListener()

```
ROYALE_API royale::CameraStatus unregisterEventListener ( )
```

Unregister the current event listener of this camera manager.

This method blocks until all pending events are sent to the listener. A registered event listener should be unregistered before it is deallocated. The event listener is automatically unregistered when this camera manager is deallocated.

Returns

SUCCESS if the event listener was successfully unregistered.

See also

registerEventListener().

Examples

sampleCameraInfo.cpp.

The documentation for this class was generated from the following file:

- CameraManager.hpp

8.2 DepthData Struct Reference

This structure defines the depth data which is delivered through the callback.

```
#include <DepthData.hpp>
```

Public Member Functions

- DepthData & operator= (const DepthData &dd)

Public Attributes

- int version
  version number of the data format
- std::chrono::microseconds timeStamp
  timestamp in microseconds precision (time since epoch 1970)
- StreamId streamId
  stream which produced the data
- uint16_t width
  width of depth image
- uint16_t height
  height of depth image
- royale::Vector< uint32_t > exposureTimes
  exposureTimes retrieved from CapturedUseCase
- royale::Vector< royale::DepthPoint > points
  array of points
- bool hasDepth
  to check presence of depth information

8.2.1 Detailed Description

This structure defines the depth data which is delivered through the callback.

This data comprises a dense 3D point cloud with the size of the depth image (width, height). The points vector encodes an array (row-based) with the size of (width x height). Based on the `depthConfidence`, the user can decide to use the 3D point or not. The point cloud uses a right handed coordinate system (x -> right, y -> down, z -> in viewing direction).

Although the points are provided as a (width ∗ height) array and are arranged in a grid, treating them as simple square pixels will provide a distorted image, because they are not necessarily in a rectilinear projection; it is more likely that the camera would have a wide-angle or even fish-eye lens. Each individual DepthPoint provides x and y coordinates in addition to the z cooordinate, these values in the individual DepthPoints will match the lens of the camera.

Examples

sampleExportPLY.cpp, sampleIReplay.cpp, sampleOpenCV.cpp, and sampleRetrieveData.cpp.

8.2.2 Member Function Documentation

8.2.2.1 operator=()

```
DepthData& operator= (
            const DepthData & dd )  [inline]
```

8.2.3 Member Data Documentation

8.2.3.1 exposureTimes

`royale::Vector<uint32_t> exposureTimes`

exposureTimes retrieved from CapturedUseCase

Examples

sampleRetrieveData.cpp.

8.2.3.2 hasDepth

`bool hasDepth`

to check presence of depth information

8.2.3.3 height

`uint16_t height`

height of depth image

Examples

sampleOpenCV.cpp, and sampleRetrieveData.cpp.

8.2.3.4 points

`royale::Vector<royale::DepthPoint> points`

array of points

Examples

sampleExportPLY.cpp, sampleOpenCV.cpp, and sampleRetrieveData.cpp.

8.2.3.5 streamId

```
StreamId streamId
```

stream which produced the data

Examples

sampleRetrieveData.cpp.

8.2.3.6 timeStamp

```
std::chrono::microseconds timeStamp
```

timestamp in microseconds precision (time since epoch 1970)

Examples

sampleIReplay.cpp.

8.2.3.7 version

```
int version
```

version number of the data format

8.2.3.8 width

```
uint16_t width
```

width of depth image

Examples

sampleOpenCV.cpp, and sampleRetrieveData.cpp.

The documentation for this struct was generated from the following file:

- DepthData.hpp

8.3 DepthImage Struct Reference

The depth image represents the depth and confidence for every pixel.

`#include <DepthImage.hpp>`

Public Attributes

- int64_t timestamp
  timestamp for the frame
- StreamId streamId
  stream which produced the data
- uint16_t width
  width of depth image
- uint16_t height
  height of depth image
- royale::Vector< uint16_t > cdData
  depth and confidence for the pixel

8.3.1 Detailed Description

The depth image represents the depth and confidence for every pixel.

The least significant 13 bits are the depth (z value along the optical axis) in millimeters. 0 stands for invalid measurement / no data.

The most significant 3 bits correspond to a confidence value. 0 is the highest confidence, 7 the second highest, and 1 the lowest.

note The meaning of the confidence bits changed between Royale v3.14.0 and v3.15.0. Before v3.15.0, zero was lowest and 7 was highest. Because of this, the member was renamed from "data" to "cdData".

8.3.2 Member Data Documentation

8.3.2.1 cdData

`royale::Vector<uint16_t> cdData`

depth and confidence for the pixel

8.3.2.2 height

`uint16_t height`

height of depth image

8.3.2.3 streamId

`StreamId streamId`

stream which produced the data

8.3.2.4 timestamp

`int64_t timestamp`

timestamp for the frame

8.3.2.5 width

`uint16_t width`

width of depth image

The documentation for this struct was generated from the following file:

- DepthImage.hpp

8.4 DepthIRImage Struct Reference

This represents combination of both depth and IR image.

`#include <DepthIRImage.hpp>`

Public Attributes

- int64_t timestamp
  timestamp for the frame
- StreamId streamId
  stream which produced the data
- uint16_t width
  width of depth image
- uint16_t height
  height of depth image
- royale::Vector< uint16_t > dpData
  depth and confidence for the pixel
- royale::Vector< uint8_t > irData
  8Bit mono IR image

8.4.1 Detailed Description

This represents combination of both depth and IR image.

Provides depth, confidence and IR 8Bit mono information for every pixel.

8.4.2 Member Data Documentation

8.4.2.1 dpData

```
royale::Vector<uint16_t> dpData
```

depth and confidence for the pixel

8.4.2.2 height

```
uint16_t height
```

height of depth image

8.4.2.3 irData

`royale::Vector<uint8_t> irData`

8Bit mono IR image

8.4.2.4 streamId

`StreamId streamId`

stream which produced the data

8.4.2.5 timestamp

`int64_t timestamp`

timestamp for the frame

8.4.2.6 width

`uint16_t width`

width of depth image

The documentation for this struct was generated from the following file:

- DepthIRImage.hpp

8.5 DepthPoint Struct Reference

Encapsulates a 3D point in object space, with coordinates in meters.

`#include <DepthData.hpp>`

Public Attributes

- float x
  X coordinate [meters].
- float y
  Y coordinate [meters].
- float z
  Z coordinate [meters].
- float noise
  noise value [meters]
- uint16_t grayValue
  16-bit gray value
- uint8_t depthConfidence
  value from 0 (invalid) to 255 (full confidence)

8.5.1 Detailed Description

Encapsulates a 3D point in object space, with coordinates in meters.

In addition to the X/Y/Z coordinate each point also includes a gray value, a noise standard deviation, and a depth confidence value.

8.5.2 Member Data Documentation

8.5.2.1 depthConfidence

```
uint8_t depthConfidence
```

value from 0 (invalid) to 255 (full confidence)

8.5.2.2 grayValue

```
uint16_t grayValue
```

16-bit gray value

8.5.2.3 noise

```
float noise
```

noise value [meters]

8.5.2.4 x

```
float x
```

X coordinate [meters].

8.5.2.5 y

```
float y
```

Y coordinate [meters].

8.5.2.6 z

```
float z
```

Z coordinate [meters].

The documentation for this struct was generated from the following file:

- DepthData.hpp

8.6 ICameraDevice Class Reference

This is the main interface for talking to the time-of-flight camera system.

```
#include <ICameraDevice.hpp>
```

Public Member Functions

- virtual ~ICameraDevice ()
- virtual royale::CameraStatus initialize ()=0

  LEVEL 1 Initializes the camera device and sets the first available use case.
- virtual royale::CameraStatus initialize (const royale::String &initUseCase)=0

  LEVEL 1 Initialize the camera and configure the system for the specified use case.
- virtual royale::CameraStatus getId (royale::String &id) const =0

  LEVEL 1 Get the ID of the camera device.
- virtual royale::CameraStatus getCameraName (royale::String &cameraName) const =0

  LEVEL 1 Returns the associated camera name as a string which is defined in the CoreConfig of each module.
- virtual royale::CameraStatus getCameraInfo (royale::Vector< royale::Pair< royale::String, royale::String >> &camInfo) const =0
- virtual royale::CameraStatus setUseCase (const royale::String &name)=0

  LEVEL 1 Sets the use case for the camera.
- virtual royale::CameraStatus getUseCases (royale::Vector< royale::String > &useCases) const =0

  LEVEL 1 Returns all use cases which are supported by the connected module and valid for the current selected CallbackData information (e.g.
- virtual royale::CameraStatus getStreams (royale::Vector< royale::StreamId > &streams) const =0

  LEVEL 1 Get the streams associated with the current use case.
- virtual royale::CameraStatus getNumberOfStreams (const royale::String &name, uint32_t &nrStreams) const =0

  LEVEL 1 Retrieves the number of streams for a specified use case.
- virtual royale::CameraStatus getCurrentUseCase (royale::String &useCase) const =0

  LEVEL 1 Gets the current use case as string.
- virtual royale::CameraStatus setExposureTime (uint32_t exposureTime, royale::StreamId streamId=0)=0

  LEVEL 1 Change the exposure time for the supported operated operation modes.
- virtual royale::CameraStatus setExposureMode (royale::ExposureMode exposureMode, royale::StreamId streamId=0)=0

  LEVEL 1 Change the exposure mode for the supported operated operation modes.
- virtual royale::CameraStatus getExposureMode (royale::ExposureMode &exposureMode, royale::StreamId streamId=0) const =0

  LEVEL 1 Retrieves the current mode of operation for acquisition of the exposure time.
- virtual royale::CameraStatus getExposureLimits (royale::Pair< uint32_t, uint32_t > &exposureLimits, royale::StreamId streamId=0) const =0

  LEVEL 1 Retrieves the minimum and maximum allowed exposure limits of the specified operation mode.
- virtual royale::CameraStatus registerDataListener (royale::IDepthDataListener *listener)=0
- virtual royale::CameraStatus unregisterDataListener ()=0

  LEVEL 1 Unregisters the data depth listener.
- virtual royale::CameraStatus registerDepthImageListener (royale::IDepthImageListener *listener)=0

  LEVEL 1 Once registering the data listener, Android depth image data is sent via the callback function.
- virtual royale::CameraStatus unregisterDepthImageListener ()=0

  LEVEL 1 Unregisters the depth image listener.
- virtual royale::CameraStatus registerSparsePointCloudListener (royale::ISparsePointCloudListener *listener)=0

  LEVEL 1 Once registering the data listener, Android point cloud data is sent via the callback function.
- virtual royale::CameraStatus unregisterSparsePointCloudListener ()=0

  LEVEL 1 Unregisters the sparse point cloud listener.
- virtual royale::CameraStatus registerIRImageListener (royale::IIRImageListener *listener)=0

  LEVEL 1 Once registering the data listener, IR image data is sent via the callback function.
- virtual royale::CameraStatus unregisterIRImageListener ()=0

  LEVEL 1 Unregisters the IR image listener.
- virtual royale::CameraStatus registerDepthIRImageListener (royale::IDepthIRImageListener *listener)=0

LEVEL 1 Once registering the data listener, depth and IR image data is sent via the callback function.

- virtual royale::CameraStatus unregisterDepthIRImageListener ()=0

  LEVEL 1 Unregisters the DepthIR image listener.
- virtual royale::CameraStatus registerEventListener (royale::IEventListener *listener)=0

  LEVEL 1 Register listener for event notifications.
- virtual royale::CameraStatus unregisterEventListener ()=0

  LEVEL 1 Unregisters listener for event notifications.
- virtual royale::CameraStatus startCapture ()=0

  LEVEL 1 Starts the video capture mode (free-running), based on the specified operation mode.
- virtual royale::CameraStatus stopCapture ()=0

  LEVEL 1 Stops the video capturing mode.
- virtual royale::CameraStatus getMaxSensorWidth (uint16_t &maxSensorWidth) const =0

  LEVEL 1 Returns the maximal width supported by the camera device.
- virtual royale::CameraStatus getMaxSensorHeight (uint16_t &maxSensorHeight) const =0

  LEVEL 1 Returns the maximal height supported by the camera device.
- virtual royale::CameraStatus getLensParameters (royale::LensParameters &param) const =0
- virtual royale::CameraStatus isConnected (bool &connected) const =0

  LEVEL 1 Returns the information if a connection to the camera could be established.
- virtual royale::CameraStatus isCalibrated (bool &calibrated) const =0

  LEVEL 1 Returns the information if the camera module is calibrated.
- virtual royale::CameraStatus isCapturing (bool &capturing) const =0

  LEVEL 1 Returns the information if the camera is currently in capture mode.
- virtual royale::CameraStatus getAccessLevel (royale::CameraAccessLevel &accessLevel) const =0

  LEVEL 1 Returns the current camera device access level.
- virtual royale::CameraStatus startRecording (const royale::String &fileName, uint32_t numberOfFrames=0, uint32_t frameSkip=0, uint32_t msSkip=0)=0
- virtual royale::CameraStatus stopRecording ()=0

  LEVEL 1 Stop recording the raw data stream into a file.
- virtual royale::CameraStatus registerRecordListener (royale::IRecordStopListener *listener)=0
- virtual royale::CameraStatus unregisterRecordListener ()=0

  LEVEL 1 Unregisters the record listener.
- virtual royale::CameraStatus registerExposureListener (royale::IExposureListener *listener)=0

  LEVEL 1 [deprecated] Once registering the exposure listener, new exposure values calculated by the processing are sent to the listener.
- virtual royale::CameraStatus registerExposureListener (royale::IExposureListener2 *listener)=0

  LEVEL 1 Once registering the exposure listener, new exposure values calculated by the processing are sent to the listener.
- virtual royale::CameraStatus unregisterExposureListener ()=0

  LEVEL 1 Unregisters the exposure listener.
- virtual royale::CameraStatus setFrameRate (uint16_t framerate)=0

  LEVEL 1 Set the frame rate to a value.
- virtual royale::CameraStatus getFrameRate (uint16_t &frameRate) const =0

  LEVEL 1 Get the current frame rate which is set for the current use case.
- virtual royale::CameraStatus getMaxFrameRate (uint16_t &maxFrameRate) const =0

  LEVEL 1 Get the maximal frame rate which can be set for the current use case.
- virtual royale::CameraStatus setExternalTrigger (bool useExternalTrigger)=0

  LEVEL 1 Enable or disable the external triggering.
- virtual royale::CameraStatus setFilterLevel (const royale::FilterLevel level, royale::StreamId streamId=0)=0

  LEVEL 1 Change the level of filtering that is used during the processing.
- virtual royale::CameraStatus getFilterLevel (royale::FilterLevel &level, royale::StreamId streamId=0) const =0

LEVEL 1 Retrieve the level of filtering for the given streamId.

- virtual royale::CameraStatus getFilterLevels (royale::Vector< royale::FilterLevel > &levels, royale::StreamId streamId=0) const =0

LEVEL 1 Retrieve the levels of filtering that are available for the given streamId.

- virtual royale::CameraStatus getExposureGroups (royale::Vector< royale::String > &exposureGroups) const =0

LEVEL 2 Get the list of exposure groups supported by the currently set use case.

- virtual royale::CameraStatus setExposureTime (const String &exposureGroup, uint32_t exposureTime)=0

LEVEL 2 Change the exposure time for the supported operated operation modes.

- virtual royale::CameraStatus getExposureLimits (const String &exposureGroup, royale::Pair< uint32_t, uint32_t > &exposureLimits) const =0

LEVEL 2 Retrieves the minimum and maximum allowed exposure limits of the specified operation mode.

- virtual royale::CameraStatus setExposureTimes (const royale::Vector< uint32_t > &exposureTimes, royale::StreamId streamId=0)=0

LEVEL 2 Change the exposure times for all sequences.

- virtual royale::CameraStatus setExposureForGroups (const royale::Vector< uint32_t > &exposureTimes)=0

LEVEL 2 Change the exposure times for all exposure groups.

- virtual royale::CameraStatus setProcessingParameters (const royale::ProcessingParameterVector &parameters, uint16_t streamId=0)=0

LEVEL 2 Set/alter processing parameters in order to control the data output.

- virtual royale::CameraStatus getProcessingParameters (royale::ProcessingParameterVector &parameters, uint16_t streamId=0)=0

LEVEL 2 Retrieve the available processing parameters which are used for the calculation.

- virtual royale::CameraStatus registerDataListenerExtended (royale::IExtendedDataListener *listener)=0

LEVEL 2 After registering the extended data listener, extended data is sent via the callback function.

- virtual royale::CameraStatus unregisterDataListenerExtended ()=0

LEVEL 2 Unregisters the data extended listener.

- virtual royale::CameraStatus setCallbackData (royale::CallbackData cbData)=0

LEVEL 2 Set the callback output data type to one type only.

- virtual royale::CameraStatus setCallbackData (uint16_t cbData)=0

LEVEL 2 [deprecated] Set the callback output data type.

- virtual royale::CameraStatus setCalibrationData (const royale::String &filename)=0

LEVEL 2 Loads a different calibration from a file.

- virtual royale::CameraStatus setCalibrationData (const royale::Vector< uint8_t > &data)=0

LEVEL 2 Loads a different calibration from a given Vector.

- virtual royale::CameraStatus getCalibrationData (royale::Vector< uint8_t > &data)=0

LEVEL 2 Retrieves the current calibration data.

- virtual royale::CameraStatus writeCalibrationToFlash ()=0

LEVEL 2 Tries to write the current calibration file into the internal flash of the device.

- virtual royale::CameraStatus setProcessingThreads (uint32_t numThreads, royale::StreamId streamId=0)=0

LEVEL 2 Set number of threads to be used in the processing.

- virtual royale::CameraStatus writeDataToFlash (const royale::Vector< uint8_t > &data)=0

LEVEL 3 Writes an arbitrary vector of data on to the storage of the device.

- virtual royale::CameraStatus writeDataToFlash (const royale::String &filename)=0

LEVEL 3 Writes an arbitrary file to the storage of the device.

- virtual royale::CameraStatus setDutyCycle (double dutyCycle, uint16_t index)=0

LEVEL 3 Change the dutycycle of a certain sequence.

- virtual royale::CameraStatus writeRegisters (const royale::Vector< royale::Pair< royale::String, uint64_t >> &registers)=0

LEVEL 3 For each element of the vector a single register write is issued for the connected imager.

- virtual royale::CameraStatus readRegisters (royale::Vector< royale::Pair< royale::String, uint64_t >> &registers)=0

LEVEL 3 For each element of the vector a single register read is issued for the connected imager.

- virtual royale::CameraStatus shiftLensCenter (int16_t tx, int16_t ty)=0

  LEVEL 3 Shift the current lens center by the given translation.
- virtual royale::CameraStatus getLensCenter (uint16_t &x, uint16_t &y)=0

  LEVEL 3 Retrieves the current lens center.

8.6.1 Detailed Description

This is the main interface for talking to the time-of-flight camera system.

Typically, an instance is created by the CameraManager which automatically detects a connected module. The support access levels can be activated by entering the correct code during creation. After creation, the ICameraDevice is in ready state and can be initialized.

Please refer to the provided examples (in the samples directory) for an overview on how to use this class.

On Windows please ensure that the CameraDevice object is destroyed before the main() function exits, for example by storing the unique_ptr from CameraManager::createCamera in a unique_ptr that will go out of scope at (or before) the end of main(). Not destroying the CameraDevice before the exit can lead to a deadlock (https://stackoverflow.↩com/questions/10915233/stdthreadjoin-hangs-if-called-after-main-exits-when-using-vs2012-rc).

