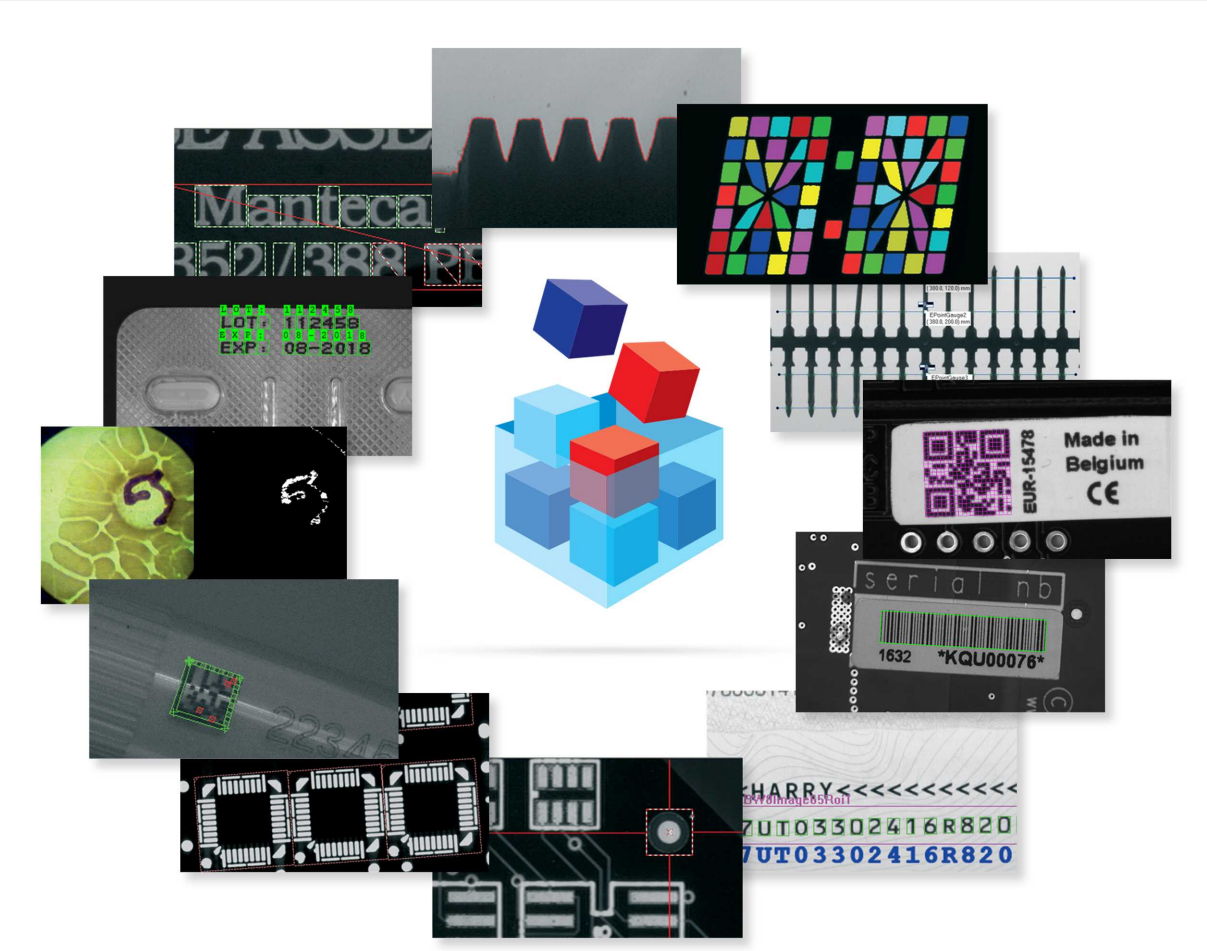
EURESYS™
Excellence in vision

REFERENCE MANUAL

# Open eVision



## Terms of Use

EURESYS s.a. shall retain all property rights, title and interest of the documentation of the hardware and the software, and of the trademarks of EURESYS s.a.

All the names of companies and products mentioned in the documentation may be the trademarks of their respective owners.

The licensing, use, leasing, loaning, translation, reproduction, copying or modification of the hardware or the software, brands or documentation of EURESYS s.a. contained in this book, is not allowed without prior notice.

EURESYS s.a. may modify the product specification or change the information given in this documentation at any time, at its discretion, and without prior notice.

EURESYS s.a. shall not be liable for any loss of or damage to revenues, profits, goodwill, data, information systems or other special, incidental, indirect, consequential or punitive damages of any kind arising in connection with the use of the hardware or the software of EURESYS s.a. or resulting of omissions or errors in this documentation.

This documentation is provided with Open eVision 2.10.0 (doc build 1121).

# Contents

# 1. Pixel Accessors

## 1.1. EBW8PixelAccessor Class

-

**Namespace:** Euresys::Open_eVision_1_2

### Methods

| | |
|---|---|
| EBW8PixelAccessor | - |
| GetPixel | - |
| SetPixel | E- |

B

## W8PixelAccessor::EBW8PixelAccessor

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
void EBW8PixelAccessor(
 EROIBW8& roi
 )
```

### Parameters

*roi*

-

# EBW8PixelAccessor::GetPixel

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
OEV_UINT8 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

-

*y*

-

# EBW8PixelAccessor::SetPixel

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
void SetPixel(
 OEV_UINT8 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

-

*x*

-

*y*

-

# 1.2. EBW16PixelAccessor Class

-

**Namespace:** Euresys::Open_eVision_1_2

## Methods

| | |
|---|---|
| EBW16PixelAccessor | - |
| GetPixel | - |
| SetPixel | E- |

B

W16PixelAccessor::EBW16PixelAccessor

-

**Namespace:** Euresys::Open_eVision_1_2

[C++]

```
void EBW16PixelAccessor(
   EROIBW16& roi
)
```

### Parameters

*roi*

-

# EBW16PixelAccessor::GetPixel

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
OEV_UINT16 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

-

*y*

-

# EBW16PixelAccessor::SetPixel

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
```

```
void SetPixel(
 OEV_UINT16 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

-

*x*

-

*y*

-

# 1.3. EBW32PixelAccessor Class

-

**Namespace:** Euresys::Open_eVision_1_2

## Methods

| | |
|---|---|
| EBW32PixelAccessor | - |
| GetPixel | - |
| SetPixel | E- |

B

W32PixelAccessor::EBW32PixelAccessor

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
void EBW32PixelAccessor(
 EROIBW32& roi
 )
```

### Parameters

*roi*

-

# EBW32PixelAccessor::GetPixel

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
OEV_UINT32 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

-

*y*

-

# EBW32PixelAccessor::SetPixel

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
void SetPixel(
 OEV_UINT32 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

-

*x*

-

*y*

-

# 1.4. EC15PixelAccessor Class

-

**Namespace:** Euresys::Open_eVision_1_2

## Methods

| | |
|---|---|
| EC15PixelAccessor | - |
| GetPixel | - |
| SetPixel | - |

# EC15PixelAccessor::EC15PixelAccessor

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
void EC15PixelAccessor(
 EROIC15& roi
 )
```

## Parameters

*roi*

-

# EC15PixelAccessor::GetPixel

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
EC15 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

## Parameters

*x*

-

*y*

-

# EC15PixelAccessor::SetPixel

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
void SetPixel(
 EC15 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

-

*x*

-

*y*

-

# 1.5. EC16PixelAccessor Class

-

**Namespace:** Euresys::Open_eVision_1_2

## Methods

| | |
|---|---|
| EC16PixelAccessor | - |
| GetPixel | - |

SetPixel

E-

C

# 16PixelAccessor::EC16PixelAccessor

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
void EC16PixelAccessor(
 EROIC16& roi
 )
```

## Parameters

*roi*

-

# EC16PixelAccessor::GetPixel

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
EC16 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

