Argus API Extensions 0.99.2

January 15, 2023

Contents

List of Tables

1 BayerAverageMap

BayerAverageMap generates local averages of a capture's raw Bayer data. These averages are generated from small rectangles, called bins, that are evenly distributed across the image.

This extension introduces two new interfaces:

1.1 IBayerAverageMapSettings

Provides methods to set bayer average map settings. **IBayerAverageMapSettings** defines two methods:

setBayerAverageMapEnable()
Enables or disables Bayer average map generation. When enabled, **CaptureMetadata** returned by completed captures will expose the **IBayerAverageMap** interface.

getBayerAverageMapEnable()
Returns whether or not Bayer average map generation is enabled.

1.2 IBayerAverageMap

Provides methods to get Bayer average map metadata. Each average is a floating-point value that is normalized such that [0.0, 1.0] maps to the full optical range of the output pixels, however values outside this range may be included in the averages so long as they are within the working range of the average calculation. For pixels that have values outside the working range, the API excludes such pixels from the average calculation and increments the clipped pixel counter for the containing region.

The **IBayerAverageMap** interface supports the following methods:

getBinStart()
Returns the starting location of the first bin, in pixels, where the location is relative to the top-left corner of the image.

getBinSize()
Returns the size of each bin, in pixels.

getBinCount()
Returns the number of bins in both the horizontal (width) and vertical (height) directions.

getBinInterval()
Returns the bin intervals for both the x and y axis, defined as the number of pixels between the first pixel of a bin and that o f the adjacent bin.

getWorkingRange()
Returns the range of values that are included in the average calculation (not clipped), and may extend beyond the normalized [0.0, 1.0] range of the optical output.

getAverages()
Returns the average values for all bins normalized such that [0.0, 1.0] maps to the optical range of the output. Argus excludes input pixels that have values outside the working range **getWorkingRange()** from the average calculation, and counts them as clipped pixels **getClipCounts()**.

getClipCounts()
Returns, for all bins, the number of pixels whose value exceeds the working range and have been excluded from average calculation.

2 BayerSharpnessMap

BayerSharpnessMap generates sharpness metrics that can be used to determine the correct position of the lens to achieve the best focus.

This extension introduces two new interfaces:

2.1 IBayerSharpnessMapSettings

Provides methods to set Bayer sharpness map. **IBayerSharpnessMapSettings** defines two methods:

setBayerSharpnessMapEnable()
Enables or disables Bayer sharpness map generation. When enabled, **CaptureMetadata** returned by completed captures will expose the **IBayerSharpnessMap** interface.

getBayerSharpnessMapEnable()
Returns whether or not Bayer sharpness map generation is enabled.

2.2 IBayerSharpnessMap

Provides methods to get Bayer sharpness metrics. Each metric is a normalized floating-point value representing the estimated sharpness for a particular color channel and pixel region, called bins, where 0.0 and 1.0 map to the minimum and maximum possible sharpness values, respectively.

The **IBayerSharpnessMap** interface supports the following methods:

getBinStart()
Returns the starting location of the first bin, in pixels, where the location is relative to the top-left corner of the image.

getBinSize()
Returns the size of each bin, in pixels.

getBinCount()
Returns the number of bins in both the horizontal (width) and vertical (height) directions.

getBinInterval()
Returns the bin intervals for both the x and y axis, defined as the number of pixels between the first pixel of a bin and that of the adjacent bin.

getSharpnessValue()
Returns the sharpness values for all bins and color channels.

3 BlockingSessionCameraProvider

BlockingSessionCameraProvider provides a method to create a blocking capture session.

This extension introduces a new interface:

3.1 IBlockingSessionCameraProvider

Provides methods used to create blocking capture session.

The **IBlockingSessionCameraProvider** interface supports the following methods:

createBlockingCaptureSession()
Creates and returns a blocking CaptureSession using the given device. For blocking CaptureSession, the capture related API call will block wait until the request is serviced by the underlying driver. This will help timing control in client side when client auto control is involved.

4 NonLinearHistogram

NonLinearHistogram provides a method to interpret the compressed histogram data correctly.

This extension introduces one new interface:

4.1 INonLinearHistogram

This interface returns the normalized bin values to correctly interpret the compressed bayer histogram data. This interface is available from histogram child objects returned by **ICaptureMetadata::getBayerHistogr**

INonLinearHistogram defines following method:

getHistogramBinValues()
Returns a floating point vector having normalized bins which maps the bin indexes of histogram **IBayerHistogram::getHistogram()**to the actual pixel values after considering compression. This vector has the same count as the vector returned by **IBayerHistogram::getBinCoun**
For Example:

```
IBayerHistogram->getHistogram(&histogram);
INonLinearHistogram->getBinValues(&values);
for(int i = 0 ; i < histogram.size() ; i++)
{
        cout<<" bin: " << i
        <<" normalized bin Value: " << values[i]
        <<" frequency: " << histogram[i];
}
```

5 PwlWdrSensorMode

PwlWdrSensorMode provideds extra functionalities for the Piecewise Linear (PWL) Wide Dynamic Range (WDR) sensor mode type.