8.6.2 Constructor & Destructor Documentation

8.6.2.1 ~ICameraDevice()

```
virtual ~ICameraDevice ( )  [virtual]
```

8.6.3 Member Function Documentation

8.6.3.1 getAccessLevel()

```
virtual royale::CameraStatus getAccessLevel (
            royale::CameraAccessLevel & accessLevel ) const  [pure virtual]
```

LEVEL 1 Returns the current camera device access level.

8.6.3.2 getCalibrationData()

```
virtual royale::CameraStatus getCalibrationData (
            royale::Vector< uint8_t > & data )  [pure virtual]
```

LEVEL 2 Retrieves the current calibration data.

Parameters

data	Vector which will be filled with the calibration data

8.6.3.3 getCameraInfo()

```
virtual royale::CameraStatus getCameraInfo (
            royale::Vector< royale::Pair< royale::String, royale::String >> & camInfo )
const  [pure virtual]
```

Examples

sampleCameraInfo.cpp.

8.6.3.4 getCameraName()

```
virtual royale::CameraStatus getCameraName (
            royale::String & cameraName ) const  [pure virtual]
```

LEVEL 1 Returns the associated camera name as a string which is defined in the CoreConfig of each module.

Examples

sampleCameraInfo.cpp.

8.6.3.5 getCurrentUseCase()

```
virtual royale::CameraStatus getCurrentUseCase (
            royale::String & useCase ) const  [pure virtual]
```

LEVEL 1 Gets the current use case as string.

Parameters

useCase	current use case identified as string

8.6.3.6 getExposureGroups()

```
virtual royale::CameraStatus getExposureGroups (
            royale::Vector< royale::String > & exposureGroups ) const  [pure virtual]
```

LEVEL 2 Get the list of exposure groups supported by the currently set use case.

8.6.3.7 getExposureLimits() [1/2]

```
virtual royale::CameraStatus getExposureLimits (
            const String & exposureGroup,
            royale::Pair< uint32_t, uint32_t > & exposureLimits ) const  [pure virtual]
```

LEVEL 2 Retrieves the minimum and maximum allowed exposure limits of the specified operation mode.

Limits may vary between exposure groups. Can be used to retrieve the allowed operational range for a manual definition of the exposure time.

Parameters

exposureGroup	exposure group to be queried
exposureLimits	pair of (minimum, maximum) exposure time in microseconds

8.6.3.8 getExposureLimits() [2/2]

```
virtual royale::CameraStatus getExposureLimits (
            royale::Pair< uint32_t, uint32_t > & exposureLimits,
            royale::StreamId streamId = 0 ) const  [pure virtual]
```

LEVEL 1 Retrieves the minimum and maximum allowed exposure limits of the specified operation mode.

Can be used to retrieve the allowed operational range for a manual definition of the exposure time.

For mixed-mode usecases a valid streamId must be passed. For usecases having only one stream the default value of 0 (which is otherwise not a valid stream id) can be used to refer to that stream. This is for backward compatibility.

Parameters

exposureLimits	contains the limits on successful return
streamId	stream for which the exposure limits should be returned

8.6.3.9 getExposureMode()

```
virtual royale::CameraStatus getExposureMode (
            royale::ExposureMode & exposureMode,
            royale::StreamId streamId = 0 ) const  [pure virtual]
```

LEVEL 1 Retrieves the current mode of operation for acquisition of the exposure time.

For mixed-mode usecases a valid streamId must be passed. For usecases having only one stream the default value of 0 (which is otherwise not a valid stream id) can be used to refer to that stream. This is for backward compatibility.

Parameters

exposureMode	contains current exposure mode on successful return
streamId	stream for which the exposure mode should be returned

8.6.3.10 getFilterLevel()

```
virtual royale::CameraStatus getFilterLevel (
            royale::FilterLevel & level,
            royale::StreamId streamId = 0 ) const  [pure virtual]
```

LEVEL 1 Retrieve the level of filtering for the given streamId.

If the processing parameters do not match any of the levels this will return FilterLevel::Custom.

8.6.3.11 getFilterLevels()

```
virtual royale::CameraStatus getFilterLevels (
            royale::Vector< royale::FilterLevel > & levels,
            royale::StreamId streamId = 0 ) const  [pure virtual]
```

LEVEL 1 Retrieve the levels of filtering that are available for the given streamId.

8.6.3.12 getFrameRate()

```
virtual royale::CameraStatus getFrameRate (
            uint16_t & frameRate ) const  [pure virtual]
```

LEVEL 1 Get the current frame rate which is set for the current use case.

This function is not supported for mixed-mode.

8.6.3.13 getId()

```
virtual royale::CameraStatus getId (
            royale::String & id ) const  [pure virtual]
```

LEVEL 1 Get the ID of the camera device.

Parameters

id	String container in which the unique ID for the camera device will be written.

Returns

CameraStatus

Examples

sampleCameraInfo.cpp.

8.6.3.14 getLensCenter()

```
virtual royale::CameraStatus getLensCenter (
            uint16_t & x,
            uint16_t & y )  [pure virtual]
```

LEVEL 3 Retrieves the current lens center.

Parameters

x	current x center
y	current y center

8.6.3.15 getLensParameters()

```
virtual royale::CameraStatus getLensParameters (
            royale::LensParameters & param ) const  [pure virtual]
```

Examples

sampleCameraInfo.cpp, and sampleOpenCV.cpp.

8.6.3.16 getMaxFrameRate()

```
virtual royale::CameraStatus getMaxFrameRate (
            uint16_t & maxFrameRate ) const  [pure virtual]
```

LEVEL 1 Get the maximal frame rate which can be set for the current use case.

This function is not supported for mixed-mode.

8.6.3.17 getMaxSensorHeight()

```
virtual royale::CameraStatus getMaxSensorHeight (
            uint16_t & maxSensorHeight ) const  [pure virtual]
```

LEVEL 1 Returns the maximal height supported by the camera device.

Examples

sampleCameraInfo.cpp.

8.6.3.18 getMaxSensorWidth()

```
virtual royale::CameraStatus getMaxSensorWidth (
            uint16_t & maxSensorWidth ) const  [pure virtual]
```

LEVEL 1 Returns the maximal width supported by the camera device.

Examples

sampleCameraInfo.cpp.

8.6.3.19 getNumberOfStreams()

```
virtual royale::CameraStatus getNumberOfStreams (
            const royale::String & name,
            uint32_t & nrStreams ) const  [pure virtual]
```

LEVEL 1 Retrieves the number of streams for a specified use case.

Parameters

name	use case name
nrStreams	number of streams for the specified use case

Examples

sampleCameraInfo.cpp.

8.6.3.20 getProcessingParameters()

```
virtual royale::CameraStatus getProcessingParameters (
            royale::ProcessingParameterVector & parameters,
            uint16_t streamId = 0 )  [pure virtual]
```

LEVEL 2 Retrieve the available processing parameters which are used for the calculation.

Some parameters may only be available on some devices (and may depend on both the processing implementation and the calibration data available from the device), therefore the length of the vector may be less than ProcessingFlag::NUM_FLAGS.

8.6.3.21 getStreams()

```
virtual royale::CameraStatus getStreams (
            royale::Vector< royale::StreamId > & streams ) const  [pure virtual]
```

LEVEL 1 Get the streams associated with the current use case.

Examples

sampleRetrieveData.cpp.

8.6.3.22 getUseCases()

```
virtual royale::CameraStatus getUseCases (
            royale::Vector< royale::String > & useCases ) const  [pure virtual]
```

LEVEL 1 Returns all use cases which are supported by the connected module and valid for the current selected CallbackData information (e.g.

Raw, Depth, ...)

Examples

sampleCameraInfo.cpp, and sampleRetrieveData.cpp.

8.6.3.23 initialize() [1/2]

```
virtual royale::CameraStatus initialize ( )  [pure virtual]
```

LEVEL 1 Initializes the camera device and sets the first available use case.

All non-SUCCESS return statuses, except for DEVICE_ALREADY_INITIALIZED, indicate a non-recoverable error condition.

Returns

SUCCESS if the camera device has been set up correctly and the default use case has been be activated.

DEVICE_ALREADY_INITIALIZED if this is called more than once.

FILE_NOT_FOUND may be returned when this ICameraDevice represents a recording.

USECASE_NOT_SUPPORTED if the camera device was successfully opened, but the default use case could not be activated. This is only expected to happen when bringing up a new module, so it's not expected at Level 1.

CALIBRATION_DATA_ERROR if the camera device has no calibration data (or data that is incompatible with the processing, requiring a more recent version of Royale); this device can not be used with Level 1. For bringing up a new module, Level 2 access can either access the hardware by closing this instance, creating a new instance, and calling setCallback↩ Data (CallbackData::Raw) before calling initialize() or by specifying different calibration data by calling setCalibrationData (const royale::String &filename) before calling initialize().

Other non-SUCCESS values indicate that the device can not be used.

Examples

sampleCameraInfo.cpp, sampleExportPLY.cpp, sampleIReplay.cpp, sampleOpenCV.cpp, sampleRecordRRF.cpp, and sampleRetrieveData.cpp.

8.6.3.24 initialize() [2/2]

```
virtual royale::CameraStatus initialize (
            const royale::String & initUseCase )  [pure virtual]
```

LEVEL 1 Initialize the camera and configure the system for the specified use case.

See also initialize ().

Parameters

initUseCase	identifies the use case by a case sensitive string

8.6.3.25 isCalibrated()

```
virtual royale::CameraStatus isCalibrated (
            bool & calibrated ) const  [pure virtual]
```

LEVEL 1 Returns the information if the camera module is calibrated.

Older camera modules can still be operated with royale, but calibration data may be incomplete.

Parameters

calibrated	true if the module contains proper calibration data

8.6.3.26 isCapturing()

```
virtual royale::CameraStatus isCapturing (
            bool & capturing ) const  [pure virtual]
```

LEVEL 1 Returns the information if the camera is currently in capture mode.

Parameters

capturing	true if camera is in capture mode

8.6.3.27 isConnected()

```
virtual royale::CameraStatus isConnected (
            bool & connected ) const  [pure virtual]
```

LEVEL 1 Returns the information if a connection to the camera could be established.

Parameters

connected	true if properly set up

8.6.3.28 readRegisters()

```
virtual royale::CameraStatus readRegisters (
            royale::Vector< royale::Pair< royale::String, uint64_t >> & registers )  [pure
virtual]
```

LEVEL 3 For each element of the vector a single register read is issued for the connected imager.

The second element of each pair will be overwritten by the value of the register given by the first element of the pair :

```
Vector<Pair<String, uint64_t» registers;
registers.push_back (Pair<String, uint64_t> ("0x0B0AD", 0));
camera->readRegisters (registers);
```

will read out the register 0x0B0AD and will replace the 0 with the current value of the register.

Parameters

registers	Contains elements of possibly not-unique (String, uint64_t) duplets. The String component can consist of: a) a base-10 decimal number in the range of [0, 65535] b) a base-16 hexadecimal number preceded by a "0x" in the range of [0, 65535]

8.6.3.29 registerDataListener()

```
virtual royale::CameraStatus registerDataListener (
            royale::IDepthDataListener * listener )  [pure virtual]
```

Examples

sampleExportPLY.cpp, sampleIReplay.cpp, sampleOpenCV.cpp, and sampleRetrieveData.cpp.

8.6.3.30 registerDataListenerExtended()

```
virtual royale::CameraStatus registerDataListenerExtended (
            royale::IExtendedDataListener * listener )  [pure virtual]
```

LEVEL 2 After registering the extended data listener, extended data is sent via the callback function.

If depth data only is specified, this listener is not called. For this case, please use the standard depth data listener.

Parameters

listener	interface which needs to implement the callback method

8.6.3.31 registerDepthImageListener()

```
virtual royale::CameraStatus registerDepthImageListener (
            royale::IDepthImageListener * listener )  [pure virtual]
```

LEVEL 1 Once registering the data listener, Android depth image data is sent via the callback function.

Consider using registerDataListener and an IDepthDataListener instead of this listener. This callback provides only an array of depth and confidence values. The mapping of pixels to the scene is similar to the pixels of a two-dimensional camera, and it is unlikely to be a rectilinear projection (although this depends on the exact camera).

Parameters

listener	interface which needs to implement the callback method

8.6.3.32 registerDepthIRImageListener()

```
virtual royale::CameraStatus registerDepthIRImageListener (
            royale::IDepthIRImageListener * listener )  [pure virtual]
```

LEVEL 1 Once registering the data listener, depth and IR image data is sent via the callback function.

Parameters

listener	interface which needs to implement the callback method

8.6.3.33 registerEventListener()

```
virtual royale::CameraStatus registerEventListener (
            royale::IEventListener * listener )  [pure virtual]
```

LEVEL 1 Register listener for event notifications.

The callback will be invoked asynchronously. Events include things like illumination unit overtemperature.

8.6.3.34 registerExposureListener() [1/2]

```
virtual royale::CameraStatus registerExposureListener (
            royale::IExposureListener * listener )  [pure virtual]
```

LEVEL 1 [deprecated] Once registering the exposure listener, new exposure values calculated by the processing are sent to the listener.

As this listener doesn't support streams, only updates for the first stream will be sent.

Only one exposure listener is supported at a time, calling this will automatically unregister any previously registered IExposureListener or IExposureListener2.

Parameters

listener	interface which needs to implement the callback method

8.6.3.35 registerExposureListener() [2/2]

```
virtual royale::CameraStatus registerExposureListener (
            royale::IExposureListener2 * listener )  [pure virtual]
```

LEVEL 1 Once registering the exposure listener, new exposure values calculated by the processing are sent to the listener.

Only one exposure listener is supported at a time, calling this will automatically unregister any previously registered IExposureListener or IExposureListener2.

Parameters

listener	interface which needs to implement the callback method

8.6.3.36 registerIRImageListener()

```
virtual royale::CameraStatus registerIRImageListener (
            royale::IIRImageListener * listener )  [pure virtual]
```

LEVEL 1 Once registering the data listener, IR image data is sent via the callback function.

Parameters

listener	interface which needs to implement the callback method

8.6.3.37 registerRecordListener()

```
virtual royale::CameraStatus registerRecordListener (
            royale::IRecordStopListener * listener )  [pure virtual]
```

Examples

sampleRecordRRF.cpp.

8.6.3.38 registerSparsePointCloudListener()

```
virtual royale::CameraStatus registerSparsePointCloudListener (
            royale::ISparsePointCloudListener * listener )  [pure virtual]
```

LEVEL 1 Once registering the data listener, Android point cloud data is sent via the callback function.

Parameters

listener	interface which needs to implement the callback method

8.6.3.39 setCalibrationData() [1/2]

```
virtual royale::CameraStatus setCalibrationData (
            const royale::String & filename )  [pure virtual]
```

LEVEL 2 Loads a different calibration from a file.

This calibration data will also be used by the processing!

Parameters

filename	name of the calibration file which should be loaded

Returns

CameraStatus

8.6.3.40 setCalibrationData() [2/2]

```
virtual royale::CameraStatus setCalibrationData (
            const royale::Vector< uint8_t > & data )  [pure virtual]
```

LEVEL 2 Loads a different calibration from a given Vector.

This calibration data will also be used by the processing!

Parameters

data	calibration data which should be used

Returns

CameraStatus

8.6.3.41 setCallbackData() [1/2]

```
virtual royale::CameraStatus setCallbackData (
            royale::CallbackData cbData )  [pure virtual]
```

LEVEL 2 Set the callback output data type to one type only.

INFO: This method needs to be called before startCapture(). If is is called while the camera is in capture mode, it will only have effect after the next stop/start sequence.

8.6.3.42 setCallbackData() [2/2]

```
virtual royale::CameraStatus setCallbackData (
            uint16_t cbData )  [pure virtual]
```

LEVEL 2 [deprecated] Set the callback output data type.

Setting multiple types currently isn't supported.

INFO: This method needs to be called before startCapture(). If is is called while the camera is in capture mode, it will only have effect after the next stop/start sequence.

8.6.3.43 setDutyCycle()

```
virtual royale::CameraStatus setDutyCycle (
            double dutyCycle,
            uint16_t index )  [pure virtual]
```

LEVEL 3 Change the dutycycle of a certain sequence.

If the dutycycle is not supported, an error will be returned. The dutycycle can also be altered during capture mode.

Parameters

dutyCycle	dutyCycle in percent (0, 100)
index	index of the sequence to change

8.6.3.44 setExposureForGroups()

```
virtual royale::CameraStatus setExposureForGroups (
            const royale::Vector< uint32_t > & exposureTimes )  [pure virtual]
```

LEVEL 2 Change the exposure times for all exposure groups.

The order of the exposure times is aligned with the order of exposure groups received by getExposureGroups. If the vector that is provided is too long the extraneous values will be discard. If the vector is too short an error will be returned.

Parameters

exposureTimes	vector with exposure times in microseconds

8.6.3.45 setExposureMode()

```
virtual royale::CameraStatus setExposureMode (
            royale::ExposureMode exposureMode,
            royale::StreamId streamId = 0 )  [pure virtual]
```

LEVEL 1 Change the exposure mode for the supported operated operation modes.

For mixed-mode use cases a valid streamId must be passed. For use cases having only one stream the default value of 0 (which is otherwise not a valid stream id) can be used to refer to that stream. This is for backward compatibility.

If MANUAL exposure mode of operation is chosen, the user is able to determine set exposure time manually within the boundaries of the exposure limits of the specific operation mode.

In AUTOMATIC mode the optimum exposure settings are determined the system itself.

The default value is MANUAL.

Parameters

exposureMode	mode of operation to determine the exposure time
streamId	which stream to change exposure mode for

8.6.3.46 setExposureTime() [1/2]

```
virtual royale::CameraStatus setExposureTime (
            const String & exposureGroup,
            uint32_t exposureTime )  [pure virtual]
```

LEVEL 2 Change the exposure time for the supported operated operation modes.

If MANUAL exposure mode of operation is chosen, the user is able to determine set exposure time manually within the boundaries of the exposure limits of the specific operation mode. On success the corresponding status message is returned. In any other mode of operation the method will return EXPOSURE_MODE_INVALID to indicate incompliance with the selected exposure mode. If the camera is used in the playback configuration a LOGIC_ERROR is returned instead.

Parameters

exposureGroup	exposure group to be updated
exposureTime	exposure time in microseconds

8.6.3.47 setExposureTime() [2/2]

```
virtual royale::CameraStatus setExposureTime (
            uint32_t exposureTime,
            royale::StreamId streamId = 0 )  [pure virtual]
```

LEVEL 1 Change the exposure time for the supported operated operation modes.

For mixed-mode use cases a valid streamId must be passed. For use cases having only one stream the default value of 0 (which is otherwise not a valid stream id) can be used to refer to that stream. This is for backward compatibility.

If MANUAL exposure mode of operation is chosen, the user is able to determine set exposure time manually within the boundaries of the exposure limits of the specific operation mode.

On success the corresponding status message is returned. In any other mode of operation the method will return EXPOSURE_MODE_INVALID to indicate non-compliance with the selected exposure mode. If the camera is used in the playback configuration a LOGIC_ERROR is returned instead.

WARNING : If this function is used on Level 3 it will ignore the limits given by the use case.

Parameters

exposureTime	exposure time in microseconds
streamId	which stream to change exposure for

Examples

sampleRetrieveData.cpp.