## Parameters

*x*

-

*y*

-

# EC16PixelAccessor::SetPixel

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
void SetPixel(
 EC16 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

-

*x*

-

*y*

-

# 1.6. EC24APixelAccessor Class

-

**Namespace:** Euresys::Open_eVision_1_2

## Methods

| | |
|---|---|
| EC24APixelAccessor | - |

| | |
|---|---|
| GetPixel | - |
| SetPixel | E- |

C

# 24APixelAccessor::EC24APixelAccessor

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
void EC24APixelAccessor(
 EROIC24A& roi
 )
```

### Parameters

*roi*

-

# EC24APixelAccessor::GetPixel

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
EC24A GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

-

*y*

-

## EC24APixelAccessor::SetPixel

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
void SetPixel(
 EC24A value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

-

*x*

-

*y*

-

# 1.7. EC24PixelAccessor Class

-

**Namespace:** Euresys::Open_eVision_1_2

## Methods

EC24PixelAccessor

| | |
|---|---|
| | - |
| GetPixel | - |
| SetPixel | E- |

C

# 24PixelAccessor::EC24PixelAccessor

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
void EC24PixelAccessor(
 EROIC24& roi
 )
```

### Parameters

*roi*

-

# EC24PixelAccessor::GetPixel

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
```

```
EC24 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

-

*y*

-

# EC24PixelAccessor::SetPixel

-

**Namespace:** Euresys::Open_eVision_1_2

```
[C++]
void SetPixel(
 EC24 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

-

*x*

-

*y*

-

# 2. Common

## 2.1. Easy Class

### Classes



## 2.2. Image and ROI Classes

### Image Classes



### ROI Classes

# 2.3. Region Classes

## Classes

# 3. Libraries

## 3.1. Easy3D Library

**Classes**

"EScaleCalibrationModel Class" on page 2336
"ESimpleCropper Class" on page 2391
"ESphericalCropper Class" on page 2395
"EStatistics Class" on page 2398
E3DViewer
"EZMapToPointCloudConverter Class" on page 2749

### Structs

E3DPoint
EDepth8
EDepth16
EDepth32f

### Enumerations

EMaximumAnalysisMode
EAlignmentPolarity
EPlaneCropperType
EObjectBasedCalibrationType
EObjectBasedCalibrationPrecisionVsSpeedTradeOff
EZMapReferencePlaneMode
EZMapOrientationVectorMode
ENoiseRemovalMethod

# 3.2. Easy3DLaserLine Library

### Classes

"EExplicitGeometricCalibrationModel Class" on page 1194
"EObjectBasedCalibrationGenerator Class" on page 1582
"EObjectBasedCalibrationModel Class" on page 1597
"ELaserLineExtractor Class" on page 1359

# 3.3. Easy3DObject Library

## Classes



# 3.4. EasyImage Library

## Classes

EasyImage
EKernel
EMovingAverage

## Enumerations

EArithmeticLogicOperation
EContourMode
EContourThreshold
EHistogramFeature
EKernelRectifier
EKernelRotation
EKernelType
EReferenceNoise
EThresholdMode

# 3.5. EasyColor Library

## Classes

EasyColor
EColorLookup
EPseudoColorLookup

## Enumerations

EColorQuantization
EColorSystem
ERgbStandard

# 3.6. EasyObject Library

## Classes

EasyObject
ECodedImage2
ECodedElement
EObject
EHole
EObjectSelection
EImageEncoder
EImageSegmenter
ETwoLayersImageSegmenter
EThreeLayersImageSegmenter
EBinaryImageSegmenter
EGrayscaleSingleThresholdSegmenter
EGrayscaleDoubleThresholdSegmenter
EColorSingleThresholdSegmenter
EColorRangeThresholdSegmenter
EImageRangeSegmenter
EReferenceImageSegmenter
ELabeledImageSegmenter
EObjectRunsIterator

## Enumerations

EEncodingConnexity
ESegmentationMethod
ESingleThresholdMode
EDoubleThresholdMode
EFeature

# 3.7. EasyMatch Library

## Classes

EMatcher

## Structs

EMatchPosition

## Enumerations

ECorrelationMode
EMatchContrastMode
EFilteringMode

# 3.8. EasyFind Library

## Classes

EFoundPattern
EPatternFinder

## Enumerations

EFindContrastMode
ELocalSearchMode
EPatternType
EReductionMode
EThinStructureMode

# 3.9. EasyGauge Library

## Classes

ECircleGauge
ELineGauge
EPointGauge
ERectangleGauge
EWedgeGauge
EFrameShape

## Enumerations

EClippingMode
EPlotItem
ETransitionChoice
ETransitionType

# 3.10. EasyOCR Library

## Classes

EOCR

## Enumerations

EMatchingMode
EShiftingMode
EOCRClass
EOCRColor
ESegmentationMode

# 3.11. EasyOCR2 Library

## Classes

EOCR2
EOCR2Text
EOCR2Line
EOCR2Word
EOCR2Char

## Structs

EOCR2CharacterCandidate

## Enumerations

EasyOCR2CharacterFilter
EasyOCR2CharSpacingBias
EasyOCR2CharWidthBias
EasyOCR2DrawDetectionStyle
EasyOCR2DrawRecognitionStyle
EasyOCR2DrawSegmentationStyle
EasyOCR2TextPolarity

# 3.12. EasyOCV Library

## Classes

EOCV
EOCVChar
EOCVText
EChecker

## Enumerations

ECharCreationMode
EDegreesOfFreedom
EDiagnostic
ELearningMode
ELocationMode
ENormalizationMode
EQualityIndicator

# 3.13. EasyBarCode Library

## Classes

EBarCode
EMailBarcode

## Enumerations

EMailBarcodeSymbologies
EMailBarcodeOrientation

# 3.14. EasyMatrixCode Library

## Classes

EMatrixCode
EMatrixCodeReader
ESearchParamsType

## Enumerations

EFamily
EFlipping
ELearnParam
ELogicalSize
EMatrixCodeContrastMode

# 3.15. EasyMatrixCode2 Library

## Classes

EMatrixCode2
EMatrixCode2Reader

## Enumerations

"EReadMode Enum" on page 2960

# 3.16. EasyQRCode Library

## Classes

EQRCode
EQRCodeDecodedStream
EQRCodeDecodedStreamPart
EQRCodeGeometry
EQRCodeReader
EQuadrilateral

## Enumerations

EQRCodeCodingMode
EQRCodeEncoding
EQRCodeLevel
EQRCodeModel
EQRCodeScanPrecision

# 3.17. EasyDeepLearning Library

## Classes

"EClassificationDataset Class" on page 769
"EClassificationMetrics Class" on page 809
"EClassificationResult Class" on page 816
"EClassifier Class" on page 821

# 3.18. Legacy

## EasyObject Library (Legacy)

### Classes

ECodedImage

### Enumerations

EConnexity
ELegacyFeature
ESelectByPosition
ESelectOption
ESortOption

### Functions

EasyObject::ContourArea
EasyObject::ContourGravityCenter
EasyObject::ContourInertia

# 4. Classes

## 4.1. E3DAxisSystem Class

Represent a 3D base axis system.

**Derived Class(es):** E3DOrthonormalAxisSystem E3DRightOrthonormalAxisSystem
**Namespace:** Euresys::Open_eVision_2_10::Easy3D

### Methods

| | |
|---|---|
| CheckIsNormal | check if axis system is Normed |
| CheckIsOrthogonal | check if axis system is Orthogonal |
| CheckIsRightHanded | check if axis system is Right |
| E3DAxisSystem | Constructs an E3DAxisSystem. |
| GetAxisX | Gets the X axis. |
| GetAxisY | Gets the Y axis. |
| GetAxisZ | Gets the Z axis. |

| | |
|---|---|
| GetNormX | Gets the norm of the X axis. |
| GetNormY | Gets the norm of the Y axis. |
| GetNormZ | Gets the norm of the Z axis. |
| GetOrigin | Gets the origin. |
| IsNormal | return true if this base axis system is Normed |
| IsOrthogonal | return true if this base axis system is Orthogonal |
| IsRightHanded | return true if this base axis system is Right Handed |
| operator!= | - |
| operator= | Assignment operator. |
| operator== | operator comparison |
| Serialize | Serializes the object with all attributes. |

# E3DAxisSystem::GetAxisX

Gets the X axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DPoint GetAxisX() const
```