This extension introduces one new interface:

5.1 IPwlWdrSensorMode

This interface returns a list of normalized float coordinates (x,y) that defines the PWL compression curve used in the PWL WDR mode. This PWL compression curve is used by the sensor to compress WDR pixel values before sending them over CSI. This is done to save bandwidth for data transmission over VI-CSI. The compression converts the WDR pixel values from InputBitDepth space to OutputBitDepth space. The Bit depths can be obtained by using the respective methods in the ISensorMode interface. @see ISensorMode

IPwlWdrSensorMode defines the following methods:

getControlPointCount()
Returns the number of control points coordinates in the Piecewise Linear compression curve.

getControlPoints()
Returns the Piecewise Linear (PWL) compression curve coordinates.

6 DolWdrSensorMode

DolWdrSensorMode provides extra functionalities for the Digital Overlap (DOL) Wide Dynamic Range (WDR) sensor mode type.

This extension introduces one new interface:

6.1 IDolWdrSensorMode

This interface returns the extended properties specific to DOL WDR sensor mode. DOL WDR is a multi-exposure technology that enables the fusion of various exposures from a single frame to produce a WDR image. Basic sensor mode properties are available through the **ISensorMode** interface.

IDolWdrSensorMode defines the following methods:

getExposureCount()
: Returns the number of exposures captured per frame for this DOL WDR mode.

getOpticalBlackRowCount()
: Returns the number of Optical Black rows at the start of each exposure in a DOL WDR frame.

getVerticalBlankPeriodRowCount()
: Populates the verticalBlankPeriodRowCounts vector to store the vertical blank period rows per DOL WDR exposure. Size of the vector is **getExposureCount()** - 1 count values.

getLineInfoMarkerWidth()
: Returns the line info markers width in pixels. These occur at the start of each pixel row to distinguish row types.

getLeftMarginWidth()
: Returns the number of margin pixels on the left per row.

getRightMarginWidth()
: Returns the number of margin pixels on the right per row.

getPhysicalResolution()
: Returns the physical resolution derived due to interleaved exposure output from DOL WDR frames.

7 DebugCaptureSession

DebugCaptureSession provides a method to dump internal libargus runtime information.

This extension introduces one new interface:

7.1 IDebugCaptureSession

This interface is used to dump CaptureSession runtime information.

IDebugCaptureSession defines the following methods:

dump()
: Returns session runtime information to the specified file descriptor.

setEventInjectionErrorMsg()
: Set event injection error id.

8 InternalFrameCount

InternalFrameCount provides accessors for an internal frame count performance metric. The "internal frame count" is an implementation-dependent metric that may be used to detect performance issues and producer frame drops for libargus which makes use of internal captures.

When a device is opened by a CaptureSession, frames may be captured and processed by libargus when the client is idle or not ready for frame output in order to maintain the device driver subsystem and/or auto-control state (exposure, white balance, etc). These captures are started and processed entirely within the libargus implementation, with no inputs from or outputs to the client application, and so are referred to as "internal" captures. These internal captures are typically submitted when there are no client requests in the capture queue or no stream buffers available for output within a sensor frame period, so knowing when an internal capture has been submitted can be used to detect application or performance issues in cases where these conditions are not expected to occur.

This extension provides internal capture information in the form of an "internal frame count", which is the total number of captures submitted by the session including both the internal captures as well as client-submitted requests. If an internal frame count gap appears between two client-submitted captures, this means that one or more internal captures have been performed.

This interface is available from CaptureMetadata objects.

This extension introduces one new interface:

8.1 IInternalFrameCount

This interface is used to query the internal frame count for a client request. Since internal captures do not generate events, detecting internal captures must be done by comparing the internal capture count of successive client-submitted capture requests.

IInternalFrameCount defines the following methods:

getInternalFrameCount()
Returns the internal frame count for the request.

9 SensorEepromData

SensorEepromData provides an interface to get EEPROM data.

This extension introduces one new interface:

9.1 ISensorEepromData

This interface is used to get EEPROM data.

ISensorEepromData defines the following methods:

getSensorEepromDataSize()
Returns the size of the EEPROM data.

getSensorEepromData()
Copies back the EEPROM data to the provided memory location. The maximum supported size of EEPROM data that can be read is 1024.

10 SensorOtpData

SensorOtpData provides an interface to get OTP data.

This extension introduces one new interface:

10.1 ISensorOtpData

This interface is used to get OTP data.

ISensorOtpData defines the following methods:

getSensorOtpDataSize()
Returns the size of the OTP data.

getSensorOtpData()
Copies back the OTP data to the provided memory location. The maximum supported size of OTP data that can be read is 512.

11 SensorPrivateMetadata

SensorPrivateMetadata provides accessors for sensor embedded metadata. This data is metadata that the sensor embeds inside the frame, the type and formatting of which depends on the sensor. It is up to the user to correctly parse the data based on the specifics of the sensor used.