8.6.3.48 setExposureTimes()

```
virtual royale::CameraStatus setExposureTimes (
            const royale::Vector< uint32_t > & exposureTimes,
            royale::StreamId streamId = 0 )  [pure virtual]
```

LEVEL 2 Change the exposure times for all sequences.

As it is possible to reuse an exposure group for different sequences it can happen that the exposure group is updated multiple times! If the vector that is provided is too long the extraneous values will be discard. If the vector is too short an error will be returned.

WARNING : If this function is used on Level 3 it will ignore the limits given by the use case.

Parameters

exposureTimes	vector with exposure times in microseconds
streamId	which stream to change exposure times for

8.6.3.49 setExternalTrigger()

```
virtual royale::CameraStatus setExternalTrigger (
            bool useExternalTrigger )  [pure virtual]
```

LEVEL 1 Enable or disable the external triggering.

Some camera modules support an external trigger, they can capture images synchronized with another device. If the hardware you are using supports it, calling setExternalTrigger(true) will make the camera capture images in this way. The call to setExternalTrigger has to be done before initializing the device.

The external signal must not exceed the maximum FPS of the chosen UseCase, but lower frame rates are supported. If no external signal is received, the imager will not start delivering images.

For information if your camera module supports external triggering and how to use it please refer to the Getting Started Guide of your camera. If the module doesn't support triggering calling this function will return a LOGIC_ERROR.

Royale currently expects a trigger pulse, not a constant trigger signal. Using a constant trigger signal might lead to a wrong framerate!

8.6.3.50 setFilterLevel()

```
virtual royale::CameraStatus setFilterLevel (
            const royale::FilterLevel level,
            royale::StreamId streamId = 0 )  [pure virtual]
```

LEVEL 1 Change the level of filtering that is used during the processing.

This will change the setting of multiple internal filters based on some predefined levels. FilterLevel::Custom is a special setting which can not be set.

8.6.3.51 setFrameRate()

```
virtual royale::CameraStatus setFrameRate (
            uint16_t framerate )  [pure virtual]
```

LEVEL 1 Set the frame rate to a value.

Upper bound is given by the use case. E.g. Use case with 5 FPS, a maximum frame rate of 5 and a minimum of 1 can be set. Setting a frame rate of 0 is not allowed.

The framerate is specific for the current use case. This function is not supported for mixed-mode.

8.6.3.52 setProcessingParameters()

```
virtual royale::CameraStatus setProcessingParameters (
            const royale::ProcessingParameterVector & parameters,
            uint16_t streamId = 0 )  [pure virtual]
```

LEVEL 2 Set/alter processing parameters in order to control the data output.

A list of processing flags is available as an enumeration. The `Variant` data type can take float, int, or bool. Please make sure to set the proper `Variant` type for the enum.

8.6.3.53 setProcessingThreads()

```
virtual royale::CameraStatus setProcessingThreads (
            uint32_t numThreads,
            royale::StreamId streamId = 0 )  [pure virtual]
```

LEVEL 2 Set number of threads to be used in the processing.

Parameters

numThreads	Numbers of threads to be set
streamId	The ID of the current stream

8.6.3.54 setUseCase()

```
virtual royale::CameraStatus setUseCase (
            const royale::String & name )  [pure virtual]
```

LEVEL 1 Sets the use case for the camera.

If the use case is supported by the connected camera device SUCCESS will be returned. Changing the use case will also change the processing parameters that are used (e.g. auto exposure)!

NOTICE: This function must not be called in the data callback - the behavior is undefined. Call it from a different thread instead.

Parameters

name	identifies the use case by an case sensitive string

Returns

SUCCESS if use case can be set

Examples

sampleRetrieveData.cpp.

8.6.3.55 shiftLensCenter()

```
virtual royale::CameraStatus shiftLensCenter (
            int16_t tx,
            int16_t ty )  [pure virtual]
```

LEVEL 3 Shift the current lens center by the given translation.

This works cumulatively (calling shiftLensCenter (0, 1) three times in a row has the same effect as calling shiftLensCenter (0, 3)). If the resulting lens center is not valid this function will return an error. This function works only for raw data readout.

Parameters

tx	translation in x direction
ty	translation in y direction

8.6.3.56 startCapture()

```
virtual royale::CameraStatus startCapture ( )  [pure virtual]
```

LEVEL 1 Starts the video capture mode (free-running), based on the specified operation mode.

A listener needs to be registered in order to retrieve the data stream. Either raw data or processed data can be consumed. If no data listener is registered an error will be returned and capturing is not started.

Examples

sampleExportPLY.cpp, sampleIReplay.cpp, sampleOpenCV.cpp, sampleRecordRRF.cpp, and sampleRetrieveData.cpp.

8.6.3.57 startRecording()

```
virtual royale::CameraStatus startRecording (
            const royale::String & fileName,
            uint32_t numberOfFrames = 0,
            uint32_t frameSkip = 0,
            uint32_t msSkip = 0 )  [pure virtual]
```

Examples

sampleRecordRRF.cpp.

8.6.3.58 stopCapture()

```
virtual royale::CameraStatus stopCapture ( )  [pure virtual]
```

LEVEL 1 Stops the video capturing mode.

All buffers should be released again by the data listener.

Examples

sampleExportPLY.cpp, sampleIReplay.cpp, sampleOpenCV.cpp, sampleRecordRRF.cpp, and sampleRetrieveData.cpp.

8.6.3.59 stopRecording()

```
virtual royale::CameraStatus stopRecording ( )  [pure virtual]
```

LEVEL 1 Stop recording the raw data stream into a file.

After the recording is stopped the file is available on the file system.

Examples

sampleRecordRRF.cpp.

8.6.3.60 unregisterDataListener()

```
virtual royale::CameraStatus unregisterDataListener ( )  [pure virtual]
```

LEVEL 1 Unregisters the data depth listener.

It's not necessary to unregister this listener (or any other listener) before deleting the ICameraDevice.

8.6.3.61 unregisterDataListenerExtended()

```
virtual royale::CameraStatus unregisterDataListenerExtended ( )  [pure virtual]
```

LEVEL 2 Unregisters the data extended listener.

It's not necessary to unregister this listener (or any other listener) before deleting the ICameraDevice.

8.6.3.62 unregisterDepthImageListener()

```
virtual royale::CameraStatus unregisterDepthImageListener ( )  [pure virtual]
```

LEVEL 1 Unregisters the depth image listener.

It's not necessary to unregister this listener (or any other listener) before deleting the ICameraDevice.

8.6.3.63 unregisterDepthIRImageListener()

```
virtual royale::CameraStatus unregisterDepthIRImageListener ( )  [pure virtual]
```

LEVEL 1 Unregisters the DepthIR image listener.

It's not necessary to unregister this listener (or any other listener) before deleting the ICameraDevice.

8.6.3.64 unregisterEventListener()

```
virtual royale::CameraStatus unregisterEventListener ( )  [pure virtual]
```

LEVEL 1 Unregisters listener for event notifications.

It's not necessary to unregister this listener (or any other listener) before deleting the ICameraDevice.

8.6.3.65 unregisterExposureListener()

```
virtual royale::CameraStatus unregisterExposureListener ( )  [pure virtual]
```

LEVEL 1 Unregisters the exposure listener.

It's not necessary to unregister this listener (or any other listener) before deleting the ICameraDevice.

8.6.3.66 unregisterIRImageListener()

```
virtual royale::CameraStatus unregisterIRImageListener ( )  [pure virtual]
```

LEVEL 1 Unregisters the IR image listener.

It's not necessary to unregister this listener (or any other listener) before deleting the ICameraDevice.

8.6.3.67 unregisterRecordListener()

```
virtual royale::CameraStatus unregisterRecordListener ( )  [pure virtual]
```

LEVEL 1 Unregisters the record listener.

It's not necessary to unregister this listener (or any other listener) before deleting the ICameraDevice.

8.6.3.68 unregisterSparsePointCloudListener()

```
virtual royale::CameraStatus unregisterSparsePointCloudListener ( )  [pure virtual]
```

LEVEL 1 Unregisters the sparse point cloud listener.

It's not necessary to unregister this listener (or any other listener) before deleting the ICameraDevice.

8.6.3.69 writeCalibrationToFlash()

```
virtual royale::CameraStatus writeCalibrationToFlash ( )  [pure virtual]
```

LEVEL 2 Tries to write the current calibration file into the internal flash of the device.

If no flash is found RESOURCE_ERROR is returned. If there are errors during the flash process it will try to restore the original calibration.

This is not yet implemented for all cameras!

Some devices also store other data in the calibration data area, for example the product identifier. This L2 method will only change the calibration data, and will preserve the other data; if an unsupported combination of existing data and new data is encountered it will return an error without writing to the storage. Only the L3 methods can change or remove the additional data.

Returns

CameraStatus

8.6.3.70 writeDataToFlash() [1/2]

```
virtual royale::CameraStatus writeDataToFlash (
            const royale::String & filename )  [pure virtual]
```

LEVEL 3 Writes an arbitrary file to the storage of the device.

If no flash is found RESOURCE_ERROR is returned.

Where the data will be written to is implementation defined. After using this function, the eye safety of the device is not guaranteed, even after reopening the device with L1 access. This method may overwrite the product identifier, and potentially even firmware in the device.

Parameters

filename	name of the file that should be flashed

8.6.3.71 writeDataToFlash() [2/2]

```
virtual royale::CameraStatus writeDataToFlash (
            const royale::Vector< uint8_t > & data )  [pure virtual]
```

LEVEL 3 Writes an arbitrary vector of data on to the storage of the device.

If no flash is found RESOURCE_ERROR is returned.

Where the data will be written to is implementation defined. After using this function, the eye safety of the device is not guaranteed, even after reopening the device with L1 access. This method may overwrite the product identifier, and potentially even firmware in the device.

Parameters

data	data that should be flashed

8.6.3.72 writeRegisters()

```
virtual royale::CameraStatus writeRegisters (
            const royale::Vector< royale::Pair< royale::String, uint64_t >> & registers )
[pure virtual]
```

LEVEL 3 For each element of the vector a single register write is issued for the connected imager.

Please be aware that any writes that will change crucial parts (starting the imager, stopping the imager, changing the ROI, ...) will not be reflected internally by Royale and might crash the program!

If this function is used on Level 4 (empty imager), please be aware that Royale will not start/stop the imager!

USE AT YOUR OWN RISK!!!

Parameters

registers	Contains elements of possibly not-unique (String, uint64_t) duplets. The String component can consist of: a) a base-10 decimal number in the range of [0, 65535] b) a base-16 hexadecimal number preceded by a "0x" in the range of [0, 65535]

The documentation for this class was generated from the following file:

- ICameraDevice.hpp

8.7 IDepthDataListener Class Reference

Provides the listener interface for consuming depth data from Royale.

`#include <IDepthDataListener.hpp>`

Public Member Functions

- virtual ~IDepthDataListener ()
- virtual void onNewData (const royale::DepthData *data)=0

  Will be called on every frame update by the Royale framework.

8.7.1 Detailed Description

Provides the listener interface for consuming depth data from Royale.

A listener needs to implement this interface and register itself as a listener to the ICameraDevice.

Examples

sampleExportPLY.cpp, sampleIReplay.cpp, sampleOpenCV.cpp, and sampleRetrieveData.cpp.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 ~IDepthDataListener()

```
virtual ~IDepthDataListener ( )  [inline], [virtual]
```

8.7.3 Member Function Documentation

8.7.3.1 onNewData()

```
virtual void onNewData (
            const royale::DepthData * data )  [pure virtual]
```

Will be called on every frame update by the Royale framework.

NOTICE: Calling other framework functions within the data callback can lead to undefined behavior and is therefore unsupported. Call these framework functions from another thread to avoid problems.

Examples

sampleIReplay.cpp.

The documentation for this class was generated from the following file:

- IDepthDataListener.hpp

8.8 IDepthImageListener Class Reference

Provides a listener interface for consuming depth images from Royale.

```
#include <IDepthImageListener.hpp>
```

Public Member Functions

- virtual ~IDepthImageListener ()
- virtual void onNewData (const royale::DepthImage *data)=0

  Will be called on every frame update by the Royale framework.

8.8.1 Detailed Description

Provides a listener interface for consuming depth images from Royale.

A listener needs to implement this interface and register itself as a listener to the ICameraDevice.

Consider using an IDepthDataListener instead of this listener. This callback provides only an array of depth and confidence values. The mapping of pixels to the scene is similar to the pixels of a two-dimensional camera, and it is unlikely to be a rectilinear projection (although this depends on the exact camera).

8.8.2 Constructor & Destructor Documentation

8.8.2.1 ~IDepthImageListener()

```
virtual ~IDepthImageListener ( )  [inline], [virtual]
```

8.8.3 Member Function Documentation

8.8.3.1 onNewData()

```
virtual void onNewData (
            const royale::DepthImage * data )  [pure virtual]
```

Will be called on every frame update by the Royale framework.

NOTICE: Calling other framework functions within the data callback can lead to undefined behavior and is therefore unsupported. Call these framework functions from another thread to avoid problems.

The documentation for this class was generated from the following file:

- IDepthImageListener.hpp

8.9 IDepthIRImageListener Class Reference

Provides a combined listener interface for consuming both depth and IR images from Royale.

```
#include <IDepthIRImageListener.hpp>
```

Public Member Functions

- virtual ~IDepthIRImageListener ()
- virtual void onNewData (const royale::DepthIRImage *data)=0

  Will be called on every frame update by the Royale framework.

8.9.1 Detailed Description

Provides a combined listener interface for consuming both depth and IR images from Royale.

A listener needs to implement this interface and register itself as a listener to the ICameraDevice.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 ~IDepthIRImageListener()

```
virtual ~IDepthIRImageListener ( )  [inline], [virtual]
```

8.9.3 Member Function Documentation

8.9.3.1 onNewData()

```
virtual void onNewData (
            const royale::DepthIRImage * data )  [pure virtual]
```

Will be called on every frame update by the Royale framework.

NOTICE: Calling other framework functions within the data callback can lead to undefined behavior and is therefore unsupported. Call these framework functions from another thread to avoid problems.

The documentation for this class was generated from the following file:

- IDepthIRImageListener.hpp

8.10 IEvent Class Reference

Interface for anything to be passed via IEventListener.

`#include <IEvent.hpp>`

Public Member Functions

- virtual ~IEvent ()
- virtual royale::EventSeverity severity () const =0

  Get the severity of this event.
- virtual const royale::String describe () const =0

  Returns debugging information intended for developers using the Royale API.
- virtual royale::EventType type () const =0

  Get the type of this event.

8.10.1 Detailed Description

Interface for anything to be passed via IEventListener.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 ~IEvent()

`virtual ~IEvent ( ) [virtual]`

8.10.3 Member Function Documentation

8.10.3.1 describe()

```
virtual const royale::String describe ( ) const  [pure virtual]
```

Returns debugging information intended for developers using the Royale API.

The strings returned may change between releases, and are unlikely to be localised, so are neither intended to be parsed automatically, nor intended to be shown to end users.

Returns

the description of this event.

8.10.3.2 severity()

```
virtual royale::EventSeverity severity ( ) const  [pure virtual]
```

Get the severity of this event.

The severity of an event denotes the level of urgency the event has. The severity may be used to determine when and where an event description should be presented.

Returns

the severity of this event.

8.10.3.3 type()

```
virtual royale::EventType type ( ) const  [pure virtual]
```

Get the type of this event.

Returns

the type of this event.

The documentation for this class was generated from the following file:

- IEvent.hpp

8.11 IEventListener Class Reference

This interface allows observers to receive events.

`#include <IEventListener.hpp>`

Public Member Functions

- virtual ROYALE_API ~IEventListener ()
- virtual ROYALE_API void onEvent (std::unique_ptr< royale::IEvent > &&event)=0
  Will be called when an event occurs.

8.11.1 Detailed Description

This interface allows observers to receive events.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 ~IEventListener()

```
virtual ROYALE_API ~IEventListener ( )  [virtual]
```

8.11.3 Member Function Documentation

8.11.3.1 onEvent()

```
virtual ROYALE_API void onEvent (
            std::unique_ptr< royale::IEvent > && event )  [pure virtual]
```

Will be called when an event occurs.

Note there are some constraints on what the user is allowed to do in the callback.

- Actually the royale API does not claim to be reentrant (and probably isn't), so the user is not supposed to call any API function from this callback besides stopCapture
- Deleting the ICameraDevice from the callback will most certainly lead to a deadlock. This has the interesting side effect that calling exit() or equivalent from the callback may cause issues.

Parameters

event	The event.

The documentation for this class was generated from the following file:

- IEventListener.hpp

8.12 IExposureListener Class Reference

Provides the listener interface for handling auto-exposure updates in royale.

`#include <IExposureListener.hpp>`

Public Member Functions

- virtual ~IExposureListener ()
- virtual void onNewExposure (const uint32_t exposureTime)=0

  Will be called when the newly calculated exposure time deviates from currently set exposure time of the current UseCase.

8.12.1 Detailed Description

Provides the listener interface for handling auto-exposure updates in royale.

To be notified of changes to the exposure, for example to update a UI slider, an application may implement this interface and register itself as a listener to the ICameraDevice. If the application merely wishes to use auto-exposure but does not need to know that the exposure has changed, it is not necessary to implement this listener.

The exposure will be changed for future captures, but there may be another capture before the new values take effect. An application that needs the values for a specific set of captured frames should use the metadata provided as part of the capture callback, for example in DepthData::exposureTimes.

8.12.2 Constructor & Destructor Documentation

8.12.2.1 ~IExposureListener()

`virtual ~IExposureListener ( ) [inline], [virtual]`

8.12.3 Member Function Documentation

8.12.3.1 onNewExposure()

```
virtual void onNewExposure (
            const uint32_t exposureTime )  [pure virtual]
```

Will be called when the newly calculated exposure time deviates from currently set exposure time of the current UseCase.

Parameters

exposureTime	Newly calculated exposure time

The documentation for this class was generated from the following file:

- IExposureListener.hpp

8.13 IExposureListener2 Class Reference

Provides the listener interface for handling auto-exposure updates in royale.

```
#include <IExposureListener2.hpp>
```

Public Member Functions

- virtual ROYALE_API ~IExposureListener2 ()
- virtual ROYALE_API void onNewExposure (const uint32_t exposureTime, const royale::StreamId streamId)=0

  Will be called when the newly calculated exposure time deviates from currently set exposure time of the current UseCase.

8.13.1 Detailed Description

Provides the listener interface for handling auto-exposure updates in royale.

To be notified of changes to the exposure, for example to update a UI slider, an application may implement this interface and register itself as a listener to the ICameraDevice. If the application merely wishes to use auto-exposure but does not need to know that the exposure has changed, it is not necessary to implement this listener.

The exposure will be changed for future captures, but there may be another capture before the new values take effect. An application that needs the values for a specific set of captured frames should use the metadata provided as part of the capture callback, for example in DepthData::exposureTimes.

8.13.2 Constructor & Destructor Documentation

8.13.2.1 ~IExposureListener2()

```
virtual ROYALE_API ~IExposureListener2 ( )  [virtual]
```

8.13.3 Member Function Documentation

8.13.3.1 onNewExposure()

```
virtual ROYALE_API void onNewExposure (
            const uint32_t exposureTime,
            const royale::StreamId streamId )  [pure virtual]
```

Will be called when the newly calculated exposure time deviates from currently set exposure time of the current UseCase.