# E3DAxisSystem::GetAxisY

Gets the Y axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DPoint GetAxisY() const
```

# E3DAxisSystem::GetAxisZ

Gets the Z axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DPoint GetAxisZ() const
```

# E3DAxisSystem::CheckIsNormal

check if axis system is Normed

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool CheckIsNormal(
 const E3DPoint& axisX,
 const E3DPoint& axisY,
 const E3DPoint& axisZ
 )
```

### Parameters

*axisX*

-

*axisY*

-

*axisZ*

-

# E3DAxisSystem::CheckIsOrthogonal

check if axis system is Orthogonal

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
bool CheckIsOrthogonal(
 const E3DPoint& axisX,
 const E3DPoint& axisY,
 const E3DPoint& axisZ
 )
```

### Parameters

*axisX*

-

*axisY*

-

*axisZ*

-

# E3DAxisSystem::CheckIsRightHanded

check if axis system is Right

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
bool CheckIsRightHanded(
 const E3DPoint& axisX,
 const E3DPoint& axisY,
 const E3DPoint& axisZ
 )
```

### Parameters

*axisX*

-

*axisY*

-

*axisZ*

-

# E3DAxisSystem::E3DAxisSystem

Constructs an E3DAxisSystem.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void E3DAxisSystem(
 )

void E3DAxisSystem(
 const E3DPoint& Origin,
 const E3DPoint& axisX,
 const E3DPoint& axisY,
 const E3DPoint& axisZ
 )

void E3DAxisSystem(
 const E3DAxisSystem& other
 )
```

## Parameters

*Origin*

The origin of the axis system

*axisX*

The X axis

*axisY*

The Y axis

*axisZ*

The Z axis

*other*

Reference to another E3DAxisSystem used for the initialization.

# E3DAxisSystem::IsNormal

return true if this base axis system is Normed

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool IsNormal(
 )
```

# E3DAxisSystem::IsOrthogonal

return true if this base axis system is Orthogonal

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool IsOrthogonal(
 )
```

# E3DAxisSystem::IsRightHanded

return true if this base axis system is Right Handed

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
bool IsRightHanded(
 )
```

# E3DAxisSystem::GetNormX

Gets the norm of the X axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetNormX() const
```

# E3DAxisSystem::GetNormY

Gets the norm of the Y axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetNormY() const
```

# E3DAxisSystem::GetNormZ

Gets the norm of the Z axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetNormZ() const
```

# E3DAxisSystem::operator!=

-

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
bool operator!=(
 const E3DAxisSystem& other
 )
```

## Parameters

*other*

-

# E3DAxisSystem::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DAxisSystem& operator=(
 const E3DAxisSystem& other
 )
```

### Parameters

*other*

The source E3DAxisSystem.

# E3DAxisSystem::operator==

operator comparison

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool operator==(
 const E3DAxisSystem& other
 )
```

### Parameters

*other*

-

# E3DAxisSystem::GetOrigin

Gets the origin.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPoint GetOrigin() const
```

## E3DAxisSystem::Serialize

Serializes the object with all attributes.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or write to.

# 4.2. E3DBox Class

A 3D box, used as bounding volume for E3DObject class. A box is defined by a center, 3 axis and 3 extent for the 3 axis. A 3D point (x,y,z) is inside the E3DBox if ... By default a E3DBox is an axis aligned unit cube, centered on the origin.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| E3DBox | Constructs an default E3DBox. By default a E3DBox is an axis aligned unit cube, centered on the origin. |
| GetAxes | 3D orthonormal axis system of the box. |
| GetCenter | 3D box center. |
| GetXAxis | X axis of the box. |
| GetXSize | Size of the 3D box along its X axis . |
| GetXYQuadrangle | 2D quadrangle of the E3DBox in the XY plane |
| GetYAxis | Y axis of the box. |
| GetYSize | Size of the 3D box along its Y axis. |
| GetZAxis | Z axis of the box. |
| GetZSize | Size of the 3D box along its Z axis . |
| Load | Loads the E3DBox. The given ESerializer must have been created for reading. |

| | |
|---|---|
| operator!= | Checks if two E3DBox are different |
| operator= | Assignment operator for the E3DBox. |
| operator== | Checks if two E3DBox are stricly equal |
| Save | Saves the E3DBox. The given ESerializer must have been created for writing. |
| SetAxes | 3D orthonormal axis system of the box. |
| SetXSize | Size of the 3D box along its X axis . |
| SetYSize | Size of the 3D box along its Y axis. |
| SetZSize | Size of the 3D box along its Z axis . |

# E3DBox::GetAxes

# E3DBox::SetAxes

3D orthonormal axis system of the box.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DOrthonormalAxisSystem GetAxes() const
void SetAxes(const E3DOrthonormalAxisSystem& axes)
```

# E3DBox::GetCenter

3D box center.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DPoint GetCenter() const
```

# E3DBox::E3DBox

Constructs an default E3DBox. By default a E3DBox is an axis aligned unit cube, centered on the origin.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void E3DBox(
 )
void E3DBox(
 float xSize,
 float ySize,
 float zSize
 )
```

```
void E3DBox(
 const E3DOrthonormalAxisSystem& axes,
 float xSize,
 float ySize,
 float zSize
 )

void E3DBox(
 const E3DBox& other
 )

void E3DBox(
 const E3DPoint& center,
 float roll,
 float pitch,
 float yaw,
 float xSize,
 float ySize,
 float zSize
 )
```

## Parameters

*xSize*

The full size of the box along the X axis.

*ySize*

The full size of the box along the Y axis.

*zSize*

The full size of the box along the Z axis.

*axes*

Axis system.

*other*

Reference to another E3DBox used for the initialization.

*center*

The 3D coordinate of the box center.

*roll*

Roll (rotation along the X axis) of the box.

*pitch*

Pitch (rotation along the Y axis) of the box.

*yaw*

Yaw (rotation along the Z axis) of the box.

# E3DBox::Load

Loads the E3DBox. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

-

# E3DBox::operator!=

Checks if two E3DBox are different

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool operator!=(
 const E3DBox& other
 )
```

### Parameters

*other*

-

# E3DBox::operator=

Assignment operator for the E3DBox.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DBox& operator=(
 const E3DBox& other
 )
```

### Parameters

*other*

-

# E3DBox::operator==

Checks if two E3DBox are stricly equal

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool operator==(
 const E3DBox& other
 )
```

### Parameters

*other*

-

# E3DBox::Save

Saves the E3DBox. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

-

# E3DBox::GetXAxis

X axis of the box.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPoint GetXAxis() const
```

# E3DBox::GetXSize

# E3DBox::SetXSize

Size of the 3D box along its X axis .

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetXSize() const
void SetXSize(float xSize)
```

# E3DBox::GetXYQuadrangle

2D quadrangle of the E3DBox in the XY plane

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EQuadrangle GetXYQuadrangle() const
```

# E3DBox::GetYAxis

Y axis of the box.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DPoint GetYAxis() const
```

# E3DBox::GetYSize

# E3DBox::SetYSize

Size of the 3D box along its Y axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetYSize() const
void SetYSize(float ySize)
```

# E3DBox::GetZAxis

Z axis of the box.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DPoint GetZAxis() const
```

## E3DBox::GetZSize

## E3DBox::SetZSize

Size of the 3D box along its Z axis .