This extension introduces new interfaces:

11.1 ISensorPrivateMetadataCaps

This interface used to query the availability and size in bytes of sensor private metadata.

ISensorPrivateMetadataCaps defines the following methods:

getMetadataSize()
Returns the size in bytes of the private metadata.

11.2 ISensorPrivateMetadataRequest

This interface enables private metadata output from a capture request.

ISensorPrivateMetadataRequest defines the following methods:

setMetadataEnable()
Enables the sensor private metadata, will only work if the sensor supports embedded metadata.

getMetadataEnable()
Returns if the metadata is enabled for this request.

11.3 ISensorPrivateMetadata

This interface used to access sensor private metadata.

ISensorPrivateMetadata defines the following methods:

getMetadataSize()
Returns the size of the embedded metadata.

getMetaData()
Copies back the metadata to the provided memory location.

12 SensorPrivateMetadataClientBuffer

SensorPrivateMetadataClientBuffer provides accessors for set client buffer for sensor embedded metadata. Sensor embeds private information (for example PDAF data) in sensor metadata. Client using **ISensorPrivateMetadata** API to obtain this metadata involves several memcpy. When sensor metadata size is large, this will cause high CPU usage and affect camera performance.

This extension introduces one new interface:

12.1 ISensorPrivateMetadataClientBufferRequest

This interface allows client to set a client buffer and Argus writes to it directly without extra memcpy. This is only supported in single process mode as in client-server (multiprocess) mode, client and server are in different process and their own address space.

ISensorPrivateMetadataClientBufferRequest defines the following methods:

setClientMetadataBuffer()
Client allocates the metadata buffer and sets the address of the buffer, Argus writes the sensor metadata directly to it. Only supported in single process mode.

getClientMetadataBufferEnable()
Returns if client metadata buffer is used for this request.

13 SensorTimestampTsc

SensorTimestampTsc provides a timestamp interface to get tegra wide timestamp system counter (TSC) HW timestamp.

This extension introduces one new interface:

13.1 ISensorTimestampTsc

This interface is used to get TSC HW timestamp.

ISensorTimestampTsc defines the following methods:

getSensorSofTimestampTsc()
Returns the VI HW start of frame (SOF) timestamp based on tegra wide timestamp system counter (TSC). This is the start timestamp for the sensor (in nanoseconds).

getSensorEofTimestampTsc()
Returns the VI HW end of frame (EOF) timestamp based on tegra wide timestamp system counter (TSC). This is the end timestamp for the sensor (in nanoseconds).

14 SyncSensorCalibrationData

SyncSensorCalibrationData provides access to sync sensor calibration data.

This extension introduces one new interface:

14.1 ISyncSensorCalibrationData

This interface is used to access sync sensor calibration data.

ISyncSensorCalibrationData defines the following methods:

getSyncSensorModuleId()
Returns the sync sensor module id in the provided memory location. The maximum supported length of sync sensor id string is 32.

getImageSizeInPixels()
Returns the size of the image in pixels.

getFocalLength()
Returns the focal length fx and fy from intrinsic parameters.

getSkew()
Returns the skew from intrinsic parameters.

getPrincipalPoint()
Returns the principal point (optical center) x and y from intrinsic parameters.

getLensDistortionType()
Returns the lens distortion type as per the model being used.

getFisheyeMappingType()
Returns the mapping type in case of fisheye distortion.

getRadialCoeffsCount()
Returns the radial coefficients count in case of polynomial or fisheye distortion.

getRadialCoeffs()
Returns the radial coefficients vector as per distortion type and size of the vector is given by getRadialCoeffsCount().

getTangentialCoeffsCount()
Returns the tangential coefficients count in case of polynomial distortion.

getTangentialCoeffs()
Returns the tangential coefficients in case of polynomial distortion and size of the vector is given by getTangentialCoeffsCount().

getRotationParams()
Returns the rotation parameter expressed in Rodrigues notation from extrinsic parameters.

getTranslationParams()
Returns the translation parameters in x, y and z co-ordinates with respect to a reference point from extrinsic parameters.

getModuleSerialNumber()
Returns the serial number of the sensor module in the provided memory location.

isImuSensorAvailable()
Returns whether IMU sensor is present or not.

getLinearAccBias()
Returns the linear acceleration bias for all three axes x, y and z of the IMU device.

getAngularVelocityBias()
Returns the angular velocity bias for all three axes x, y and z of the IMU device.

getGravityAcc()
Returns the gravity acceleration for all three axes x, y and z of the IMU device.

getImuRotationParams()
Returns the IMU rotation parameter expressed in Rodrigues notation from extrinsic parameters.

getImuTranslationParams()
Returns the IMU translation parameters in x, y and z co-ordinates with respect to a reference point from extrinsic parameters.

getUpdateRate()
Returns the update rate.

getLinearAccNoiseDensity
Returns the linear acceleration noise density.

getLinearAccRandomWalk
Returns the linear acceleration random walk.

getAngularVelNoiseDensity
Returns the angular velocity noise density.

getLinearAccNoiseDensity
Returns the angular velocity random walk.