Parameters

exposureTime	Newly calculated exposure time
streamId	Current stream identifier

The documentation for this class was generated from the following file:

- IExposureListener2.hpp

8.14 IExtendedData Class Reference

Interface for getting additional data to the standard depth data.

```
#include <IExtendedData.hpp>
```

Public Member Functions

- virtual ~IExtendedData ()
- virtual bool hasDepthData () const =0

  Indicates if the getDepthData() has valid data.
- virtual bool hasRawData () const =0

  Indicates if the getRawData() has valid data.
- virtual bool hasIntermediateData () const =0

  Indicates if the getIntermediateData() has valid data.
- virtual const royale::RawData * getRawData () const =0

  Returns the RawData structure.
- virtual const royale::DepthData * getDepthData () const =0

  Returns the DepthData structure.
- virtual const royale::IntermediateData * getIntermediateData () const =0

  Returns the IntermediateData structure.

8.14.1 Detailed Description

Interface for getting additional data to the standard depth data.

The retrieval of this data requires L2 access. Please be aware that not all data is filled. Therefore, use the has* calls to check if data is provided.

8.14.2 Constructor & Destructor Documentation

8.14.2.1 ~IExtendedData()

```
virtual ~IExtendedData ( )  [virtual]
```

8.14.3 Member Function Documentation

8.14.3.1 getDepthData()

```
virtual const royale::DepthData* getDepthData ( ) const  [pure virtual]
```

Returns the DepthData structure.

Returns

instance of DepthData if available, nullptr else

8.14.3.2 getIntermediateData()

```
virtual const royale::IntermediateData* getIntermediateData ( ) const  [pure virtual]
```

Returns the IntermediateData structure.

Returns

instance of IntermediateData if available, nullptr else

8.14.3.3 getRawData()

```
virtual const royale::RawData* getRawData ( ) const  [pure virtual]
```

Returns the RawData structure.

Returns

instance of RawData if available, nullptr else

8.14.3.4 hasDepthData()

```
virtual bool hasDepthData ( ) const  [pure virtual]
```

Indicates if the getDepthData() has valid data.

If false, then the getDepthData() will return nullptr.

8.14.3.5 hasIntermediateData()

```
virtual bool hasIntermediateData ( ) const  [pure virtual]
```

Indicates if the getIntermediateData() has valid data.

If false, then the getIntermediateData() will return nullptr.

8.14.3.6 hasRawData()

```
virtual bool hasRawData ( ) const  [pure virtual]
```

Indicates if the getRawData() has valid data.

If false, then the getRawData() will return nullptr.

The documentation for this class was generated from the following file:

- IExtendedData.hpp

8.15 IExtendedDataListener Class Reference

```
#include <IExtendedDataListener.hpp>
```

Public Member Functions

- virtual ~IExtendedDataListener ()=default
- virtual void onNewData (const royale::IExtendedData *data)=0
  Callback which is getting called by the ICameraDevice.

8.15.1 Constructor & Destructor Documentation

8.15.1.1 ~IExtendedDataListener()

```
virtual ~IExtendedDataListener ( )  [virtual], [default]
```

8.15.2 Member Function Documentation

8.15.2.1 onNewData()

```
virtual void onNewData (
            const royale::IExtendedData * data )  [pure virtual]
```

Callback which is getting called by the ICameraDevice.

If the data is required after this call, please copy away the data, the memory block will be reused.

NOTICE: Calling other framework functions within the data callback can lead to undefined behavior and is therefore unsupported. Call these framework functions from another thread to avoid problems.

Parameters

data	pointer to the underlying raw frames containing pointers to the raw frames

The documentation for this class was generated from the following file:

- IExtendedDataListener.hpp

8.16 IIRImageListener Class Reference

Provides the listener interface for consuming infrared images from Royale.

`#include <IIRImageListener.hpp>`

Public Member Functions

- virtual ~IIRImageListener ()
- virtual void onNewData (const royale::IRImage *data)=0

  Will be called on every frame update by the Royale framework.

8.16.1 Detailed Description

Provides the listener interface for consuming infrared images from Royale.

A listener needs to implement this interface and register itself as a listener to the ICameraDevice.

8.16.2 Constructor & Destructor Documentation

8.16.2.1 ~IIRImageListener()

`virtual ~IIRImageListener ( )  [inline], [virtual]`

8.16.3 Member Function Documentation

8.16.3.1 onNewData()

```
virtual void onNewData (
             const royale::IRImage * data )  [pure virtual]
```

Will be called on every frame update by the Royale framework.

NOTICE: Calling other framework functions within the data callback can lead to undefined behavior and is therefore unsupported. Call these framework functions from another thread to avoid problems.

The documentation for this class was generated from the following file:

- IIRImageListener.hpp

8.17 IntermediateData Struct Reference

This structure defines the Intermediate depth data which is delivered through the callback if the user has access level 2 for the CameraDevice.

```
#include <IntermediateData.hpp>
```

Public Member Functions

- IntermediateData & operator= (const IntermediateData &dd)

Public Attributes

- int version
  version number of the data format
- std::chrono::microseconds timeStamp
  timestamp in microseconds precision (time since epoch 1970)
- StreamId streamId
  stream which produced the data
- uint16_t width
  width of distance image
- uint16_t height
  height of distance image
- royale::Vector< royale::IntermediatePoint > points
  array of intermediate points
- royale::Vector< uint32_t > modulationFrequencies
  modulation frequencies for each sequence
- royale::Vector< uint32_t > exposureTimes
  integration times for each sequence
- uint32_t numFrequencies
  number of processed frequencies
- ProcessingParameterMap processingParameters
  processing Parameters used

8.17.1 Detailed Description

This structure defines the Intermediate depth data which is delivered through the callback if the user has access level 2 for the CameraDevice.

8.17.2 Member Function Documentation

8.17.2.1 operator=()

```
IntermediateData& operator= (
            const IntermediateData & dd )  [inline]
```

8.17.3 Member Data Documentation

8.17.3.1 exposureTimes

```
royale::Vector<uint32_t> exposureTimes
```

integration times for each sequence

8.17.3.2 height

```
uint16_t height
```

height of distance image

8.17.3.3 modulationFrequencies

```
royale::Vector<uint32_t> modulationFrequencies
```

modulation frequencies for each sequence

8.17.3.4 numFrequencies

`uint32_t numFrequencies`

number of processed frequencies

8.17.3.5 points

`royale::Vector<royale::IntermediatePoint> points`

array of intermediate points

8.17.3.6 processingParameters

`ProcessingParameterMap processingParameters`

processing Parameters used

8.17.3.7 streamId

`StreamId streamId`

stream which produced the data

8.17.3.8 timeStamp

`std::chrono::microseconds timeStamp`

timestamp in microseconds precision (time since epoch 1970)

8.17.3.9 version

`int version`

version number of the data format

8.17.3.10 width

`uint16_t width`

width of distance image

The documentation for this struct was generated from the following file:

- IntermediateData.hpp

8.18 IntermediatePoint Struct Reference

In addition to the standard depth point, the intermediate point also stores information which is calculated as temporaries in the processing pipeline.

`#include <IntermediateData.hpp>`

Public Attributes

- float distance
  radial distance of the current pixel
- float amplitude
  amplitude value of the current pixel
- float intensity
  intensity value of the current pixel
- uint32_t flags
  flag value of the current pixel

8.18.1 Detailed Description

In addition to the standard depth point, the intermediate point also stores information which is calculated as temporaries in the processing pipeline.

Distance : Radial distance for each point (in meter) Amplitude : Grayscale image that also provides a hint on the amount of reflected light. The values are positive, but the range depends on the camera that is used. Intensity : Intensity image (values can be negative in some cases) Flags : Flag image that shows invalid pixels. For a description of the flags please refer to the documentation you receive after getting level 2 access from pmd.

8.18.2 Member Data Documentation

8.18.2.1 amplitude

```
float amplitude
```

amplitude value of the current pixel

8.18.2.2 distance

```
float distance
```

radial distance of the current pixel

8.18.2.3 flags

```
uint32_t flags
```

flag value of the current pixel

8.18.2.4 intensity

```
float intensity
```

intensity value of the current pixel

The documentation for this struct was generated from the following file:

- IntermediateData.hpp

8.19 Variant::InvalidType Struct Reference

This will be thrown if a wrong type is used.

`#include <Variant.hpp>`

8.19.1 Detailed Description

This will be thrown if a wrong type is used.

The documentation for this struct was generated from the following file:

- Variant.hpp

8.20 IPlaybackStopListener Class Reference

`#include <IPlaybackStopListener.hpp>`

Public Member Functions

- virtual ~IPlaybackStopListener ()=default
- virtual void onPlaybackStopped ()=0

  Will be called if the playback is stopped.

8.20.1 Detailed Description

Examples

sampleExportPLY.cpp, and sampleIReplay.cpp.

8.20.2 Constructor & Destructor Documentation

8.20.2.1 ~IPlaybackStopListener()

`virtual ~IPlaybackStopListener ( ) [virtual], [default]`

8.20.3 Member Function Documentation

8.20.3.1 onPlaybackStopped()

```
virtual void onPlaybackStopped ( )  [pure virtual]
```

Will be called if the playback is stopped.

Examples

sampleIReplay.cpp.

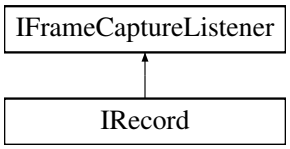
The documentation for this class was generated from the following file:

- IPlaybackStopListener.hpp

8.21 IRecord Class Reference

```
#include <IRecord.hpp>
```

Inheritance diagram for IRecord:



Public Member Functions

- virtual ~IRecord () override
- virtual bool isRecording ()=0

  Indicates that recording is currently active.
- virtual void setProcessingParameters (const royale::ProcessingParameterVector &parameters, const royale::StreamId streamId)=0

  Set/alter processing parameters in order to control the data output when the recording is replayed.
- virtual void resetParameters ()=0

  Resets the internal processing parameters of the recording.
- virtual void startRecord (const royale::String &filename, const std::vector< uint8_t > &calibrationData, const royale::String &imagerSerial, const uint32_t numFrames=0, const uint32_t frameSkip=0, const uint32_t msSkip=0)=0

Starts a recording under the given filename.

- virtual void stopRecord ()=0

  Stops the current recording.
- virtual bool setFrameCaptureListener (royale::collector::IFrameCaptureListener *captureListener)=0

  Set/alter the current IFrameCaptureListener.
- virtual void registerEventListener (royale::IEventListener *listener)=0

  Register listener for event notifications.
- virtual void unregisterEventListener ()=0

  Unregisters listener for event notifications.
- virtual operator bool () const =0

  bool conversion operator, returns if object is a "real" recorder or just a dummy.

8.21.1 Constructor & Destructor Documentation

8.21.1.1 ~IRecord()

```
virtual ~IRecord ( )  [override], [virtual]
```

8.21.2 Member Function Documentation

8.21.2.1 isRecording()

```
virtual bool isRecording ( )  [pure virtual]
```

Indicates that recording is currently active.

8.21.2.2 operator bool()

```
virtual operator bool ( ) const  [pure virtual]
```

bool conversion operator, returns if object is a "real" recorder or just a dummy.

8.21.2.3 registerEventListener()

```
virtual void registerEventListener (
            royale::IEventListener * listener )  [pure virtual]
```

Register listener for event notifications.

8.21.2.4 resetParameters()

```
virtual void resetParameters ( )  [pure virtual]
```

Resets the internal processing parameters of the recording.

Otherwise the recording would not know when it is safe to discard old parameter sets.

8.21.2.5 setFrameCaptureListener()

```
virtual bool setFrameCaptureListener (
            royale::collector::IFrameCaptureListener * captureListener )  [pure virtual]
```

Set/alter the current IFrameCaptureListener.

This can only be done if no recording is happening. If the system is recording, the listener will not be changed and false is returned.

8.21.2.6 setProcessingParameters()

```
virtual void setProcessingParameters (
            const royale::ProcessingParameterVector & parameters,
            const royale::StreamId streamId )  [pure virtual]
```

Set/alter processing parameters in order to control the data output when the recording is replayed.

A list of processing flags is available as an enumeration. The `Variant` data type can take float, int, or bool. Please make sure to set the proper `Variant` type for the enum.

8.21.2.7 startRecord()

```
virtual void startRecord (
            const royale::String & filename,
            const std::vector< uint8_t > & calibrationData,
            const royale::String & imagerSerial,
            const uint32_t numFrames = 0,
            const uint32_t frameSkip = 0,
            const uint32_t msSkip = 0 )  [pure virtual]
```

Starts a recording under the given filename.

If there is already a recording running it will be stopped and the new recording will start.

Parameters

filename	Filename which should be used
calibrationData	Calibration data used for the recording
imagerSerial	Serial number of the imager used for the recording
numFrames	Number of frames which should be recorded (0 equals infinite frames)
frameSkip	Number of frames which should be skipped after every recorded frame (0 equals all frames will be recorded)
msSkip	Time which should be skipped after every recorded frame (0 equals all frames will be recorded)

8.21.2.8 stopRecord()

```
virtual void stopRecord ( )  [pure virtual]
```

Stops the current recording.

If no recording is running the function will return.

8.21.2.9 unregisterEventListener()

```
virtual void unregisterEventListener ( )  [pure virtual]
```

Unregisters listener for event notifications.

The documentation for this class was generated from the following file:

- IRecord.hpp

8.22 IRecordStopListener Class Reference

This interface needs to be implemented if the client wants to get notified when recording stopped after the specified number of frames.

```
#include <IRecordStopListener.hpp>
```

Public Member Functions

- virtual ROYALE_API ~IRecordStopListener ()
- virtual ROYALE_API void onRecordingStopped (const uint32_t numFrames)=0

  Will be called if the recording is stopped.

8.22.1 Detailed Description

This interface needs to be implemented if the client wants to get notified when recording stopped after the specified number of frames.

Examples

sampleRecordRRF.cpp.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 ~IRecordStopListener()

```
virtual ROYALE_API ~IRecordStopListener ( )  [virtual]
```

8.22.3 Member Function Documentation

8.22.3.1 onRecordingStopped()

```
virtual ROYALE_API void onRecordingStopped (
            const uint32_t numFrames )  [pure virtual]
```

Will be called if the recording is stopped.

Parameters

numFrames	Number of frames that have been recorded

Examples

sampleRecordRRF.cpp.

The documentation for this class was generated from the following file:

- IRecordStopListener.hpp

8.23 IReplay Class Reference

`#include <IReplay.hpp>`

Public Member Functions

- virtual ~IReplay ()
- virtual royale::CameraStatus seek (const uint32_t frameNumber)=0
  Seek to a different frame inside the recording.
- virtual void loop (const bool restart)=0
  Enable/Disable looping of the playback.
- virtual void useTimestamps (const bool timestampsUsed)=0
  If enabled, the playback will respect the timestamps and will pause accordingly.
- virtual uint32_t frameCount ()=0
  Retrieves the number of frames in the recording.
- virtual uint32_t currentFrame ()=0
  Retrieves the current frame in the recording.
- virtual void pause ()=0
  Pauses the current playback.
- virtual void resume ()=0
  Resumes the current playback.
- virtual void registerStopListener (royale::IPlaybackStopListener *listener)=0
  Once registering the playback stop listener it will be called when the playback is stopped.
- virtual void unregisterStopListener ()=0
  Unregisters the playback stop listener.
- virtual uint16_t getFileVersion ()=0
  Retrieves the version of the file that was opened.
- virtual uint32_t getMajorVersion ()=0
  Retrieves the build of royale that created this file.
- virtual uint32_t getMinorVersion ()=0
  Retrieves the build of royale that created this file.
- virtual uint32_t getPatchVersion ()=0
  Retrieves the build of royale that created this file.
- virtual uint32_t getBuildVersion ()=0
  Retrieves the build of royale that created this file.
- virtual royale::CameraStatus setPlaybackRange (uint32_t first, uint32_t last)=0
  Sets the playback range.
- virtual void getPlaybackRange (uint32_t &first, uint32_t &last)=0
  Retrieves the playback range.

8.23.1 Detailed Description

Examples

sampleExportPLY.cpp, and sampleIReplay.cpp.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 ~IReplay()

```
virtual ~IReplay ( )  [virtual]
```

8.23.3 Member Function Documentation

8.23.3.1 currentFrame()

```
virtual uint32_t currentFrame ( )  [pure virtual]
```

Retrieves the current frame in the recording.

8.23.3.2 frameCount()

```
virtual uint32_t frameCount ( )  [pure virtual]
```

Retrieves the number of frames in the recording.

8.23.3.3 getBuildVersion()

```
virtual uint32_t getBuildVersion ( )  [pure virtual]
```

Retrieves the build of royale that created this file.

8.23.3.4 getFileVersion()

```
virtual uint16_t getFileVersion ( )  [pure virtual]
```

Retrieves the version of the file that was opened.

8.23.3.5 getMajorVersion()

```
virtual uint32_t getMajorVersion ( )  [pure virtual]
```

Retrieves the build of royale that created this file.

8.23.3.6 getMinorVersion()

```
virtual uint32_t getMinorVersion ( )  [pure virtual]
```

Retrieves the build of royale that created this file.

8.23.3.7 getPatchVersion()

```
virtual uint32_t getPatchVersion ( )  [pure virtual]
```

Retrieves the build of royale that created this file.

8.23.3.8 getPlaybackRange()

```
virtual void getPlaybackRange (
            uint32_t & first,
            uint32_t & last )  [pure virtual]
```

Retrieves the playback range.

8.23.3.9 loop()

```
virtual void loop (
            const bool restart )  [pure virtual]
```

Enable/Disable looping of the playback.

Parameters

restart	Enable/Disable looping

Examples

sampleExportPLY.cpp.

8.23.3.10 pause()

```
virtual void pause ( )  [pure virtual]
```

Pauses the current playback.

8.23.3.11 registerStopListener()

```
virtual void registerStopListener (
            royale::IPlaybackStopListener * listener )  [pure virtual]
```

Once registering the playback stop listener it will be called when the playback is stopped.

Parameters

listener	interface which needs to implement the callback method

Examples

sampleIReplay.cpp.

8.23.3.12 resume()

```
virtual void resume ( )  [pure virtual]
```

Resumes the current playback.

8.23.3.13 seek()

```
virtual royale::CameraStatus seek (
            const uint32_t frameNumber )  [pure virtual]
```

Seek to a different frame inside the recording.

Parameters

frameNumber	frame which will be read next

8.23.3.14 setPlaybackRange()

```
virtual royale::CameraStatus setPlaybackRange (
            uint32_t first,
            uint32_t last )  [pure virtual]
```

Sets the playback range.

8.23.3.15 unregisterStopListener()

```
virtual void unregisterStopListener ( )  [pure virtual]
```

Unregisters the playback stop listener.

8.23.3.16 useTimestamps()

```
virtual void useTimestamps (
            const bool timestampsUsed )  [pure virtual]
```

If enabled, the playback will respect the timestamps and will pause accordingly.

Parameters

timestampsUsed	Use timestamps

The documentation for this class was generated from the following file:

- IReplay.hpp

8.24 IRImage Struct Reference

Infrared image with 8Bit mono information for every pixel.

`#include <IRImage.hpp>`

Public Attributes

- int64_t timestamp
  timestamp for the frame
- StreamId streamId
  stream which produced the data
- uint16_t width
  width of depth image
- uint16_t height
  height of depth image
- royale::Vector< uint8_t > data
  8Bit mono IR image

8.24.1 Detailed Description

Infrared image with 8Bit mono information for every pixel.