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetZSize() const
void SetZSize(float zSize)
```

# 4.3. E3DObject Class

A E3DObject is a geometric description of a set of 3D points, produced by E3DObjectExtractor. Several 3D features are available. All 3D features are expressed in the ZMap metric coordinate system.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

### Methods

| | |
|---|---|
| Draw | Draws the specified feature of the object in the given graphic context |
| E3DObject | Constructs an E3DObject with default (invalid) values |
| GetArea | Object area in metric units. |

| | |
|---|---|
| GetAveragePosition | 3D average position of the object. |
| GetBasePlane | Base plane. The base plane is calculated using points surrounding the object. |
| GetBaseTilt | Angle between the base plane and the vertical (Z) axis. |
| GetBoundingBox | The E3DBox (3D oriented bounding box) of the object. The bounding box is oriented in the XY plane of the ZMap space (rotation over the ZMap Z axis). The bounding box X and Y sizes are the object length and width (see E3DObject and E3DObject). The bounding box Z size is always in the ZMap Z axis direction. |
| GetLength | Length of the object in metric units. The length is the largest dimension of the object on the XY plane of the ZMap space. |
| GetLocalHeight | Local height of the object in metric units. The local height of the object is relative to the surroundings. The base plane is used as the reference for the calculation of the local height. If it is not possible to evaluate a base plane, the local height has the same value as the reference height (E3DObject) |
| GetLocalTilt | Angle between the object plane and the base plane. |
| GetLocalTopPosition | 3D highest position of the object relatively to the object base plane. If it is not possible to evaluate a base plane, the local top position is the reference top position (E3DObject) |

| | |
|---|---|
| GetNumPixels | Number of ZMap pixels composing the object. |
| GetOrientation | Orientation of the object. The orientation is the angle between the object major (longest) axis and the ZMap X axis. |
| GetPlane | Plane fitted to the object 3D positions. |
| GetRectangleRegion | ERectangleRegion enclosing the object ZMap pixels. |
| GetReferenceHeight | Reference height of the object in metric units. The reference height of the object is relative to the ZMap origin (also known as the reference plane). |
| GetReferenceTilt | Angle between the object plane and the vertical (Z) axis. |
| GetReferenceTopPosition | 3D top position relative to the ZMap origin (this is the position with the highest Z coordinate) |
| GetRegion | ERegion composed of the object ZMap pixels. |
| GetVolume | Object volume in metric units. |
| GetWidth | Width of the object in metric units. The width is the smallest dimension of the object on the XY plane of the ZMap space. |

| | |
|---|---|
| Load | Loads the E3DObject. The given ESerializer must have been created for reading. |
| operator= | Assignment operator |
| operator== | Comparison operator |
| Save | Saves the E3DObject. The given ESerializer must have been created for writing. |

# E3DObject::GetArea

Object area in metric units.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetArea() const
```

# E3DObject::GetAveragePosition

3D average position of the object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPoint GetAveragePosition() const
```

# E3DObject::GetBasePlane

Base plane. The base plane is calculated using points surrounding the object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPlane GetBasePlane() const
```

# E3DObject::GetBaseTilt

Angle between the base plane and the vertical (Z) axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetBaseTilt() const
```

# E3DObject::GetBoundingBox

The E3DBox (3D oriented bounding box) of the object. The bounding box is oriented in the XY plane of the ZMap space (rotation over the ZMap Z axis). The bounding box X and Y sizes are the object length and width (see E3DObject and E3DObject). The bounding box Z size is always in the ZMap Z axis direction.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DBox GetBoundingBox() const
```

# E3DObject::Draw

Draws the specified feature of the object in the given graphic context

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Draw(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::Easy3D::E3DObjectFeature feature,
 const ERGBColor& color,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*graphicContext*

Graphic context on which to draw.

*feature*

The feature to draw.

*color*

The color in which to draw the feature (optional).

*zoomX*

-

*zoomY*

-

*panX*

-

*panY*

-

### Remarks

Drawing is done in the device context associated to the desired window.

# E3DObject::E3DObject

Constructs an E3DObject with default (invalid) values

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void E3DObject(
 )
void E3DObject(
 const E3DObject& other
 )
```

### Parameters

*other*

-

# E3DObject::GetLength

Length of the object in metric units. The length is the largest dimension of the object on the XY plane of the ZMap space.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetLength() const
```

# E3DObject::Load

Loads the E3DObject. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

-

# E3DObject::GetLocalHeight

Local height of the object in metric units. The local height of the object is relative to the surroundings. The base plane is used as the reference for the calculation of the local height. If it is not possible to evaluate a base plane, the local height has the same value as the reference height (E3DObject)

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetLocalHeight() const
```

# E3DObject::GetLocalTilt

Angle between the object plane and the base plane.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetLocalTilt() const
```

# E3DObject::GetLocalTopPosition

3D highest position of the object relatively to the object base plane. If it is not possible to evaluate a base plane, the local top position is the reference top position (E3DObject)

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DPoint GetLocalTopPosition() const
```

# E3DObject::GetNumPixels

Number of ZMap pixels composing the object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
int GetNumPixels() const
```

# E3DObject::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DObject& operator=(
 const E3DObject& other
 )
```

## Parameters

*other*

-

# E3DObject::operator==

Comparison operator

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
bool operator==(
 const E3DObject& other
 )
```

### Parameters

*other*

The other E3DObject.

# E3DObject::GetOrientation

Orientation of the object. The orientation is the angle between the object major (longest) axis and the ZMap X axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetOrientation() const
```

# E3DObject::GetPlane

Plane fitted to the object 3D positions.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPlane GetPlane() const
```

# E3DObject::GetRectangleRegion

ERectangleRegion enclosing the object ZMap pixels.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const ERectangleRegion& GetRectangleRegion() const
```

# E3DObject::GetReferenceHeight

Reference height of the object in metric units. The reference height of the object is relative to the ZMap origin (also known as the reference plane).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetReferenceHeight() const
```

# E3DObject::GetReferenceTilt

Angle between the object plane and the vertical (Z) axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
float GetReferenceTilt() const
```

# E3DObject::GetReferenceTopPosition

3D top position relative to the ZMap origin (this is the position with the highest Z coordinate)

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPoint GetReferenceTopPosition() const
```

# E3DObject::GetRegion

ERegion composed of the object ZMap pixels.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const ERegion& GetRegion() const
```

# E3DObject::Save

Saves the E3DObject. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

-

# E3DObject::GetVolume

Object volume in metric units.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetVolume() const
```

# E3DObject::GetWidth

Width of the object in metric units. The width is the smallest dimension of the object on the XY plane of the ZMap space.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetWidth() const
```

# 4.4. E3DObjectExtractor Class

E3DObjectExtractor is used to extract 3D objects from a ZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| AddToMesh | For all extracted objects, adds a 3D representation (in the form of a list of triangles) of the given feature to the given mesh. If the feature is not defined for the E3DObject, this method has no effect. |
| Draw | Draws the specified feature of all extracted objects in the given graphic context. If the feature is not defined for the E3DObject, this method has no effect. |
| E3DObjectExtractor | Constructs an E3DObjectExtractor with default (invalid) values |
| Extract | Processes the ZMap and extracts a list of 3D objects matching the criteria. Returns the number of extracted objects. |
| GetAreaRange | The allowed area range for the objects. Area is expressed in the metric coordinate system. E.g, if the ZMap space is in mm, the area is in mm2. |

| | |
|---|---|
| GetAspectRatioRange | The extraction 2D aspect ratio range. The aspect ratio is the smallest dimension divided by the largest dimension, its value is always between 0 and 1. The smaller the ratio, the more elongated the object is. A value of 1 means a perfectly square object (length=width). |
| GetLengthRange | The extraction object length range in metric units. Length is the largest dimension of the object, expressed the ZMap coordinate system. |
| GetLocalHeightRange | The extraction object local height range in metric units. Local height is the dimension of the object along the normal of the base plane. For a height based on the ZMap origin, use E3DObjectExtractor. |
| GetLocalTiltRange | The allowed angle range of the object local tilt. This is the angle between the base plane and the object plane. A value of 0 means that the object top surface is parallel to its base. Valid values are between 0 and +90 degrees (or 0 and Pi if angle unit is radians). |
| GetObjects | Returns the list of extracted objects. |
| GetOrientationRange | The allowed angle range of the oriented 2D rectangle region. This is the angle of the longest axis (the length of the object), in counter clockwise direction, from the horizontal axis. Valid values are between -90 and +90 degrees (or -Pi/2 and Pi/2 if angle unit is radians). |

| | |
|---|---|
| GetReferenceHeightRange | The extraction object reference height range in metric units. Reference height is the dimension of the object along the ZMap Z axis from ZMap origin. For a height based on the object base plane, use E3DObjectExtractor. |
| GetReferenceTiltRange | The allowed angle range of the object reference tilt. This is the angle between the object plane and the ZMap Z Axis. A value of 0 means that the object top surface is parallel to the ZMap XY plane. Valid values are between 0 and +90 degrees (or 0 and Pi if angle unit is radians). |
| GetVolumeRange | The allowed volume range for the objects. Volume is expressed in the metric coordinate system. E.g, if the ZMap space is in mm, the volume is in mm3. |
| GetWidthRange | The extraction object width range in metric units. Width is the smallest dimension of the object, expressed the ZMap coordinate system. |
| Load | Load the E3DObjectExtractor. The given ESerializer must have been created for reading. |
| operator= | Assignment operator |

| | |
|---|---|
| operator== | Comparison operator |
| Save | Save the E3DObjectExtractor. The given ESerializer must have been created for writing. |
| SetAreaRange | The allowed area range for the objects. Area is expressed in the metric coordinate system. E.g, if the ZMap space is in mm, the area is in mm2. |
| SetAspectRatioRange | The extraction 2D aspect ratio range. The aspect ratio is the smallest dimension divided by the largest dimension, its value is always between 0 and 1. The smaller the ratio, the more elongated the object is. A value of 1 means a perfectly square object (length=width). |
| SetLengthRange | The extraction object length range in metric units. Length is the largest dimension of the object, expressed the ZMap coordinate system. |
| SetLocalHeightRange | The extraction object local height range in metric units. Local height is the dimension of the object along the normal of the base plane. For a height based on the ZMap origin, use E3DObjectExtractor. |
| SetLocalTiltRange | The allowed angle range of the object local tilt. This is the angle between the base plane and the object plane. A value of 0 means that the object top surface is parallel to its base. Valid values are between 0 and +90 degrees (or 0 and Pi if angle unit is radians). |

SetOrientationRange

The allowed angle range of the oriented 2D rectangle region. This is the angle of the longest axis (the length of the object), in counter clockwise direction, from the horizontal axis. Valid values are between -90 and +90 degrees (or -Pi/2 and Pi/2 if angle unit is radians).

SetReferenceHeightRange

The extraction object reference height range in metric units. Reference height is the dimension of the object along the ZMap Z axis from ZMap origin. For a height based on the object base plane, use E3DObjectExtractor.

SetReferenceTiltRange

The allowed angle range of the object reference tilt. This is the angle between the object plane and the ZMap Z Axis. A value of 0 means that the object top surface is parallel to the ZMap XY plane. Valid values are between 0 and +90 degrees (or 0 and Pi if angle unit is radians).

SetVolumeRange

The allowed volume range for the objects. Volume is expressed in the metric coordinate system. E.g, if the ZMap space is in mm, the volume is in mm3.

SetWidthRange

The extraction object width range in metric units. Width is the smallest dimension of the object, expressed the ZMap coordinate system.

# E3DObjectExtractor::AddToMesh

For all extracted objects, adds a 3D representation (in the form of a list of triangles) of the given feature to the given mesh. If the feature is not defined for the E3DObject, this method has no effect.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void AddToMesh(
 Euresys::Open_eVision_2_10::Easy3D::E3DObjectFeature feature,
 EMesh& mesh
 )
```

### Parameters

*feature*

The feature to draw, only 3D features are supported.

*mesh*

The mesh to add the graphics for.

# E3DObjectExtractor::GetAreaRange

# E3DObjectExtractor::SetAreaRange

The allowed area range for the objects. Area is expressed in the metric coordinate system. E.g, if the ZMap space is in mm, the area is in mm2.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
const EFloatRange& GetAreaRange() const
void SetAreaRange(const EFloatRange& areaRange)
```

# E3DObjectExtractor::GetAspectRatioRange

# E3DObjectExtractor::SetAspectRatioRange

The extraction 2D aspect ratio range. The aspect ratio is the smallest dimension divided by the largest dimension, its value is always between 0 and 1. The smaller the ratio, the more elongated the object is. A value of 1 means a perfectly square object (length=width).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
const EFloatRange& GetAspectRatioRange() const
void SetAspectRatioRange(const EFloatRange& arRange)
```

# E3DObjectExtractor::Draw

Draws the specified feature of all extracted objects in the given graphic context. If the feature is not defined for the E3DObject, this method has no effect.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Draw(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::Easy3D::E3DObjectFeature feature,
 const ERGBColor& color,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

### Parameters

*graphicContext*

Graphic context on which to draw.

*feature*

The feature to draw.

*color*

The color in which to draw the feature (optional).

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

### Remarks

<Drawing is done in the device context associated to the desired window.

# E3DObjectExtractor::E3DObjectExtractor

Constructs an E3DObjectExtractor with default (invalid) values

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void E3DObjectExtractor(
 )
void E3DObjectExtractor(
 const E3DObjectExtractor& other
 )
```

### Parameters

*other*

-

# E3DObjectExtractor::Extract

Processes the ZMap and extracts a list of 3D objects matching the criteria. Returns the number of extracted objects.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
int Extract(
 const EZMap8& zMap
 )
int Extract(
 const EZMap8& zMap,
 ERegion& region
 )
int Extract(
 const EZMap16& zMap
 )
int Extract(
 const EZMap16& zMap,
 ERegion& region
 )
```

```
int Extract(
 const EZMap32f& zMap
 )
int Extract(
 const EZMap32f& zMap,
 ERegion& region
 )
int Extract(
 const EZMap* zMap
 )
int Extract(
 const EZMap* zMap,
 ERegion& region
 )
```

### Parameters

*zMap*

The source ZMap.

*region*

The region of interest, only pixels inside the given region are considered for object extraction.

# E3DObjectExtractor::GetLengthRange

# E3DObjectExtractor::SetLengthRange

The extraction object length range in metric units. Length is the largest dimension of the object, expressed the ZMap coordinate system.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
const EFloatRange& GetLengthRange() const
```

```
void SetLengthRange(const EFloatRange& lengthRange)
```

# E3DObjectExtractor::Load

Load the E3DObjectExtractor. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

-

# E3DObjectExtractor::GetLocalHeightRange

# E3DObjectExtractor::SetLocalHeightRange

The extraction object local height range in metric units. Local height is the dimension of the object along the normal of the base plane. For a height based on the ZMap origin, use E3DObjectExtractor.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
const EFloatRange& GetLocalHeightRange() const
void SetLocalHeightRange(const EFloatRange& localHeightRange)
```

# E3DObjectExtractor::GetLocalTiltRange

# E3DObjectExtractor::SetLocalTiltRange

The allowed angle range of the object local tilt. This is the angle between the base plane and the object plane. A value of 0 means that the object top surface is parallel to its base. Valid values are between 0 and +90 degrees (or 0 and Pi if angle unit is radians).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
const EFloatRange& GetLocalTiltRange() const
void SetLocalTiltRange(const EFloatRange& tiltRange)
```

# E3DObjectExtractor::GetObjects

Returns the list of extracted objects.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
std::vector<Euresys::Open_eVision_2_10::Easy3D::E3DObject> GetObjects() const
```

# E3DObjectExtractor::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DObjectExtractor& operator=(
 const E3DObjectExtractor& other
 )
```

### Parameters

*other*

-

# E3DObjectExtractor::operator==

Comparison operator

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool operator==(
 const E3DObjectExtractor& other
 )
```

### Parameters

*other*

The other object.