8.24.2 Member Data Documentation

8.24.2.1 data

`royale::Vector<uint8_t> data`

8Bit mono IR image

8.24.2.2 height

`uint16_t height`

height of depth image

8.24.2.3 streamId

`StreamId streamId`

stream which produced the data

8.24.2.4 timestamp

`int64_t timestamp`

timestamp for the frame

8.24.2.5 width

`uint16_t width`

width of depth image

The documentation for this struct was generated from the following file:

- IRImage.hpp

8.25 ISparsePointCloudListener Class Reference

Provides the listener interface for consuming sparse point clouds from Royale.

`#include <ISparsePointCloudListener.hpp>`

Public Member Functions

- virtual ~ISparsePointCloudListener ()
- virtual void onNewData (const royale::SparsePointCloud *data)=0

  Will be called on every frame update by the Royale framework.

8.25.1 Detailed Description

Provides the listener interface for consuming sparse point clouds from Royale.

A listener needs to implement this interface and register itself as a listener to the ICameraDevice.

8.25.2 Constructor & Destructor Documentation

8.25.2.1 ~ISparsePointCloudListener()

```
virtual ~ISparsePointCloudListener ( )  [inline], [virtual]
```

8.25.3 Member Function Documentation

8.25.3.1 onNewData()

```
virtual void onNewData (
            const royale::SparsePointCloud * data )  [pure virtual]
```

Will be called on every frame update by the Royale framework.

NOTICE: Calling other framework functions within the data callback can lead to undefined behavior and is therefore unsupported. Call these framework functions from another thread to avoid problems.

The documentation for this class was generated from the following file:

- ISparsePointCloudListener.hpp

8.26 LensParameters Struct Reference

This container stores the lens parameters from the camera module.

`#include <LensParameters.hpp>`

Public Attributes

- royale::Pair< float, float > principalPoint
  cx/cy
- royale::Pair< float, float > focalLength
  fx/fy
- royale::Pair< float, float > distortionTangential
  p1/p2
- royale::Vector< float > distortionRadial
  k1/k2/k3

8.26.1 Detailed Description

This container stores the lens parameters from the camera module.

Examples

sampleCameraInfo.cpp, and sampleOpenCV.cpp.

8.26.2 Member Data Documentation

8.26.2.1 distortionRadial

`royale::Vector<float> distortionRadial`

k1/k2/k3

Examples

sampleCameraInfo.cpp, and sampleOpenCV.cpp.

8.26.2.2 distortionTangential

```
royale::Pair<float, float> distortionTangential
```

p1/p2

Examples

sampleCameraInfo.cpp, and sampleOpenCV.cpp.

8.26.2.3 focalLength

```
royale::Pair<float, float> focalLength
```

fx/fy

Examples

sampleCameraInfo.cpp, and sampleOpenCV.cpp.

8.26.2.4 principalPoint

```
royale::Pair<float, float> principalPoint
```

cx/cy

Examples

sampleCameraInfo.cpp, and sampleOpenCV.cpp.

The documentation for this struct was generated from the following file:

- LensParameters.hpp

8.27 RawData Struct Reference

This structure defines the raw data which is delivered through the callback only exposed for access LEVEL 2.

```
#include <RawData.hpp>
```

Public Member Functions

- ROYALE_API RawData ()
- ROYALE_API RawData (size_t rawVectorSize)

Public Attributes

- std::chrono::microseconds timeStamp
  timestamp in microseconds precision (time since epoch 1970)
- StreamId streamId
  stream which produced the data
- uint16_t width
  width of raw frame
- uint16_t height
  height of raw frame
- royale::Vector< const uint16_t * > rawData
  pointer to each raw frame
- royale::Vector< royale::String > exposureGroupNames
  name of each exposure group
- royale::Vector< size_t > rawFrameCount
  raw frame count of each exposure group
- royale::Vector< uint32_t > modulationFrequencies
  modulation frequencies for each sequence
- royale::Vector< uint32_t > exposureTimes
  integration times for each sequence
- float illuminationTemperature
  temperature of illumination
- royale::Vector< uint16_t > phaseAngles
  phase angles for each raw frame
- royale::Vector< uint8_t > illuminationEnabled
  status of the illumination for each raw frame (1-enabled/0-disabled)

8.27.1 Detailed Description

This structure defines the raw data which is delivered through the callback only exposed for access LEVEL 2.

This data comprises the raw phase images coming directly from the imager.

8.27.2 Constructor & Destructor Documentation

8.27.2.1 RawData() [1/2]

```
ROYALE_API RawData ( )  [explicit]
```

8.27.2.2 RawData() [2/2]

```
ROYALE_API RawData (
            size_t rawVectorSize )  [explicit]
```

8.27.3 Member Data Documentation

8.27.3.1 exposureGroupNames

```
royale::Vector<royale::String> exposureGroupNames
```

name of each exposure group

8.27.3.2 exposureTimes

```
royale::Vector<uint32_t> exposureTimes
```

integration times for each sequence

8.27.3.3 height

```
uint16_t height
```

height of raw frame

8.27.3.4 illuminationEnabled

`royale::Vector<uint8_t> illuminationEnabled`

status of the illumination for each raw frame (1-enabled/0-disabled)

8.27.3.5 illuminationTemperature

`float illuminationTemperature`

temperature of illumination

8.27.3.6 modulationFrequencies

`royale::Vector<uint32_t> modulationFrequencies`

modulation frequencies for each sequence

8.27.3.7 phaseAngles

`royale::Vector<uint16_t> phaseAngles`

phase angles for each raw frame

8.27.3.8 rawData

`royale::Vector<const uint16_t *> rawData`

pointer to each raw frame

8.27.3.9 rawFrameCount

```
royale::Vector<size_t> rawFrameCount
```

raw frame count of each exposure group

8.27.3.10 streamId

```
StreamId streamId
```

stream which produced the data

8.27.3.11 timeStamp

```
std::chrono::microseconds timeStamp
```

timestamp in microseconds precision (time since epoch 1970)

8.27.3.12 width

```
uint16_t width
```

width of raw frame

The documentation for this struct was generated from the following file:

- RawData.hpp

8.28 SparsePointCloud Struct Reference

The sparse point cloud gives XYZ and confidence for every valid point.

```
#include <SparsePointCloud.hpp>
```

Public Attributes

- int64_t timestamp
  timestamp for the frame
- StreamId streamId
  stream which produced the data
- uint32_t numPoints
  the number of valid points
- royale::Vector< float > xyzcPoints
  XYZ and confidence for every valid point.

8.28.1 Detailed Description

The sparse point cloud gives XYZ and confidence for every valid point.

It is given as an array of packed coordinate quadruplets (x,y,z,c) as floating point values. The x, y and z coordinates are in meters. The confidence (c) has a floating point value in [0.0, 1.0], where 1 corresponds to full confidence.

8.28.2 Member Data Documentation

8.28.2.1 numPoints

```
uint32_t numPoints
```

the number of valid points

8.28.2.2 streamId

```
StreamId streamId
```

stream which produced the data

8.28.2.3 timestamp

```
int64_t timestamp
```

timestamp for the frame

8.28.2.4 xyzcPoints

```
royale::Vector<float> xyzcPoints
```

XYZ and confidence for every valid point.

The documentation for this struct was generated from the following file:

- SparsePointCloud.hpp

8.29 Variant Class Reference

Implements a variant type which can take different basic data types, the default type is int and the value is set to zero.

```
#include <Variant.hpp>
```

Classes

- struct InvalidType

  This will be thrown if a wrong type is used.

Public Member Functions

- ROYALE_API Variant ()
- ROYALE_API Variant (int n, int min=std::numeric_limits< int >::lowest(), int max=std::numeric_limits< int >::max())
- ROYALE_API Variant (float n, float min=std::numeric_limits< float >::lowest(), float max=std::numeric_limits< float >::max())
- ROYALE_API Variant (bool n)
- ROYALE_API Variant (royale::VariantType type, uint32_t value)
- ROYALE_API Variant (royale::String val, royale::Vector< royale::Pair< royale::String, int >> possibleVals)
- ROYALE_API Variant (int val, royale::Vector< royale::Pair< royale::String, int >> possibleVals)
- ROYALE_API ~Variant ()
- ROYALE_API void setFloat (float n)
- ROYALE_API float getFloat () const
- ROYALE_API float getFloatMin () const
- ROYALE_API float getFloatMax () const
- ROYALE_API void setInt (int n)
- ROYALE_API int getInt () const
- ROYALE_API int getIntMin () const
- ROYALE_API int getIntMax () const
- ROYALE_API void setBool (bool n)
- ROYALE_API bool getBool () const
- ROYALE_API void setData (royale::VariantType type, uint32_t value)
- ROYALE_API uint32_t getData () const
- ROYALE_API royale::VariantType variantType () const
- ROYALE_API void setEnumValue (int val)
- ROYALE_API void setEnumValue (royale::String val)
- ROYALE_API int getEnumValue () const
- ROYALE_API royale::String getEnumString () const
- ROYALE_API royale::Vector< royale::Pair< royale::String, int > > getPossibleVals () const
- ROYALE_API bool operator== (const royale::Variant &v) const
- ROYALE_API bool operator!= (const royale::Variant &v) const
- ROYALE_API bool operator< (const royale::Variant &v) const

8.29.1 Detailed Description

Implements a variant type which can take different basic data types, the default type is int and the value is set to zero.

8.29.2 Constructor & Destructor Documentation

8.29.2.1 Variant() [1/7]

```
ROYALE_API Variant ( )
```

8.29.2.2 Variant() [2/7]

```
ROYALE_API Variant (
            int n,
            int min = std::numeric_limits< int >::lowest(),
            int max = std::numeric_limits< int >::max() )
```

8.29.2.3 Variant() [3/7]

```
ROYALE_API Variant (
            float n,
            float min = std::numeric_limits< float >::lowest(),
            float max = std::numeric_limits< float >::max() )
```

8.29.2.4 Variant() [4/7]

```
ROYALE_API Variant (
            bool n )
```

8.29.2.5 Variant() [5/7]

```
ROYALE_API Variant (
            royale::VariantType type,
            uint32_t value )
```

8.29.2.6 Variant() [6/7]

```
ROYALE_API Variant (
            royale::String val,
            royale::Vector< royale::Pair< royale::String, int >> possibleVals )
```

8.29.2.7 Variant() [7/7]

```
ROYALE_API Variant (
            int val,
            royale::Vector< royale::Pair< royale::String, int >> possibleVals )
```

8.29.2.8 ~Variant()

```
ROYALE_API ~Variant ( )
```

8.29.3 Member Function Documentation

8.29.3.1 getBool()

```
ROYALE_API bool getBool ( ) const
```

8.29.3.2 getData()

```
ROYALE_API uint32_t getData ( ) const
```

8.29.3.3 getEnumString()

```
ROYALE_API royale::String getEnumString ( ) const
```

8.29.3.4 getEnumValue()

```
ROYALE_API int getEnumValue ( ) const
```

8.29.3.5 getFloat()

```
ROYALE_API float getFloat ( ) const
```

8.29.3.6 getFloatMax()

```
ROYALE_API float getFloatMax ( ) const
```

8.29.3.7 getFloatMin()

```
ROYALE_API float getFloatMin ( ) const
```

8.29.3.8 getInt()

```
ROYALE_API int getInt ( ) const
```

8.29.3.9 getIntMax()

```
ROYALE_API int getIntMax ( ) const
```

8.29.3.10 getIntMin()

```
ROYALE_API int getIntMin ( ) const
```

8.29.3.11 getPossibleVals()

```
ROYALE_API royale::Vector<royale::Pair<royale::String, int> > getPossibleVals ( ) const
```

8.29.3.12 operator"!=()

```
ROYALE_API bool operator!= (
            const royale::Variant & v ) const
```

8.29.3.13 operator<()

```
ROYALE_API bool operator< (
            const royale::Variant & v ) const
```

8.29.3.14 operator==()

```
ROYALE_API bool operator== (
            const royale::Variant & v ) const
```

8.29.3.15 setBool()

```
ROYALE_API void setBool (
            bool n )
```

8.29.3.16 setData()

```
ROYALE_API void setData (
            royale::VariantType type,
            uint32_t value )
```

8.29.3.17 setEnumValue() [1/2]

```
ROYALE_API void setEnumValue (
            int val )
```

8.29.3.18 setEnumValue() [2/2]

```
ROYALE_API void setEnumValue (
            royale::String val )
```

8.29.3.19 setFloat()

```
ROYALE_API void setFloat (
            float n )
```

8.29.3.20 setInt()

```
ROYALE_API void setInt (
            int n )
```

8.29.3.21 variantType()

```
ROYALE_API royale::VariantType variantType ( ) const
```

8.29.4 Member Data Documentation

8.29.4.1 b

```
bool b
```

8.29.4.2 f

```
float f
```

8.29.4.3 i

```
int i
```

The documentation for this class was generated from the following file:

- Variant.hpp


infineon
pmd
can you imagine

Chapter 9

File Documentation

9.1 CallbackData.hpp File Reference

Namespaces

- royale

Enumerations

- enum CallbackData : uint16_t { None = 0x00, Raw = 0x01, Depth = 0x02, Intermediate = 0x04 }
  Specifies the type of data which should be captured and returned as callback.

9.2 CameraAccessLevel.hpp File Reference

Namespaces

- royale

Enumerations

- enum CameraAccessLevel { L1 = 1, L2 = 2, L3 = 3, L4 = 4 }
  This enum defines the access level.

9.3 CameraManager.hpp File Reference

```
#include <memory>
#include <royale/Definitions.hpp>
#include <royale/ICameraDevice.hpp>
#include <royale/Vector.hpp>
#include <royale/String.hpp>
#include <royale/TriggerMode.hpp>
```

Classes

- class CameraManager

  The CameraManager is responsible for detecting and creating instances of `ICameraDevices` one for each connected (supported) camera device.

Namespaces

- royale

9.4 Definitions.hpp File Reference

Macros

- #define ROYALE_API
- #define ADD_DEBUG_CONSOLE

9.4.1 Macro Definition Documentation

9.4.1.1 ADD_DEBUG_CONSOLE

```
#define ADD_DEBUG_CONSOLE
```

9.4.1.2 ROYALE_API

```
#define ROYALE_API
```

9.5 DepthData.hpp File Reference

```
#include <royale/Definitions.hpp>
#include <royale/Vector.hpp>
#include <royale/StreamId.hpp>
#include <memory>
#include <cstdint>
#include <cstring>
#include <chrono>
```

Classes

- struct DepthPoint

  Encapsulates a 3D point in object space, with coordinates in meters.
- struct DepthData

  This structure defines the depth data which is delivered through the callback.

Namespaces

- royale

9.6 DepthImage.hpp File Reference

```
#include <royale/Definitions.hpp>
#include <royale/Vector.hpp>
#include <royale/StreamId.hpp>
#include <memory>
#include <cstdint>
```

Classes

- struct DepthImage

  The depth image represents the depth and confidence for every pixel.

Namespaces

- royale

9.7 DepthIRImage.hpp File Reference

```
#include <royale/DepthImage.hpp>
#include <royale/IRImage.hpp>
```

Classes

- struct DepthIRImage
  This represents combination of both depth and IR image.

Namespaces

- royale

9.8 ExposureMode.hpp File Reference

Namespaces

- royale

Enumerations

- enum ExposureMode { MANUAL, AUTOMATIC }
  The ExposureMode is used to switch between manual and automatic exposure time handling.

9.9 FilterLevel.hpp File Reference

```
#include <royale/String.hpp>
```

Namespaces

- royale

Enumerations

- enum FilterLevel {
  Off = 0, Deprecated1 = 1, Deprecated2 = 2, Deprecated3 = 3,
  Deprecated4 = 4, IR1 = 5, IR2 = 6, AF1 = 7,
  CM1 = 8, Binning_1_Basic = 9, Binning_2_Basic = 10, Binning_3_Basic = 11,
  Binning_4_Basic = 12, Binning_8_Basic = 13, Binning_10_Basic = 14, Binning_1_Efficiency = 15,
  Binning_2_Efficiency = 16, Binning_3_Efficiency = 17, Binning_4_Efficiency = 18, Binning_8_Efficiency = 19,
  Binning_10_Efficiency = 20, Legacy = 200, Full = 255, Custom = 256 }

  Royale allows to set different filter levels.

Functions

- ROYALE_API royale::String getFilterLevelName (royale::FilterLevel level)
- ROYALE_API royale::FilterLevel getFilterLevelFromName (royale::String name)

9.10 ICameraDevice.hpp File Reference

```
#include <memory>
#include <royale/Status.hpp>
#include <royale/IDepthDataListener.hpp>
#include <royale/IDepthImageListener.hpp>
#include <royale/ISparsePointCloudListener.hpp>
#include <royale/IIRImageListener.hpp>
#include <royale/IDepthIRImageListener.hpp>
#include <royale/IExtendedDataListener.hpp>
#include <royale/IEventListener.hpp>
#include <royale/LensParameters.hpp>
#include <royale/ProcessingFlag.hpp>
#include <royale/CallbackData.hpp>
#include <royale/IRecordStopListener.hpp>
#include <royale/IExposureListener.hpp>
#include <royale/IExposureListener2.hpp>
#include <royale/ExposureMode.hpp>
#include <royale/FilterLevel.hpp>
#include <royale/Vector.hpp>
#include <royale/String.hpp>
#include <royale/CameraAccessLevel.hpp>
#include <royale/StreamId.hpp>
```

Classes

- class ICameraDevice
  This is the main interface for talking to the time-of-flight camera system.

Namespaces

- royale

9.11 IDepthDataListener.hpp File Reference

```
#include <string>
#include <royale/DepthData.hpp>
```

Classes

- class IDepthDataListener
  Provides the listener interface for consuming depth data from Royale.

Namespaces

- royale

9.12 IDepthImageListener.hpp File Reference

```
#include <string>
#include <royale/DepthImage.hpp>
```

Classes

- class IDepthImageListener
  Provides a listener interface for consuming depth images from Royale.

Namespaces

- royale

9.13 IDepthIRImageListener.hpp File Reference

```
#include <string>
#include <royale/DepthIRImage.hpp>
```

Classes

- class IDepthIRImageListener
  Provides a combined listener interface for consuming both depth and IR images from Royale.

Namespaces

- royale

9.14 IEvent.hpp File Reference

```
#include <royale/Definitions.hpp>
#include <royale/String.hpp>
```

Classes

- class IEvent
  Interface for anything to be passed via IEventListener.

Namespaces

- royale

Enumerations

- enum EventSeverity { ROYALE_INFO = 0, ROYALE_WARNING = 1, ROYALE_ERROR = 2, ROYALE_FATAL = 3 }
  Severity of an IEvent.
- enum EventType {
  ROYALE_CAPTURE_STREAM, ROYALE_DEVICE_DISCONNECTED, ROYALE_OVER_TEMPERATURE, ROYALE_RAW_FRAME_STAT
  ROYALE_EYE_SAFETY, ROYALE_PROCESSING, ROYALE_RECORDING, ROYALE_FRAME_DROP,
  ROYALE_UNKNOWN, ROYALE_ERROR_DESCRIPTION }
  Type of an IEvent.

9.15 IEventListener.hpp File Reference

```
#include <royale/Definitions.hpp>
#include <memory>
```

Classes

- class IEventListener
  This interface allows observers to receive events.