# E3DObjectExtractor::GetOrientationRange

# E3DObjectExtractor::SetOrientationRange

The allowed angle range of the oriented 2D rectangle region. This is the angle of the longest axis (the length of the object), in counter clockwise direction, from the horizontal axis. Valid values are between -90 and +90 degrees (or -Pi/2 and Pi/2 if angle unit is radians).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
const EFloatRange& GetOrientationRange() const
void SetOrientationRange(const EFloatRange& orientationRange)
```

# E3DObjectExtractor::GetReferenceHeightRange

# E3DObjectExtractor::SetReferenceHeightRange

The extraction object reference height range in metric units. Reference height is the dimension of the object along the ZMap Z axis from ZMap origin. For a height based on the object base plane, use E3DObjectExtractor.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
const EFloatRange& GetReferenceHeightRange() const
void SetReferenceHeightRange(const EFloatRange& referenceHeightRange)
```

# E3DObjectExtractor::GetReferenceTiltRange

# E3DObjectExtractor::SetReferenceTiltRange

The allowed angle range of the object reference tilt. This is the angle between the object plane and the ZMap Z Axis. A value of 0 means that the object top surface is parallel to the ZMap XY plane. Valid values are between 0 and +90 degrees (or 0 and Pi if angle unit is radians).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const EFloatRange& GetReferenceTiltRange() const
void SetReferenceTiltRange(const EFloatRange& tiltRange)
```

# E3DObjectExtractor::Save

Save the E3DObjectExtractor. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

-

# E3DObjectExtractor::GetVolumeRange

# E3DObjectExtractor::SetVolumeRange

The allowed volume range for the objects. Volume is expressed in the metric coordinate system. E.g, if the ZMap space is in mm, the volume is in mm3.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const EFloatRange& GetVolumeRange() const
void SetVolumeRange(const EFloatRange& volumeRange)
```

# E3DObjectExtractor::GetWidthRange

# E3DObjectExtractor::SetWidthRange

The extraction object width range in metric units. Width is the smallest dimension of the object, expressed the ZMap coordinate system.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const EFloatRange& GetWidthRange() const
void SetWidthRange(const EFloatRange& widthRange)
```

# 4.5. E3DOrthonormalAxisSystem Class

E3DOrthonormalAxisSystem is a subassembly of E3DAxisSystem with properties Orthogonal and Normed

**Base Class:** E3DAxisSystem

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| E3DOrthonormalAxisSystem | Constructs an E3DOrthonormalAxisSystem. |
| operator= | Assignment operator. |

3

# DOrthonormalAxisSystem::E3DOrthonormalAxisSystem

Constructs an E3DOrthonormalAxisSystem.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void E3DOrthonormalAxisSystem(
 )

void E3DOrthonormalAxisSystem(
 const E3DAxisSystem& other
 )
```

```
void E3DOrthonormalAxisSystem(
 const E3DOrthonormalAxisSystem& other
 )
void E3DOrthonormalAxisSystem(
 const E3DPoint& Origin,
 const E3DPoint& axisX,
 const E3DPoint& axisY,
 const E3DPoint& axisZ
 )
```

### Parameters

*other*

Reference to another E3DOrthonormalAxisSystem used for the initialization.

*Origin*

The origin of the axis system

*axisX*

The X axis

*axisY*

The Y axis

*axisZ*

The Z axis

### Remarks

throws an exception if the axis are not orthogonal and normed

# E3DOrthonormalAxisSystem::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
E3DOrthonormalAxisSystem& operator=(
 const E3DOrthonormalAxisSystem& other
 )
```

### Parameters

*other*

The source E3DOrthonormalAxisSystem.

# 4.6. E3DPlane Class

Represents a 3D plane.
The equation of the plane is "n_vect . (x,y,z) = signedDistance" where "n_vect" is the normal vector and "signedDistance" is the signed distance from the origin to the plane.
The signed distance is positive when the vector binding the origin to the closest point on the plane has the same direction as "n_vect" and is negative when this vector has the opposite direction as "n_vect".

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| Define | (re)Defines the plane parameters:<br>It is possible to use the normal and the signed distance of the plane.<br>In that case, it exits with an exception if the normal vector is the null vector.<br>It is also possible to use 3 points of the plane.<br>In that case, it exits with an exception if the 3 points are aligned. |
| DistanceTo | Returns the signed distance between the plane and a given point.<br>A positive distance means that the vector connecting the plane to the point has the same direction as the normal while a negative distance means that it has the opposite direction. |

| | |
|---|---|
| E3DPlane | Creates an E3DPlane object.<br>It is possible to initialize it by specifying its normal and signed distance from the origin.<br>In that case, it exits with an exception if the norm of the normal is null.<br>It is also possible to initialize it by specifying 3 points of the plane.<br>In that case, it exits with an exception if the 3 points are aligned. |
| GetNormal | Gets/Sets the normal vector of the plane. |
| GetSignedDistanceFromOrigin | Gets/Sets the signed distance between the origin and the plane. |
| Load | Loads a E3DPlane. The given ESerializer must have been created for reading. |
| operator- | Operator "-": returns a new E3DPlane translated in the inverse of the direction of the normal to the plane. |
| operator+ | Operator "+": returns a new E3DPlane translated in the direction of the normal to the plane. |
| operator= | Assignment operator |
| ProjectPoint | Returns the position of the given point projected on the plane. |

| | |
|---|---|
| Save | Saves a E3DPlane. The given ESerializer must have been created for writing. |
| SetNormal | Gets/Sets the normal vector of the plane. |
| SetSignedDistanceFromOrigin | Gets/Sets the signed distance between the origin and the plane. |



# DPlane::Define

(re)Defines the plane parameters:
It is possible to use the normal and the signed distance of the plane.
In that case, it exits with an exception if the normal vector is the null vector.
It is also possible to use 3 points of the plane.
In that case, it exits with an exception if the 3 points are aligned.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Define(
 const E3DPoint& normal,
 float signedDistance
 )

void Define(
 const E3DPoint& point1,
 const E3DPoint& point2,
 const E3DPoint& point3
 )
```

## Parameters

*normal*

The normal vector, represented by an E3DPoint.

*signedDistance*

The signed distance between the origin and the plane.

*point1*

First point.

*point2*

Second point.

*point3*

Third point.

### Remarks

When we define a plane by specifying 3 points, the normal vector always points toward the positive Z.

# E3DPlane::DistanceTo

Returns the signed distance between the plane and a given point.
A positive distance means that the vector connecting the plane to the point has the same direction as the normal while a negative distance means that it has the opposite direction.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float DistanceTo(
 const E3DPoint& point
 )
```

### Parameters

*point*

The 3D point to measure the distance to.

# E3DPlane::E3DPlane

Creates an E3DPlane object.
It is possible to initialize it by specifying its normal and signed distance from the origin.
In that case, it exits with an exception if the norm of the normal is null.
It is also possible to initialize it by specifying 3 points of the plane.
In that case, it exits with an exception if the 3 points are aligned.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void E3DPlane(
 )
void E3DPlane(
 const E3DPoint& normal,
 float signedDistance
 )
void E3DPlane(
 const E3DPoint& point1,
 const E3DPoint& point2,
 const E3DPoint& point3
 )
void E3DPlane(
 const E3DPlane& other
 )
```

## Parameters

*normal*

-

*signedDistance*

-

*point1*

-

*point2*

-

*point3*

-

*other*

-

### Remarks

When we define a plane by specifying 3 points, the normal vector always points toward the positive Z.

# E3DPlane::Load

Loads a E3DPlane. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from.

# E3DPlane::GetNormal

# E3DPlane::SetNormal

Gets/Sets the normal vector of the plane.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPoint GetNormal() const
void SetNormal(const E3DPoint& vector)
```

### Remarks

Normal values will be stored normalized.

# E3DPlane::operator-

Operator "-": returns a new E3DPlane translated in the inverse of the direction of the normal to the plane.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPlane operator-(
 float offset
 )
```

### Parameters

*offset*

offset value

# E3DPlane::operator+

Operator "+": returns a new E3DPlane translated in the direction of the normal to the plane.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPlane operator+(
 float offset
 )
```

### Parameters

*offset*

offset value

# E3DPlane::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPlane& operator=(
 const E3DPlane& other
 )
```

### Parameters

*other*

The E3DPlane object that should be copied.

# E3DPlane::ProjectPoint

Returns the position of the given point projected on the plane.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DPoint ProjectPoint(
 const E3DPoint& point
 )
```

### Parameters

*point*

The 3D point to project on plane.

# E3DPlane::Save

Saves a E3DPlane. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is written to.

# E3DPlane::GetSignedDistanceFromOrigin

# E3DPlane::SetSignedDistanceFromOrigin

Gets/Sets the signed distance between the origin and the plane.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetSignedDistanceFromOrigin() const
void SetSignedDistanceFromOrigin(float signedDistance)
```

# 4.7. E3DRightOrthonormalAxisSystem Class

E3DRightOrthonormalAxisSystem is a subassembly of E3DAxisSystem with properties Orthogonal and Normed and Right

**Base Class:** E3DAxisSystem

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| E3DRightOrthonormalAxisSystem | Constructs an E3DRightOrthonormalAxisSystem. |
| operator= | Assignment operator. |

3

DRightOrthonormalAxisSystem::E3DRightOrthonormalAxisSystem

Constructs an E3DRightOrthonormalAxisSystem.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void E3DRightOrthonormalAxisSystem(
 )
void E3DRightOrthonormalAxisSystem(
 const E3DAxisSystem& other
 )
void E3DRightOrthonormalAxisSystem(
 const E3DRightOrthonormalAxisSystem& other
 )
void E3DRightOrthonormalAxisSystem(
 const E3DPoint& Origin,
 const E3DPoint& axisX,
 const E3DPoint& axisY,
 const E3DPoint& axisZ
 )
```

### Parameters

*other*

Reference to another E3DRightOrthonormalAxisSystem used for the initialization.

*Origin*

The origin of the axis system

*axisX*

The X axis

*axisY*

The Y axis

*axisZ*

The Z axis

### Remarks

throws an exception if the axis are not orthogonal and normed and right

# E3DRightOrthonormalAxisSystem::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DRightOrthonormalAxisSystem& operator=(
 const E3DRightOrthonormalAxisSystem& other
 )
```

### Parameters

*other*

The source E3DRightOrthonormalAxisSystem.

# 4.8. E3DTransformMatrix Class

Represents a 3D transformation [4x4] matrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| CreateAnisotropicScalingMatrix | Creates an anisotropic scaling E3DTransformMatrix. |
| CreateIdentityMatrix | Creates an identity (neutral) E3DTransformMatrix. |
| CreateIsotropicScalingMatrix | Creates an isotropic scaling E3DTransformMatrix. |
| CreateOrthoBasis | Creates a orthonormal E3DTransformMatrix basis (corresponds to a rigid transformation).<br>The vector e1, e2, e3 should form a right-handed orthogonal basis. |
| CreateOrthographicProjectionMatrix | Creates an othographic projection E3DTransformMatrix. |

| | |
|---|---|
| CreatePerspectiveProjectionMatrix | Creates a perspective projection E3DTransformMatrix. |
| CreateRotationXMatrix | Creates a rotation E3DTransformMatrix around the X axis matrix. |
| CreateRotationYMatrix | Creates a rotation E3DTransformMatrix around the Y axis matrix. |
| CreateRotationZMatrix | Creates a rotation E3DTransformMatrix around the Z axis matrix. |
| CreateTranslationMatrix | Creates a translation E3DTransformMatrix. |
| E3DTransformMatrix | Creates an E3DTransformMatrix object. |
| GetOrthoBasis | Gets the orthogonal basis represented by this transformation or throws an exception if it is not a rigid transformation. |
| GetValue | Gets a value from the E3DTransformMatrix object. |
| Inverse | Returns the inverted E3DTransformMatrix.<br>An exception will be thrown if the determinant of the matrix is **0**. |
| IsRigid | Checks that the transformation is a rigid transformation (keep the distances and angles). |
| Load | Loads the E3DTransformMatrix object. The given ESerializer must have been created for reading. |

| | |
|---|---|
| operator!= | Checks if two E3DTransformMatrix objects are stricly differents (binary level). |
| operator* | E3DTransformMatrix product. Combines the transformations of the two matrices. |
| operator+ | E3DTransformMatrix sum. Sums the current and the given matrix, returns the result. |
| operator= | Assignment operator. |
| operator== | Checks if two E3DTransformMatrix objects are stricly equals (binary level). |
| Save | Saves the E3DTransformMatrix object. The given ESerializer must have been created for writing. |
| SetValue | Sets a value in the E3DTransformMatrix object. |
| Transpose | Returns the transposed E3DTransformMatrix.<br>If the matrix is orthogonal (rotation only transformation), the transposed matrix is the inverse transformation. |

E3

# DTransformMatrix::CreateAnisotropicScalingMatrix

Creates an anisotropic scaling E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix CreateAnisotropicScalingMatrix(
 float scaleX,
 float scaleY,
 float scaleZ
 )
```

### Parameters

*scaleX*

Scaling factor along the X axis.

*scaleY*

Scaling factor along the Y axis.

*scaleZ*

Scaling factor along the Z axis.

# E3DTransformMatrix::CreateIdentityMatrix

Creates an identity (neutral) E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix CreateIdentityMatrix(
 )
```

# E3DTransformMatrix::CreateIsotropicScalingMatrix

Creates an isotropic scaling E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix CreateIsotropicScalingMatrix(
 float scale
 )
```

### Parameters

*scale*

Scaling factor.

# E3DTransformMatrix::CreateOrthoBasis

Creates a orthonormal E3DTransformMatrix basis (corresponds to a rigid transformation).
The vector e1, e2, e3 should form a right-handed orthogonal basis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix CreateOrthoBasis(
 const E3DPoint& e1,
 const E3DPoint& e2,
 const E3DPoint& e3,
 const E3DPoint& t
 )
```

### Parameters

*e1*

Vector 1.

*e2*

Vector 2.

*e3*

Vector 3.

*t*

Translation.

# E3DTransformMatrix::CreateOrthographicProjectionMatrix

Creates an othographic projection E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix CreateOrthographicProjectionMatrix(
 float width,
 float height
 )
```

## Parameters

*width*

Width of the viewport.

*height*

Height of the viewport.

# E3DTransformMatrix::CreatePerspectiveProjectionMatrix

Creates a perspective projection E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
E3DTransformMatrix CreatePerspectiveProjectionMatrix(
 float distance,
 float width,
 float height
 )
```

### Parameters

*distance*

Distance of the viewport to the origin.

*width*

Width of the viewport.

*height*

Height of the viewport.

# E3DTransformMatrix::CreateRotationXMatrix

Creates a rotation E3DTransformMatrix around the X axis matrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix CreateRotationXMatrix(
 float Angle
 )
```

### Parameters

*Angle*

Rotation angle.

# E3DTransformMatrix::CreateRotationYMatrix

Creates a rotation E3DTransformMatrix around the Y axis matrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix CreateRotationYMatrix(
 float Angle
 )
```

## Parameters

*Angle*

Rotation angle.