Namespaces

- royale

9.16 IExposureListener.hpp File Reference

```
#include <royale/Vector.hpp>
#include <royale/Definitions.hpp>
#include <royale/StreamId.hpp>
#include <cstdint>
```

Classes

- class IExposureListener
  Provides the listener interface for handling auto-exposure updates in royale.

Namespaces

- royale

9.17 IExposureListener2.hpp File Reference

```
#include <royale/Definitions.hpp>
#include <royale/StreamId.hpp>
#include <cstdint>
```

Classes

- class IExposureListener2
  Provides the listener interface for handling auto-exposure updates in royale.

Namespaces

- royale

9.18 IExtendedData.hpp File Reference

```
#include <royale/Definitions.hpp>
#include <royale/RawData.hpp>
#include <royale/DepthData.hpp>
#include <royale/IntermediateData.hpp>
#include <memory>
#include <cstdint>
#include <vector>
#include <chrono>
```

Classes

- class IExtendedData
  Interface for getting additional data to the standard depth data.

Namespaces

- royale

9.19 IExtendedDataListener.hpp File Reference

```
#include <string>
#include <royale/IExtendedData.hpp>
```

Classes

- class IExtendedDataListener

Namespaces

- royale

9.20 IIRImageListener.hpp File Reference

```
#include <string>
#include <royale/IRImage.hpp>
```

Classes

- class IIRImageListener

  Provides the listener interface for consuming infrared images from Royale.

Namespaces

- royale

9.21 IntermediateData.hpp File Reference

```
#include <royale/Definitions.hpp>
#include <royale/DepthData.hpp>
#include <royale/Vector.hpp>
#include <royale/StreamId.hpp>
#include <royale/ProcessingFlag.hpp>
#include <memory>
#include <cstdint>
#include <chrono>
```

Classes

- struct IntermediatePoint

  In addition to the standard depth point, the intermediate point also stores information which is calculated as temporaries in the processing pipeline.
- struct IntermediateData

  This structure defines the Intermediate depth data which is delivered through the callback if the user has access level 2 for the CameraDevice.

Namespaces

- royale

9.22 IPlaybackStopListener.hpp File Reference

```
#include <cstdint>
```

Classes

- class IPlaybackStopListener

Namespaces

- royale

9.23 IRecord.hpp File Reference

```
#include <collector/IFrameCaptureListener.hpp>
#include <royale/IEventListener.hpp>
#include <royale/ProcessingFlag.hpp>
#include <royale/StreamId.hpp>
```

Classes

- class IRecord

Namespaces

- royale

9.24 IRecordStopListener.hpp File Reference

```
#include <cstdint>
#include <royale/Definitions.hpp>
```

Classes

- class IRecordStopListener
  This interface needs to be implemented if the client wants to get notified when recording stopped after the specified number of frames.

Namespaces

- royale

9.25 IReplay.hpp File Reference

```
#include <cstdint>
#include <royale/Status.hpp>
#include <royale/IPlaybackStopListener.hpp>
```

Classes

- class IReplay

Namespaces

- royale

9.26 IRImage.hpp File Reference

```
#include <royale/Definitions.hpp>
#include <royale/Vector.hpp>
#include <royale/StreamId.hpp>
#include <memory>
#include <cstdint>
```

Classes

- struct IRImage
  Infrared image with 8Bit mono information for every pixel.

Namespaces

- royale

9.27 ISparsePointCloudListener.hpp File Reference

```
#include <string>
#include <royale/SparsePointCloud.hpp>
```

Classes

- class ISparsePointCloudListener
  Provides the listener interface for consuming sparse point clouds from Royale.

Namespaces

- royale

9.28 LensParameters.hpp File Reference

```
#include <royale/Pair.hpp>
#include <royale/Vector.hpp>
```

Classes

- struct LensParameters
  This container stores the lens parameters from the camera module.

Namespaces

- royale

9.29 ProcessingFlag.hpp File Reference

```
#include <royale/SpectreProcessingType.hpp>
#include <royale/Variant.hpp>
#include <royale/Vector.hpp>
#include <royale/String.hpp>
```

Namespaces

- royale
- royale::parameter

Typedefs

- typedef royale::Vector< royale::Pair< royale::ProcessingFlag, royale::Variant > > ProcessingParameterVector

  This is a map combining a set of flags which can be set/altered in access LEVEL 2 and the set value as Variant type.
- typedef std::map< royale::ProcessingFlag, royale::Variant > ProcessingParameterMap
- typedef std::pair< royale::ProcessingFlag, royale::Variant > ProcessingParameterPair

Enumerations

- enum ProcessingFlag {
  ConsistencyTolerance_Float = 0, FlyingPixelsF0_Float, FlyingPixelsF1_Float, FlyingPixelsFarDist_Float,
  FlyingPixelsNearDist_Float, LowerSaturationThreshold_Int, UpperSaturationThreshold_Int, MPIAmpThreshold_Float,
  MPIDistThreshold_Float, MPINoiseDistance_Float, NoiseThreshold_Float, AdaptiveNoiseFilterType_Int,
  AutoExposureRefAmplitude_Float, UseAdaptiveNoiseFilter_Bool, UseAutoExposure_Bool, UseRemoveFlyingPixel_Bool,
  UseMPIFlagAverage_Bool, UseMPIFlag_Amp_Bool, UseMPIFlag_Dist_Bool, UseValidateImage_Bool,
  UseRemoveStrayLight_Bool, UseSparsePointCloud_Bool, UseFilter2Freq_Bool, GlobalBinning_Int,
  UseAdaptiveBinning_Bool, AutoExposureRefValue_Float, UseSmoothingFilter_Bool, SmoothingAlpha_Float,
  SmoothingFilterType_Int, UseFlagSBI_Bool, UseHoleFilling_Bool, Reserved1,
  Reserved2, Reserved3, Reserved4, Reserved5,
  Reserved6, Reserved7, Reserved8, AutoExpoMin_Int,
  AutoExpoMax_Int, SpectreProcessingType_Int, UseGrayImageFallbackAmplitude_Bool, GrayImageMeanMap_Int,
  NoiseFilterSigmaD_Float, NoiseFilterIterations_Int, FlyingPixelAngleLimit_Float, FlyingPixelAmpThreshold_Float,
  FlyingPixelMinNeighbors_Int, FlyingPixelMaxNeighbors_Int, FlyingPixelNoiseRatioThresh_Float, SmoothingFilterResetThreshold_Float,
  CCThresh_Int, PhaseNoiseThresh_Float, StraylightThreshold_Float, NoiseFilterSigmaA_Float,
  TwoFreqCombinationType_Int, UseCorrectMPI_Bool, NUM_FLAGS }

  This is a list of flags which can be set/altered in access LEVEL 2 in order to control the processing pipeline.

Functions

- ROYALE_API royale::String getProcessingFlagName (royale::ProcessingFlag mode)

  For debugging, printable strings corresponding to the ProcessingFlag enumeration.
- ROYALE_API bool parseProcessingFlagName (const royale::String &modeName, royale::ProcessingFlag &processing↩ Flag)

  Convert a string received from getProcessingFlagName back into its ProcessingFlag.
- ROYALE_API ProcessingParameterMap combineProcessingMaps (const ProcessingParameterMap &a, const ProcessingParameterMap &b)

  Takes ProcessingParameterMaps a and b and returns a combination of both.

9.30 RawData.hpp File Reference

```
#include <royale/Definitions.hpp>
#include <royale/Vector.hpp>
#include <royale/String.hpp>
#include <royale/StreamId.hpp>
#include <memory>
#include <cstdint>
#include <chrono>
```

Classes

- struct RawData

  This structure defines the raw data which is delivered through the callback only exposed for access LEVEL 2.

Namespaces

- royale

9.31 README.md File Reference

9.32 SparsePointCloud.hpp File Reference

```
#include <royale/Definitions.hpp>
#include <royale/Vector.hpp>
#include <royale/StreamId.hpp>
#include <memory>
#include <cstdint>
```

Classes

- struct SparsePointCloud
  The sparse point cloud gives XYZ and confidence for every valid point.

Namespaces

- royale

9.33 SpectreProcessingType.hpp File Reference

```
#include <royale/String.hpp>
```

Namespaces

- royale

Enumerations

- enum SpectreProcessingType {
  AUTO = 1, CB_BINNED_WS = 2, NG = 3, AF = 4,
  CB_BINNED_NG = 5, GRAY_IMAGE = 6, CM_FI = 7, NUM_TYPES }
  This is a list of pipelines that can be set in Spectre.

Functions

- ROYALE_API royale::String getSpectreProcessingTypeName (royale::SpectreProcessingType mode)
  Converts the given processing type into a readable string.
- ROYALE_API bool getSpectreProcessingTypeFromName (const royale::String &modeName, royale::SpectreProcessingType &processingType)
  Converts the name of a processing type into an enum value.

9.34 Status.hpp File Reference

```
#include <royale/String.hpp>
```

Namespaces

- royale

Enumerations

- enum CameraStatus {
  SUCCESS = 0, RUNTIME_ERROR = 1024, DISCONNECTED = 1026, INVALID_VALUE = 1027,
  TIMEOUT = 1028, LOGIC_ERROR = 2048, NOT_IMPLEMENTED = 2049, OUT_OF_BOUNDS = 2050,
  RESOURCE_ERROR = 4096, FILE_NOT_FOUND = 4097, COULD_NOT_OPEN = 4098, DATA_NOT_FOUND = 4099,
  DEVICE_IS_BUSY = 4100, WRONG_DATA_FORMAT_FOUND = 4101, USECASE_NOT_SUPPORTED = 5001,
  FRAMERATE_NOT_SUPPORTED = 5002,
  EXPOSURE_TIME_NOT_SUPPORTED = 5003, DEVICE_NOT_INITIALIZED = 5004, CALIBRATION_DATA_ERROR
  = 5005, INSUFFICIENT_PRIVILEGES = 5006,
  DEVICE_ALREADY_INITIALIZED = 5007, EXPOSURE_MODE_INVALID = 5008, NO_CALIBRATION_DATA = 5009,
  INSUFFICIENT_BANDWIDTH = 5010,
  DUTYCYCLE_NOT_SUPPORTED = 5011, SPECTRE_NOT_INITIALIZED = 5012, NO_USE_CASES = 5013,
  NO_USE_CASES_FOR_LEVEL = 5014,
  FSM_INVALID_TRANSITION = 8096, UNKNOWN = 0x7fffff01 }

Functions

- ROYALE_API royale::String getStatusString (royale::CameraStatus status)
  Get a human-readable description for a given error message.
- inline ::std::ostream & operator<< (::std::ostream &os, royale::CameraStatus status)

9.35 StreamId.hpp File Reference

```
#include <cstdint>
```

Namespaces

- royale

Typedefs

- using StreamId = uint16_t
  The StreamId uniquely identifies a stream of measurements within a usecase.

9.36 TriggerMode.hpp File Reference

Namespaces

- royale

Enumerations

- enum TriggerMode { MASTER = 0, SLAVE = 1 }
  Trigger mode used by the camera.

9.37 Variant.hpp File Reference

```
#include <royale/Definitions.hpp>
#include <royale/Vector.hpp>
#include <royale/String.hpp>
#include <limits>
#include <float.h>
#include <cstdint>
```

Classes

- class Variant
  Implements a variant type which can take different basic data types, the default type is int and the value is set to zero.
- struct Variant::InvalidType
  This will be thrown if a wrong type is used.

Namespaces

- royale

Enumerations

- enum VariantType { Int, Float, Bool, Enum }

Chapter 10

Example Documentation

10.1 sampleCameraInfo.cpp

LEVEL 1 Retrieve further information for this specific camera.The return value is a map, where the keys are depending on the used camera

```
/****************************************************************************\
 * Copyright (C) 2017 Infineon Technologies & pmdtechnologies ag
 *
 * THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
 * KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A
 * PARTICULAR PURPOSE.
 *
 \****************************************************************************/
#include <royale.hpp>
#include <sample_utils/EventReporter.hpp>
#include <sample_utils/PlatformResources.hpp>
int main()
{
    using namespace sample_utils;
    using namespace std;
    // Windows requires that the application allocate these, not the DLL.
    PlatformResources resources;
    // this represents the main camera device object
    std::unique_ptr<royale::ICameraDevice> cameraDevice;
    // the camera manager will query for a connected camera
    {
        royale::CameraManager manager;
        sample_utils::EventReporter eventReporter;
        manager.registerEventListener (&eventReporter);
        royale::Vector<royale::String> camlist (manager.getConnectedCameraList());
        cout « "Detected " « camlist.size() « " camera(s)." « endl;
        bool camlistEmpty = camlist.empty();
        if (!camlistEmpty)
        {
            cameraDevice = manager.createCamera (camlist[0]);
        }
        camlist.clear();
        // EventReporter will be deallocated before manager. So eventReporter must be
        // unregistered. Declare eventReporter before manager to make the next call unnecessary.
        manager.unregisterEventListener ();
        if (camlistEmpty)
```

```
        {
            cerr « "No suitable camera device detected." « endl
#ifdef WIN32
                 « "Please make sure that a supported camera is plugged in and all drivers are installed" «
        endl;
#else
                 « "Please make sure that a supported camera is plugged in and you have proper USB
        permission" « endl;
#endif
            return 1;
        }
    }
    // the camera device is now available and CameraManager can be deallocated here
    if (cameraDevice == nullptr)
    {
        cerr « "Cannot create the camera device" « endl;
        return 1;
    }
    // IMPORTANT: call the initialize method before working with the camera device
    auto status = cameraDevice->initialize();
    if (status != royale::CameraStatus::SUCCESS)
    {
        cerr « "Cannot initialize the camera device, error string : " « getErrorString (status) « endl;
        return 1;
    }
    royale::String id;
    royale::String name;
    uint16_t maxSensorWidth;
    uint16_t maxSensorHeight;
    status = cameraDevice->getId (id);
    if (royale::CameraStatus::SUCCESS != status)
    {
        cerr « "failed to get ID: " « getErrorString (status) « endl;
        return 1;
    }
    status = cameraDevice->getCameraName (name);
    if (royale::CameraStatus::SUCCESS != status)
    {
        cerr « "failed to get name: " « getErrorString (status) « endl;
        return 1;
    }
    status = cameraDevice->getMaxSensorWidth (maxSensorWidth);
    if (royale::CameraStatus::SUCCESS != status)
    {
        cerr « "failed to get max sensor width: " « getErrorString (status) « endl;
        return 1;
    }
    status = cameraDevice->getMaxSensorHeight (maxSensorHeight);
    if (royale::CameraStatus::SUCCESS != status)
    {
        cerr « "failed to get max sensor height: " « getErrorString (status) « endl;
        return 1;
    }
    royale::Vector<royale::String> useCases;
    status = cameraDevice->getUseCases (useCases);
    if (royale::CameraStatus::SUCCESS != status)
    {
        cerr « "failed to get available use cases: " « getErrorString (status) « endl;
        return 1;
    }
    royale::Vector<royale::Pair<royale::String, royale::String» cameraInfo;
    status = cameraDevice->getCameraInfo (cameraInfo);
    if (royale::CameraStatus::SUCCESS != status)
    {
        cerr « "failed to get camera info: " « getErrorString (status) « endl;
        return 1;
    }
    // display some information about the connected camera
    cout « "==================================" « endl;
    cout « "        Camera information"           « endl;
```

```
    cout « "====================================" « endl;
    cout « "Id:               " « id « endl;
    cout « "Type:             " « name « endl;
    cout « "Width:            " « maxSensorWidth « endl;
    cout « "Height:           " « maxSensorHeight « endl;
    cout « "Operation modes:  " « useCases.size() « endl;
    const auto listIndent = std::string ("    ");
    const auto noteIndent = std::string ("        ");
    for (size_t i = 0; i < useCases.size(); ++i)
    {
        cout « listIndent « useCases[i] « endl;
        uint32_t streamCount = 0;
        status = cameraDevice->getNumberOfStreams (useCases[i], streamCount);
        if (royale::CameraStatus::SUCCESS == status && streamCount > 1)
        {
            cout « noteIndent « "this operation mode has " « streamCount « " streams" « endl;
        }
    }
    cout « "CameraInfo items: " « cameraInfo.size() « endl;
    for (size_t i = 0; i < cameraInfo.size(); ++i)
    {
        cout « listIndent « cameraInfo[i] « endl;
    }
    royale::LensParameters lens;
    status = cameraDevice->getLensParameters (lens);
    if (royale::CameraStatus::SUCCESS != status)
    {
        cerr « "failed to get lens parameters: " « getErrorString (status) « endl;
        return 1;
    }
    cout « "Lens focal length: " « lens.focalLength.first « " / " « lens.focalLength.second « endl;
    cout « "Principal point: " « lens.principalPoint.first « " / " « lens.principalPoint.second « endl;
    cout « "Distortion tangential: " « lens.distortionTangential.first « " / " «
       lens.distortionTangential.second « endl;
    cout « "Distortion radial: ";
    for (auto curK : lens.distortionRadial)
    {
        cout « curK « " / ";
    }
    cout « endl;
    return 0;
}
```

10.2 sampleExportPLY.cpp

LEVEL 1 Once registering a record listener, the listener gets notified once recording has stopped after specified frames.

Parameters

listener	interface which needs to implement the callback method

```
/****************************************************************************\
 * Copyright (C) 2017 Infineon Technologies & pmdtechnologies ag
 *
 * THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
 * KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A
 * PARTICULAR PURPOSE.
 *
 \****************************************************************************/
#include <iostream>
#include <thread>
```

```
#include <chrono>
#include <mutex>
#include <fstream>
#include <sstream>
#include <string>
#include <royale.hpp>
#include <royale/IPlaybackStopListener.hpp>
#include <royale/IReplay.hpp>
using namespace std;
namespace
{
    class MyListener : public royale::IDepthDataListener
    {
    public:
        MyListener (string rrfFile, uint32_t numFrames) :
            m_frameNumber (0),
            m_numFrames (numFrames),
            m_rrfFile (std::move (rrfFile))
        {
        }
        void writePLY (const string &filename, const royale::DepthData *data)
        {
            // For an explanation of the PLY file format please have a look at
            // https://en.wikipedia.org/wiki/PLY_(file_format)
            ofstream outputFile;
            stringstream stringStream;
            outputFile.open (filename, ofstream::out);
            if (outputFile.fail())
            {
                cerr « "Outputfile " « filename « " could not be opened!" « endl;
                return;
            }
            else
            {
                // if the file was opened successfully write the PLY header
                stringStream « "ply" « endl;
                stringStream « "format ascii 1.0" « endl;
                stringStream « "comment Generated by sampleExportPLY" « endl;
                stringStream « "element vertex " « data->points.size() « endl;
                stringStream « "property float x" « endl;
                stringStream « "property float y" « endl;
                stringStream « "property float z" « endl;
                stringStream « "element face 0" « endl;
                stringStream « "property list uchar int vertex_index" « endl;
                stringStream « "end_header" « endl;
                // output XYZ coordinates into one line
                for (size_t i = 0; i < data->points.size(); ++i)
                {
                    stringStream « data->points.at (i).x « " " « data->points.at (i).y « " " «
       data->points.at (i).z « endl;
                }
                // output stringstream to file and close it
                outputFile « stringStream.str();
                outputFile.close();
            }
        }
        void onNewData (const royale::DepthData *data) override
        {
            stringstream filename;
            m_frameNumber++;
            cout « "Exporting frame " « m_frameNumber « " of " « m_numFrames « endl;
            filename « m_frameNumber « ".ply";
            writePLY (filename.str(), data);
        }
    private:
        uint32_t m_frameNumber; // The current frame number
        uint32_t m_numFrames;   // Total number of frames in the recording
        string m_rrfFile;       // Recording file that was opened
    };
    class MyPlaybackStopListener : public royale::IPlaybackStopListener
```

```
    {
    public:
        MyPlaybackStopListener()
        {
            m_playbackRunning = true;
        }
        void onPlaybackStopped() override
        {
            lock_guard<mutex> lock (m_stopMutex);
            m_playbackRunning = false;
        }
        void waitForStop()
        {
            bool running = true;
            do
            {
                {
                    lock_guard<mutex> lock (m_stopMutex);
                    running = m_playbackRunning;
                }
                this_thread::sleep_for (chrono::milliseconds (50));
            }
            while (running);
        }
    private:
        mutex m_stopMutex;      // Mutex to synchronize the access to m_playbackRunning
        bool m_playbackRunning; // Shows if the playback is still running
    };
}
int main (int argc, char *argv[])
{
    // This is the data listener which will receive callbacks.  It's declared
    // before the cameraDevice so that, if this function exits with a 'return'
    // statement while the camera is still capturing, it will still be in scope
    // until the cameraDevice's destructor implicitly deregisters the listener.
    unique_ptr<MyListener> listener;
    // PlaybackStopListener which will be called as soon as the playback stops.
    MyPlaybackStopListener stopListener;
    // Royale's API treats the .rrf file as a camera, which it captures data from.
    unique_ptr<royale::ICameraDevice> cameraDevice;
    // check the command line for a given file
    if (argc < 2)
    {
        cout << "Usage " << argv[0] << " rrfFileToExport" << endl;
        cout << endl;
        cout << "Each frame of the recording is saved as a separate .ply file " << endl;
        cout << "in the current directory." << endl;
        return 1;
    }
    // Use the camera manager to open the recorded file, this block scope is because we can allow
    // the CameraManager to go out of scope once the file has been opened.
    {
        royale::CameraManager manager;
        // create a device from the file
        cameraDevice = manager.createCamera (argv[1]);
    }
    // if the file was loaded correctly the cameraDevice is now available
    if (cameraDevice == nullptr)
    {
        cerr << "Cannot load the file " << argv[1] << endl;
        return 1;
    }
    // cast the cameraDevice to IReplay which offers more options for playing
    // back recordings
    auto replayControls = dynamic_cast<royale::IReplay *> (cameraDevice.get());
    if (replayControls == nullptr)
    {
        cerr << "Unable to cast to IReplay interface" << endl;
        return 1;
    }
```

```
    // IMPORTANT: call the initialize method before working with the camera device
    if (cameraDevice->initialize() != royale::CameraStatus::SUCCESS)
    {
        cerr « "Cannot initialize the camera device" « endl;
        return 1;
    }
    // turn off the looping of the playback
    replayControls->loop (false);
    // Turn off the timestamps to speed up the conversion. If timestamps are enabled, an .rrf that
    // was recorded at 5FPS will generate callbacks to onNewData() at only 5 callbacks per second.
    replayControls->useTimestamps (false);
    // retrieve the total number of frames from the recording
    auto numFrames = replayControls->frameCount();
    // Create and register the data listener
    listener.reset (new MyListener (argv[1], numFrames));
    if (cameraDevice->registerDataListener (listener.get()) != royale::CameraStatus::SUCCESS)
    {
        cerr « "Error registering data listener" « endl;
        return 1;
    }
    // register a playback stop listener. This will be called as soon
    // as the file has been played back once (because loop is turned off)
    replayControls->registerStopListener (&stopListener);
    // start capture mode
    if (cameraDevice->startCapture() != royale::CameraStatus::SUCCESS)
    {
        cerr « "Error starting the capturing" « endl;
        return 1;
    }
    // block until the playback has finished
    stopListener.waitForStop();
    // stop capture mode
    if (cameraDevice->stopCapture() != royale::CameraStatus::SUCCESS)
    {
        cerr « "Error stopping the capturing" « endl;
        return 1;
    }
    return 0;
}
```

10.3 sampleIReplay.cpp

Class that can be used to get more control over the playback of a recording. To access the interface the ICameraDevice has to be casted to an IReplay object.

```
/****************************************************************************\
 * Copyright (C) 2019 Infineon Technologies & pmdtechnologies ag
 *
 * THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
 * KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A
 * PARTICULAR PURPOSE.
 *
 \****************************************************************************/
#include <iostream>
#include <memory>
#include <thread>
#include <chrono>
#include <royale.hpp>
#include <royale/IReplay.hpp>
namespace
{
    class MyPlaybackStopListener : public royale::IPlaybackStopListener
    {
        void onPlaybackStopped() override
        {
```

```
            std::cout « "received a stop signal from the IReplay interface\n";
        }
    };
    class MyDepthDataListener : public royale::IDepthDataListener
    {
        void onNewData (const royale::DepthData *data) override
        {
            std::cout « "received depth data with time_stamp: " « data->timeStamp.count() « '\n';
        };
    };
}
int main (int argc, char *argv[])
{
    // We receive our RRF-File from the command line.
    //
    if (argc < 2)
    {
        std::cerr « "Wrong usage of this sample! Please pass an RRF-File as first parameter.\n";
        return -1;
    }
    auto rrf_file = argv[1];
    // This represents the main camera device object.
    //
    std::unique_ptr<royale::ICameraDevice> camera;
    {
        // The camera manager can be created locally.
        // It is only used to create a camera device object
        // and can be destroyed afterwards.
        //
        royale::CameraManager manager{};
        camera = manager.createCamera (rrf_file);
    }
    // We have to check if the camera device was created successfully.
    //
    if (camera == nullptr)
    {
        std::cerr « "Can not create the camera! This may be caused by passing a bad RRF-File.\n";
        return -2;
    }
    // Before the camera device is ready we have to invoke initialize on it.
    //
    if (camera->initialize() != royale::CameraStatus::SUCCESS)
    {
        std::cerr « "Camera can not be initialized!\n";
        return -3;
    }
    // Now that the camera is ready we create our IDepthDataListener
    // and register it to the camera device.
    //
    MyDepthDataListener depth_data_listener{};
    camera->registerDataListener (&depth_data_listener);
    // As we know that the camera device was created using a RRF-File
    // we can expect that the underlying object implements IReplay.
    // To use this interface we can dynamic_cast the object to an IReplay object.
    //
    auto replay = dynamic_cast<royale::IReplay *> (camera.get());
    if (replay == nullptr)
    {
        std::cerr « "Can not cast the camera into an IReplay object!\n";
        return -4;
    }
    // Now that we have access to the IReplay interface we can register
    // our IReplayStopListener.
    //
    MyPlaybackStopListener playback_stop_listener{};
    replay->registerStopListener (&playback_stop_listener);
    // In the next step we set the current
    // frame of the replay object to its last one.
    // We also set the replay not to loop its frames.
    // This means that only the last frame will be played when the camera starts capturing.
```

```
    //
    auto frame_count = replay->frameCount();
    replay->seek (frame_count - 1);
    replay->loop (false);
    // Here starts the first demonstrating part of this sample.
    // If the replay behaves as we defined, the camera
    // will only capture one depth data before it stops capturing.
    //
    std::cout « "start playback...\n";
    camera->startCapture();
    std::this_thread::sleep_for (std::chrono::seconds{ 6 });
    camera->stopCapture();
    std::cout « "stopped playback.\n";
    // In the next step we set the current
    // frame of the replay object to its last one.
    // We also set the replay to loop its frames.
    // This means that the play will start at the same
    // frame as before but this time the replay will loop its content.
    //
    replay->seek (frame_count - 1);
    replay->loop (true);
    // Here starts the second demonstrating part of this sample.
    // If the replay behaves as we defined, the camera
    // will capture multiple depth data.
    //
    // We also pause and resume the playback here after 2 and 4 seconds.
    // Therefor the camera should capture depth data in the first two seconds,
    // then stop capture data for two seconds and
    // then continue capture data for two more seconds.
    //
    std::cout « "start playback...\n";
    camera->startCapture();
    std::this_thread::sleep_for (std::chrono::seconds{ 2 });
    replay->pause();
    std::this_thread::sleep_for (std::chrono::seconds{ 2 });
    replay->resume();
    std::this_thread::sleep_for (std::chrono::seconds{ 2 });
    camera->stopCapture();
    std::cout « "stopped playback.\n";
    // In the next step we set the current frame of the
    // replay object to its first one.
    // We also set the replay not to use timestamps.
    // This has the effect that the Replay will not try to
    // send the frames with the original fps but as fast as possible.
    //
    replay->seek (0);
    replay->useTimestamps (false);
    // Here starts the third demonstrating part of this sample.
    // If the replay behaves as we defined, the camera
    // will capture multiple depth data much faster as in the demonstration before.
    //
    std::cout « "start playback...\n";
    camera->startCapture();
    std::this_thread::sleep_for (std::chrono::seconds{ 2 });
    camera->stopCapture();
    std::cout « "stopped playback.\n";
    return 0;
}
```

10.4 sampleOpenCV.cpp

LEVEL 1 Gets the intrinsics of the camera module which are stored in the calibration file

Parameters

param	LensParameters is storing all the relevant information (c,f,p,k)

Returns

CameraStatus

```
/****************************************************************************\
 * Copyright (C) 2017 Infineon Technologies & pmdtechnologies ag
 *
 * THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
 * KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A
 * PARTICULAR PURPOSE.
 *
 \****************************************************************************/
#include <royale.hpp>
#include <iostream>
#include <mutex>
#include <opencv2/opencv.hpp>
#include <sample_utils/PlatformResources.hpp>
using namespace royale;
using namespace sample_utils;
using namespace std;
using namespace cv;
// Linker errors for the OpenCV sample
//
// If this example gives linker errors about undefined references to cv::namedWindow and cv::imshow,
// or QFontEngine::glyphCache and qMessageFormatString (from OpenCV to Qt), it may be caused by a
// change in the compiler's C++ ABI.
//
// With Ubuntu and Debian's distribution packages, the libopencv packages that have 'v5' at the end
// of their name, for example libopencv-video2.4v5, are compatible with GCC 5 (and GCC 6), but
// incompatible with GCC 4.8 and GCC 4.9. The -dev packages don't have the postfix, but depend on
// the v5 (or non-v5) version of the corresponding lib package.  When Ubuntu moves to OpenCV 3.0,
// they're likely to drop the postfix (but the packages will be for GCC 5 or later).
//
// If you are manually installing OpenCV or Qt, you need to ensure that the binaries were compiled
// with the same version of the compiler.  The version number of the packages themselves doesn't say
// which ABI they use, it depends on which version of the compiler was used.
class MyListener : public IDepthDataListener
{
public :
    MyListener() :
        undistortImage (false)
    {
    }
    void onNewData (const DepthData *data)
    {
        // this callback function will be called for every new
        // depth frame
        std::lock_guard<std::mutex> lock (flagMutex);
        // create two images which will be filled afterwards
        // each image containing one 32Bit channel
        zImage.create (Size (data->width, data->height), CV_32FC1);
        grayImage.create (Size (data->width, data->height), CV_32FC1);
        // set the image to zero
        zImage = Scalar::all (0);
        grayImage = Scalar::all (0);
        int k = 0;
        for (int y = 0; y < zImage.rows; y++)
        {
            float *zRowPtr = zImage.ptr<float> (y);
            float *grayRowPtr = grayImage.ptr<float> (y);
            for (int x = 0; x < zImage.cols; x++, k++)
            {
```

```
                auto curPoint = data->points.at (k);
                if (curPoint.depthConfidence > 0)
                {
                    // if the point is valid, map the pixel from 3D world
                    // coordinates to a 2D plane (this will distort the image)
                    zRowPtr[x] = adjustZValue (curPoint.z);
                    grayRowPtr[x] = adjustGrayValue (curPoint.grayValue);
                }
            }
        }
        // create images to store the 8Bit version (some OpenCV
        // functions may only work on 8Bit images)
        zImage8.create (Size (data->width, data->height), CV_8UC1);
        grayImage8.create (Size (data->width, data->height), CV_8UC1);
        // convert images to the 8Bit version
        // This sample uses a fixed scaling of the values to (0, 255) to avoid flickering.
        // You can also replace this with an automatic scaling by using
        // normalize(zImage, zImage8, 0, 255, NORM_MINMAX, CV_8UC1)
        // normalize(grayImage, grayImage8, 0, 255, NORM_MINMAX, CV_8UC1)
        zImage.convertTo (zImage8, CV_8UC1);
        grayImage.convertTo (grayImage8, CV_8UC1);
        if (undistortImage)
        {
            // call the undistortion function on the z image
            Mat temp = zImage8.clone();
            undistort (temp, zImage8, cameraMatrix, distortionCoefficients);
        }
        // scale and display the depth image
        scaledZImage.create (Size (data->width * 4, data->height * 4), CV_8UC1);
        resize (zImage8, scaledZImage, scaledZImage.size());
        imshow ("Depth", scaledZImage);
        if (undistortImage)
        {
            // call the undistortion function on the gray image
            Mat temp = grayImage8.clone();
            undistort (temp, grayImage8, cameraMatrix, distortionCoefficients);
        }
        // scale and display the gray image
        scaledGrayImage.create (Size (data->width * 4, data->height * 4), CV_8UC1);
        resize (grayImage8, scaledGrayImage, scaledGrayImage.size());
        imshow ("Gray", scaledGrayImage);
    }
    void setLensParameters (const LensParameters &lensParameters)
    {
        // Construct the camera matrix
        // (fx   0    cx)
        // (0    fy   cy)
        // (0    0    1 )
        cameraMatrix = (Mat1d (3, 3) << lensParameters.focalLength.first, 0,
       lensParameters.principalPoint.first,
                        0, lensParameters.focalLength.second, lensParameters.principalPoint.second,
                        0, 0, 1);
        // Construct the distortion coefficients
        // k1 k2 p1 p2 k3
        distortionCoefficients = (Mat1d (1, 5) << lensParameters.distortionRadial[0],
                                  lensParameters.distortionRadial[1],
                                  lensParameters.distortionTangential.first,
                                  lensParameters.distortionTangential.second,
                                  lensParameters.distortionRadial[2]);
    }
    void toggleUndistort()
    {
        std::lock_guard<std::mutex> lock (flagMutex);
        undistortImage = !undistortImage;
    }
private:
    // adjust z value to fit fixed scaling, here max dist is 2.5m
    // the max dist here is used as an example and can be modified
    float adjustZValue (float zValue)
    {
```

```
        float clampedDist = std::min (2.5f, zValue);
        float newZValue = clampedDist / 2.5f * 255.0f;
        return newZValue;
    }
    // adjust gray value to fit fixed scaling, here max value is 180
    // the max value here is used as an example and can be modified
    float adjustGrayValue (uint16_t grayValue)
    {
        float clampedVal = std::min (180.0f, grayValue * 1.0f);
        float newGrayValue = clampedVal / 180.f * 255.0f;
        return newGrayValue;
    }
    // define images for depth and gray
    // and for their 8Bit and scaled versions
    Mat zImage, zImage8, scaledZImage;
    Mat grayImage, grayImage8, scaledGrayImage;
    // lens matrices used for the undistortion of
    // the image
    Mat cameraMatrix;
    Mat distortionCoefficients;
    std::mutex flagMutex;
    bool undistortImage;
};
int main (int argc, char *argv[])
{
    // Windows requires that the application allocate these, not the DLL.
    PlatformResources resources;
    // This is the data listener which will receive callbacks.  It's declared
    // before the cameraDevice so that, if this function exits with a 'return'
    // statement while the camera is still capturing, it will still be in scope
    // until the cameraDevice's destructor implicitly de-registers the listener.
    MyListener listener;
    // this represents the main camera device object
    std::unique_ptr<ICameraDevice> cameraDevice;
    // the camera manager will query for a connected camera
    {
        CameraManager manager;
        // check the number of arguments
        if (argc > 1)
        {
            // if the program was called with an argument try to open this as a file
            cout « "Trying to open : " « argv[1] « endl;
            cameraDevice = manager.createCamera (argv[1]);
        }
        else
        {
            // if no argument was given try to open the first connected camera
            royale::Vector<royale::String> camlist (manager.getConnectedCameraList());
            cout « "Detected " « camlist.size() « " camera(s)." « endl;
            if (!camlist.empty())
            {
                cameraDevice = manager.createCamera (camlist[0]);
            }
            else
            {
                cerr « "No suitable camera device detected." « endl
                     « "Please make sure that a supported camera is plugged in, all drivers are "
                     « "installed, and you have proper USB permission" « endl;
                return 1;
            }
            camlist.clear();
        }
    }
    // the camera device is now available and CameraManager can be deallocated here
    if (cameraDevice == nullptr)
    {
        // no cameraDevice available
        if (argc > 1)
        {
            cerr « "Could not open " « argv[1] « endl;
```

```
            return 1;
        }
        else
        {
            cerr « "Cannot create the camera device" « endl;
            return 1;
        }
    }
    // IMPORTANT: call the initialize method before working with the camera device
    auto status = cameraDevice->initialize();
    if (status != CameraStatus::SUCCESS)
    {
        cerr « "Cannot initialize the camera device, error string : " « getErrorString (status) « endl;
        return 1;
    }
    // retrieve the lens parameters from Royale
    LensParameters lensParameters;
    status = cameraDevice->getLensParameters (lensParameters);
    if (status != CameraStatus::SUCCESS)
    {
        cerr « "Can't read out the lens parameters" « endl;
        return 1;
    }
    listener.setLensParameters (lensParameters);
    // register a data listener
    if (cameraDevice->registerDataListener (&listener) != CameraStatus::SUCCESS)
    {
        cerr « "Error registering data listener" « endl;
        return 1;
    }
    // create two windows
    namedWindow ("Depth", WINDOW_AUTOSIZE);
    namedWindow ("Gray", WINDOW_AUTOSIZE);
    // start capture mode
    if (cameraDevice->startCapture() != CameraStatus::SUCCESS)
    {
        cerr « "Error starting the capturing" « endl;
        return 1;
    }
    int currentKey = 0;
    while (currentKey != 27)
    {
        // wait until a key is pressed
        currentKey = waitKey (0) & 255;
        if (currentKey == 'd')
        {
            // toggle the undistortion of the image
            listener.toggleUndistort();
        }
    }
    // stop capture mode
    if (cameraDevice->stopCapture() != CameraStatus::SUCCESS)
    {
        cerr « "Error stopping the capturing" « endl;
        return 1;
    }
    return 0;
}
```

10.5 sampleRecordRRF.cpp

LEVEL 1 Start recording the raw data stream into a file.The recording will capture the raw data coming from the imager. If frameSkip and msSkip are both zero every frame will be recorded. If both are non-zero the behavior is implementation-defined.

Parameters

fileName	full path of target filename (proposed suffix is .rrf)
numberOfFrames	indicate the maximal number of frames which should be captured (stop will be called automatically). If zero (default) is set, recording will happen till stopRecording is called.
frameSkip	indicate how many frames should be skipped after every recorded frame. If zero (default) is set and msSkip is zero, every frame will be recorded.
msSkip	indicate how many milliseconds should be skipped after every recorded frame. If zero (default) is set and frameSkip is zero, every frame will be recorded.

```
/****************************************************************************\
 * Copyright (C) 2018 Infineon Technologies & pmdtechnologies ag
 *
 * THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
 * KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A
 * PARTICULAR PURPOSE.
 *
 \****************************************************************************/
#include <royale.hpp>
#include <sample_utils/PlatformResources.hpp>
#include <condition_variable>
#include <mutex>
#include <memory>
#include <iostream>
#include <cstdint>
// This is a standard implementation for handling an error on the camera device.
//
// In other applications it might be a better idea to retry some of the methods
// but for this sample this should be enough.
#define CHECKED_CAMERA_METHOD(METHOD_TO_INVOKE)\
do\
{\
    auto status = METHOD_TO_INVOKE;\
    if (royale::CameraStatus::SUCCESS != status)\
    {\
        std::cout << royale::getStatusString(status).c_str() << std::endl\
            << "Press Enter to exit the application ...";\
        \
        return -1;\
    }\
} while (0)
namespace
{
    std::mutex mtx;
    std::condition_variable condition;
    bool notified = false;
    class MyRecordListener : public royale::IRecordStopListener
    {
    public:
        void onRecordingStopped (const uint32_t numFrames) override
        {
            std::cout << "The onRecordingStopped was invoked with numFrames=" << numFrames << std::endl;
            // Notify the main method to return
            std::unique_lock<std::mutex> lock (mtx);
            notified = true;
            condition.notify_all();
        }
    };
    MyRecordListener myRecordListener;
}
int main (int argc, char **argv)
{
    // Receive the parameters to capture from the command line:
    if (2 > argc)
    {
```

```
    std::cout « "There are no parameters specified! Use:" « std::endl
             « argv[0] « " C:/path/to/file.rrf [numberOfFrames [framesToSkip [msToSkip]]]" « std::endl;
    return -1;
}
royale::String file (argv[1]);
auto numberOfFrames = argc >= 3 ? std::stoi (argv[2]) : 0;
auto framesToSkip = argc >= 4 ? std::stoi (argv[3]) : 0;
auto msToSkip = argc >= 5 ? std::stoi (argv[4]) : 0;
// If the first argument was "--help", don't treat that as a filename
if (file == "--help" || file == "-h" || file == "/?")
{
    std::cout « argv[0] « ":" « std::endl;
    std::cout « "--help, -h, /? : this help" « std::endl;
    std::cout « std::endl;
    std::cout « argv[0] « " C:/path/to/file.rrf [numberOfFrames [framesToSkip [msToSkip]]]" « std::endl;
    std::cout « "Record to the given RRF file, overwriting it if it already exists" « std::endl;
    return 0;
}
// Print the parsed parameters to the command line
std::cout « "start recording using following parameters:" « std::endl
          « "file=" « file « std::endl
          « "numberOfFrames=" « numberOfFrames « std::endl
          « "framesToSkip=" « framesToSkip « std::endl
          « "msToSkip=" « msToSkip « std::endl;
// Windows requires that the application allocate these, not the DLL.
sample_utils::PlatformResources platformResources;
std::unique_ptr<royale::ICameraDevice> cameraDevice;
// The camera manager will query for a connected camera.
{
    royale::CameraManager manager;
    auto connectedCameraList = manager.getConnectedCameraList();
    if (0 >= connectedCameraList.count())
    {
        std::cout « "There is no camera connected!" « std::endl;
        return -1;
    }
    cameraDevice = manager.createCamera (connectedCameraList[0]);
}
// The camera device is now available and CameraManager can be deallocated here.
if (nullptr == cameraDevice)
{
    std::cout « "The camera can not be created!" « std::endl;
    return -1;
}
// IMPORTANT: call the initialize method before working with the camera device
CHECKED_CAMERA_METHOD (cameraDevice->initialize());
// If the user specified a number of frames to capture we use a listener to tidy up
// the camera afterwards.
if (0 < numberOfFrames)
{
    CHECKED_CAMERA_METHOD (cameraDevice->registerRecordListener (&myRecordListener));
    CHECKED_CAMERA_METHOD (cameraDevice->startCapture());
    CHECKED_CAMERA_METHOD (cameraDevice->startRecording (file, static_cast<uint32_t> (numberOfFrames),
                                                         static_cast<uint32_t> (framesToSkip),
   msToSkip));
    // It is important not to close the main method before
    // the record stop callback was received!
    std::unique_lock<std::mutex> lock (mtx);
    // loop to avoid spurious wakeups
    while (!notified)
    {
        condition.wait (lock);
    }
    auto status = cameraDevice->stopCapture();
    if (royale::CameraStatus::SUCCESS != status)
    {
        std::cout « "Failed to close the camera device with status="
                 « royale::getStatusString (status).c_str()
                 « std::endl;
    }
```

```
    }
    // We stop the recording on key press if the user did not limit the number of frames to capture.
    else
    {
        CHECKED_CAMERA_METHOD (cameraDevice->startCapture());
        CHECKED_CAMERA_METHOD (cameraDevice->startRecording (file, static_cast<uint32_t> (numberOfFrames),
                                                             static_cast<uint32_t> (framesToSkip),
       msToSkip));
        std::cout « "Press Enter to stop capture ..." « std::endl;
        std::cin.get();
        CHECKED_CAMERA_METHOD (cameraDevice->stopRecording());
        CHECKED_CAMERA_METHOD (cameraDevice->stopCapture());
    }
    return 0;
}
```

10.6 sampleRetrieveData.cpp

LEVEL 1 Once registering the data listener, 3D point cloud data is sent via the callback function.

Parameters

listener	interface which needs to implement the callback method

```
/****************************************************************************\
 * Copyright (C) 2017 Infineon Technologies & pmdtechnologies ag
 *
 * THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
 * KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A
 * PARTICULAR PURPOSE.
 *
 \****************************************************************************/
#include <royale.hpp>
#include <algorithm>
#include <iomanip>
#include <iostream>
#include <thread>
#include <mutex>
#include <chrono>
#include <sample_utils/PlatformResources.hpp>
using namespace sample_utils;
using namespace std;
class MyListener : public royale::IDepthDataListener
{
    static const size_t MAX_HEIGHT = 40;
    static const size_t MAX_WIDTH = 76;
    char asciiPoint (const royale::DepthData *data, std::size_t x, std::size_t y)
    {
        // Using a bounds-check here is inefficient, but allows the scale-to-max-width-or-height
        // loop in onNewData to be simpler.
        if (x >= data->width || y >= data->height)
        {
            return ' ';
        }
        auto depth = data->points[y * data->width + x];
        if (depth.depthConfidence < 128)
        {
            return ' ';
        }
        const royale::Vector<royale::Pair<float, char» DEPTH_LETTER
        {
```

```
            {0.25f, 'O'},
            {0.30f, 'o'},
            {0.35f, '+'},
            {0.40f, '-'},
            {1.00f, '.'},
        };
        for (auto i : DEPTH_LETTER)
        {
            if (depth.z < i.first)
            {
                return i.second;
            }
        }
        return ' ';
    }
    struct MyFrameData
    {
        std::vector<uint32_t> exposureTimes;
        std::vector<std::string> asciiFrame;
    };
public:
    void onNewData (const royale::DepthData *data) override
    {
        // Demonstration of how to retrieve the exposure times for the current stream. This is a
        // vector which can contain several numbers, because the depth frame is calculated from
        // several individual raw frames.
        auto exposureTimes = data->exposureTimes;
        // The data pointer will become invalid after onNewData returns.  When processing the data,
        // it's necessary to either:
        // 1. Do all the processing before this method returns, or
        // 2. Copy the data (not just the pointer) for later processing, or
        // 3. Do processing that needs the full data here, and generate or copy only the data
        //    required for later processing
        //
        // The Royale library's depth-processing thread may block while waiting for this function to
        // return; if this function is slow then there may be some lag between capture and onNewData
        // for the next frame.  If it's very slow then Royale may drop frames to catch up.
        //
        // This sample code assumes that the UI toolkit will provide a callback to the paint()
        // method as needed, but does the initial processing of the data in the current thread.
        //
        // This sample code uses option 3 above, the conversion from DepthData to MyFrameData is
        // done in this method, and MyFrameData provides all the data required for the paint()
        // method, without needing any pointers to the memory owned by the DepthData instance.
        //
        // The image will also be scaled to the expected width, or may be scaled down to less than
        // the expected width if necessary for the height limitation.
        std::size_t scale = std::max ({size_t (1u), data->width / m_widthPerStream, data->height /
       MAX_HEIGHT});
        std::size_t height = data->height / scale;
        // To reduce the depth data to ascii art, this sample discards most of the information in
        // the data.  However, even if we had a full GUI, treating the depth data as an array of
        // (data->width * data->height) z coordinates would lose the accuracy.  The 3D depth
        // points are not arranged in a rectilinear projection, and the discarded x and y
        // coordinates from the depth points account for the optical projection (or optical
        // distortion) of the camera.
        std::vector<std::string> asciiFrame;
        for (auto y = 0u; y < height; y++)
        {
            std::string asciiLine;
            asciiLine.reserve (m_widthPerStream);
            for (auto x = 0u; x < m_widthPerStream; x++)
            {
                // There is a bounds-check in the asciiPoint function, it returns a space character
                // if x or y is out-of-bounds.
                asciiLine.push_back (asciiPoint (data, x * scale, y * scale));
            }
            asciiFrame.push_back (std::move (asciiLine));
        }
        // Scope for a lock while updating the internal model
```

```
    {
        std::unique_lock<std::mutex> lock (m_lockForReceivedData);
        auto &receivedData = m_receivedData[data->streamId];
        receivedData.asciiFrame.swap (asciiFrame);
        receivedData.exposureTimes = exposureTimes.toStdVector();
    }
    // In a full application, the call below to paint() should be done in a separate thread, as
    // explained in the comment above.  UI toolkits are expected to provide a method to
    // request an asynchronous repaint of the screen, without blocking onNewData().
    //
    // But for the purposes of a demo, dropped frames are an accepted trade-off for not
    // depending on a specific UI toolkit.
    paint();
}
void paint ()
{
    // to show multiple streams side-by-size, we temporarily blit them in to this area
    std::vector<std::string> allFrames;
    std::map<royale::StreamId, std::vector<uint32_t» allExposureTimes;
    // scope for the thread-safety lock
    {
        // while locked, copy the data to local structures
        std::unique_lock<std::mutex> lock (m_lockForReceivedData);
        if (m_receivedData.empty())
        {
            return;
        }
        // allocate the allFrames area, with a set of empty strings
        {
            const auto received = m_receivedData.begin();
            allFrames.resize (received->second.asciiFrame.size());
            for (auto &str : allFrames)
            {
                str.reserve (m_width);
            }
        }
        // For each stream, append its data to each line of allFrames
        for (auto streamId : m_streamIds)
        {
            const auto received = m_receivedData.find (streamId);
            if (received != m_receivedData.end())
            {
                for (auto y = 0u; y < allFrames.size(); y++)
                {
                    // The at() in the next line could throw if streams have different-sized
                    // frames. That situation isn't expected in Royale, so this example omits
                    // error handling for this.
                    const auto &src = received->second.asciiFrame.at (y);
                    auto &dest = allFrames.at (y);
                    dest.append (src.begin(), src.end());
                }
                allExposureTimes[streamId] = received->second.exposureTimes;
            }
            else
            {
                // There's no capture for this stream yet, leave a blank space
                for (auto &dest : allFrames)
                {
                    dest.append (m_widthPerStream, ' ');
                }
            }
        }
    }
    // If this is a mixed mode (with multiple streams), print a header with the StreamIds
    if (m_streamIds.size() > 1)
    {
        for (auto streamId : m_streamIds)
        {
            // streamIds are uint16_ts.  To print them with C++'s '«' operator they should be
            // converted to unsigned, otherwise they may be interpreted as wide-characters.
```

```
                cout « setw (static_cast<int> (m_widthPerStream)) « setfill (' ') « unsigned (streamId);
            }
            cout « endl;
        }
        // Print the data from all of the captured streams
        for (const auto &line : allFrames)
        {
            cout « string (line.data(), line.size()) « endl;
        }
        for (auto streamId : m_streamIds)
        {
            const auto &exposureTimes = allExposureTimes.find (streamId);
            cout « "ExposureTimes for stream[" « static_cast<unsigned> (streamId) « "] : ";
            if (exposureTimes == allExposureTimes.end())
            {
                cout « "no frames received yet for this stream";
            }
            else
            {
                bool firstElementOfList = true;
                for (const auto exposure : exposureTimes->second)
                {
                    if (firstElementOfList)
                    {
                        firstElementOfList = false;
                    }
                    else
                    {
                        cout « ", ";
                    }
                    cout « exposure;
                }
            }
            cout « endl;
        }
    }
    explicit MyListener (const royale::Vector<royale::StreamId> &streamIds) :
        m_width (MAX_WIDTH),
        m_widthPerStream (MAX_WIDTH / streamIds.size()),
        m_streamIds (streamIds)
    {
    }
private:
    const std::size_t m_width;
    const std::size_t m_widthPerStream;
    const royale::Vector<royale::StreamId> m_streamIds;
    std::map<royale::StreamId, MyFrameData> m_receivedData;
    std::mutex m_lockForReceivedData;
};
int main (int argc, char **argv)
{
    // Windows requires that the application allocate these, not the DLL.  We expect typical
    // Royale applications to be using a GUI toolkit such as Qt, which has its own equivalent of this
    // PlatformResources class (automatically set up by the toolkit).
    PlatformResources resources;
    // This is the data listener which will receive callbacks.  It's declared
    // before the cameraDevice so that, if this function exits with a 'return'
    // statement while the camera is still capturing, it will still be in scope
    // until the cameraDevice's destructor implicitly deregisters the listener.
    unique_ptr<MyListener> listener;
    // this represents the main camera device object
    unique_ptr<royale::ICameraDevice> cameraDevice;
    // if non-null, load this file instead of connecting to a camera
    unique_ptr<royale::String> rrfFile;
    // if non-null, choose this use case instead of the default
    unique_ptr<royale::String> commandLineUseCase;
    if (argc > 1)
    {
        // Files recorded with RoyaleViewer have this filename extension
        const auto FILE_EXTENSION = royale::String (".rrf");
```

```
    auto arg = std::unique_ptr<royale::String> (new royale::String (argv[1]));
    if (*arg == "--help" || *arg == "-h" || *arg == "/?" || argc > 2)
    {
        cout « argv[0] « ":" « endl;
        cout « "--help, -h, /? : this help" « endl;
        cout « endl;
        cout « "With no command line arguments: opens a camera and retrieves data using the default use
    case" « endl;
        cout « endl;
        cout « "With an argument that ends \".rrf\": assumes the argument is the filename of a
    recording, and plays it" « endl;
        cout « "When playing back a recording, the only use case available is the one that was
    recorded." « endl;
        cout « endl;
        cout « "With an argument that doesn't end \".rrf\": assumes the argument is the name of a
    use-case" « endl;
        cout « "It will open the camera, and if there's a use case with that exact name will use it." «
    endl;
        return 0;
    }
    else if (arg->size() > FILE_EXTENSION.size() && 0 == arg->compare (arg->size() -
    FILE_EXTENSION.size(), FILE_EXTENSION.size(), FILE_EXTENSION))
    {
        cout « "Assuming command-line argument is the filename of an RRF recording" « endl;
        rrfFile = std::move (arg);
    }
    else
    {
        cout « "Assuming command-line argument is the name of a use case, as it does not end \".rrf\"" «
    endl;
        commandLineUseCase = std::move (arg);
    }
}
// the camera manager will either open a recording or query for a connected camera
if (rrfFile)
{
    royale::CameraManager manager;
    cameraDevice = manager.createCamera (*rrfFile);
}
else
{
    royale::CameraManager manager;
    auto camlist = manager.getConnectedCameraList();
    cout « "Detected " « camlist.size() « " camera(s)." « endl;
    if (!camlist.empty())
    {
        cout « "CamID for first device: " « camlist.at (0).c_str() « " with a length of (" « camlist.at
    (0).length() « ")" « endl;
        cameraDevice = manager.createCamera (camlist[0]);
    }
}
// the camera device is now available and CameraManager can be deallocated here
if (cameraDevice == nullptr)
{
    cerr « "Cannot create the camera device" « endl;
    return 1;
}
// IMPORTANT: call the initialize method before working with the camera device
if (cameraDevice->initialize() != royale::CameraStatus::SUCCESS)
{
    cerr « "Cannot initialize the camera device" « endl;
    return 1;
}
royale::Vector<royale::String> useCases;
auto status = cameraDevice->getUseCases (useCases);
if (status != royale::CameraStatus::SUCCESS || useCases.empty())
{
    cerr « "No use cases are available" « endl;
    cerr « "getUseCases() returned: " « getErrorString (status) « endl;
    return 1;
```

```
}
// choose a use case
auto selectedUseCaseIdx = 0u;
if (commandLineUseCase)
{
    auto useCaseFound = false;
    for (auto i = 0u; i < useCases.size(); ++i)
    {
        if (*commandLineUseCase == useCases[i])
        {
            selectedUseCaseIdx = i;
            useCaseFound = true;
            break;
        }
    }
    if (!useCaseFound)
    {
        cerr « "Error: the chosen use case is not supported by this camera" « endl;
        cerr « "A list of supported use cases is printed by sampleCameraInfo" « endl;
        return 1;
    }
}
else
{
    // choose the first use case
    selectedUseCaseIdx = 0;
}
// set an operation mode
if (cameraDevice->setUseCase (useCases.at (selectedUseCaseIdx)) != royale::CameraStatus::SUCCESS)
{
    cerr « "Error setting use case" « endl;
    return 1;
}
// Retrieve the IDs of the different streams
royale::Vector<royale::StreamId> streamIds;
if (cameraDevice->getStreams (streamIds) != royale::CameraStatus::SUCCESS)
{
    cerr « "Error retrieving streams" « endl;
    return 1;
}
cout « "Stream IDs : ";
for (auto curStream : streamIds)
{
    cout « curStream « ", ";
}
cout « endl;
// register a data listener
listener.reset (new MyListener (streamIds));
if (cameraDevice->registerDataListener (listener.get()) != royale::CameraStatus::SUCCESS)
{
    cerr « "Error registering data listener" « endl;
    return 1;
}
// start capture mode
if (cameraDevice->startCapture() != royale::CameraStatus::SUCCESS)
{
    cerr « "Error starting the capturing" « endl;
    return 1;
}
// let the camera capture for some time
this_thread::sleep_for (chrono::seconds (5));
// Change the exposure time for the first stream of the use case (Royale will limit this to an
// eye-safe exposure time, with limits defined by the use case).  The time is given in
// microseconds.
//
// Non-mixed mode use cases have exactly one stream, mixed mode use cases have more than one.
// For this example we only change the first stream.
if (cameraDevice->setExposureTime (200, streamIds[0]) != royale::CameraStatus::SUCCESS)
{
    cerr « "Cannot set exposure time for stream" « streamIds[0] « endl;
```

```
    }
    else
    {
        cout « "Changed exposure time for stream " « streamIds[0] « " to 200 microseconds ..." « endl;
    }
    // let the camera capture for some time
    this_thread::sleep_for (chrono::seconds (5));
    // stop capture mode
    if (cameraDevice->stopCapture() != royale::CameraStatus::SUCCESS)
    {
        cerr « "Error stopping the capturing" « endl;
        return 1;
    }
    return 0;
}
```


infineon
pmd
can you imagine

Index