# E3DTransformMatrix::CreateRotationZMatrix

Creates a rotation E3DTransformMatrix around the Z axis matrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix CreateRotationZMatrix(
 float Angle
 )
```

## Parameters

*Angle*

Rotation angle.

# E3DTransformMatrix::CreateTranslationMatrix

Creates a translation E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix CreateTranslationMatrix(
 float dX,
 float dY,
 float dZ
 )
```

### Parameters

*dX*

Translation along the X axis.

*dY*

Translation along the Y axis.

*dZ*

Translation along the Z axis.

# E3DTransformMatrix::E3DTransformMatrix

Creates an E3DTransformMatrix object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void E3DTransformMatrix(
 )
```

```
void E3DTransformMatrix(
 const E3DTransformMatrix& other
 )

void E3DTransformMatrix(
 double m00,
 double m10,
 double m20,
 double m30,
 double m01,
 double m11,
 double m21,
 double m31,
 double m02,
 double m12,
 double m22,
 double m32,
 double m03,
 double m13,
 double m23,
 double m33
 )
```

## Parameters

*other*

-

*m00*

-

*m10*

-

*m20*

-

*m30*

-

*m01*

-

*m11*

-

*m21*

-

*m31*
-
*m02*
-
*m12*
-
*m22*
-
*m32*
-
*m03*
-
*m13*
-
*m23*
-
*m33*
-

### Remarks

The matrix is initialized with the given values.
By default, the matrix is initialized as an identity (neutral) matrix.
The value indices are m(column, row).

# E3DTransformMatrix::GetOrthoBasis

Gets the orthogonal basis represented by this transformation or throws an exception if it is not a rigid transformation.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void GetOrthoBasis(
 E3DPoint& e1,
 E3DPoint& e2,
 E3DPoint& e3,
 E3DPoint& t
 )
```

### Parameters

*e1*

Vector 1.

*e2*

Vector 2.

*e3*

Vector 3.

*t*

Translation.

# E3DTransformMatrix::GetValue

Gets a value from the E3DTransformMatrix object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetValue(
 OEV_UINT32 column,
 OEV_UINT32 row
 )
```

### Parameters

*column*

Column of the value to get, from 0 to 3.

*row*

Row of the value to get, from 0 to 3.

# E3DTransformMatrix::Inverse

Returns the inverted E3DTransformMatrix.
An exception will be thrown if the determinant of the matrix is **0**.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix Inverse(
 )
```

# E3DTransformMatrix::IsRigid

Checks that the transformation is a rigid transformation (keep the distances and angles).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool IsRigid(
 )
```

# E3DTransformMatrix::Load

Loads the E3DTransformMatrix object. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# E3DTransformMatrix::operator!=

Checks if two E3DTransformMatrix objects are stricly differents (binary level).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool operator!=(
 const E3DTransformMatrix& other
 )
```

### Parameters

*other*

The other matrix.

# E3DTransformMatrix::operator*

E3DTransformMatrix product. Combines the transformations of the two matrices.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix operator*(
 const E3DTransformMatrix& matrix
 )
E3DPoint operator*(
 E3DPoint P
 )
```

### Parameters

*matrix*

Matrix to combine with the current matrix.

*P*

Point to transform with the current matrix.

# E3DTransformMatrix::operator+

E3DTransformMatrix sum. Sums the current and the given matrix, returns the result.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix operator+(
 const E3DTransformMatrix& matrix
 )
```

### Parameters

*matrix*

Matrix to add with the current matrix.

# E3DTransformMatrix::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix& operator=(
 const E3DTransformMatrix& other
 )
```

## Parameters

*other*

An other E3DTransformMatrix.

# E3DTransformMatrix::operator==

Checks if two E3DTransformMatrix objects are stricly equals (binary level).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool operator==(
 const E3DTransformMatrix& other
 )
```

## Parameters

*other*

The other matrix.

# E3DTransformMatrix::Save

Saves the E3DTransformMatrix object. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# E3DTransformMatrix::SetValue

Sets a value in the E3DTransformMatrix object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetValue(
 OEV_UINT32 column,
 OEV_UINT32 row,
 float value
 )
```

### Parameters

*column*

Column of the value to set, from 0 to 3.

*row*

Row of the value to set, from 0 to 3.

*value*

Value to set.

## E3DTransformMatrix::Transpose

Returns the transposed E3DTransformMatrix.
If the matrix is orthogonal (rotation only transformation), the transposed matrix is the inverse transformation.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix Transpose(
 )
```

# 4.9. E3DViewer Class

Manages a viewer window for EPointCloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

### Methods

| | |
|---|---|
| ConfigureRenderSource | Sets the 3D source to be rendered. It replaces the current object. |
| E3DViewer | Creates an E3DViewer object. |

| | |
|---|---|
| GenerateColors | Generates the colors associated to the E3DViewer object to be rendered.<br>Colors are calculated from point coordinates, several mappings are exposed in EColorRampMode. |
| GetPointSize | Displays size of the points, in pixels. |
| GetRenderDecimationLevel | Point cloud decimation level during render. |
| GetRenderGrid | Enables or disables the display of Grid. |
| GetRenderGridStep | Enables or disables the display of Grid value. |
| GetWireframeMode | Enables or disables the display of wireframe triangles. |
| HideFeatureFor3DObject | Set the E3DObjectFeature of the registered E3DObject at the specified location idx to be not rendered. |
| HideFeatureForAll3DObjects | Set the E3DObjectFeature of all the registered E3DObject to be not rendered. |
| Register3DObjects | Set the list of E3DObject from which their features can be rendered. |
| RemoveCurrent3DObjects | Remove all E3DObject currently registered. |

| | |
|---|---|
| SetAutoRotate | Enables and configures the auto rotate display. |
| SetColors | Sets the colors associated to the E3DViewer object to be rendered. |
| SetFeatureStyleFor3DObject | Set how the E3DObjectFeature of the registered E3DObject at the specified location idx should be rendered. |
| SetFeatureStyleForAll3DObjects | Set how the E3DObjectFeature of all the registered E3DObject should be rendered. |
| SetFocus | The viewer window takes the focus. |
| SetPointSize | Displays size of the points, in pixels. |
| SetPosition | Sets the position of the 3D viewer window. |
| SetRenderDecimationLevel | Point cloud decimation level during render. |
| SetRenderGrid | Enables or disables the display of Grid. |
| SetRenderGridStep | Enables or disables the display of Grid value. |
| SetViewAngle | Sets view angle. |

| | |
|---|---|
| SetViewDistance | Sets view distance. |
| SetViewTarget | Sets view target position (by default, the camera position is 'look at center of the point cloud'). |
| SetWireframeMode | Enables or disables the display of wireframe triangles. |
| Show | Shows the viewer window. |
| ShowFeatureFor3DObject | Set the E3DObjectFeature of the registered E3DObject at the specified location idx to be rendered. |
| ShowFeatureForAll3DObjects | Set the E3DObjectFeature of all the registered E3DObject to be rendered. |
| StopAutoRotate | Stops the auto rotate display. |

# E3DViewer::SetColors

Sets the colors associated to the E3DViewer object to be rendered.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SetColors(const std::vector<Euresys::Open_eVision_2_10::EC24>& colourIn)
```

### Remarks

List of colors, and one color per point.

# E3DViewer::ConfigureRenderSource

Sets the 3D source to be rendered. It replaces the current object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void ConfigureRenderSource(
 const EPointCloud& sourceObject,
 bool keepCurrentView
 )

void ConfigureRenderSource(
 const EMesh& sourceObject,
 bool keepCurrentView
 )

void ConfigureRenderSource(
 const EZMap8& sourceObject,
 bool keepCurrentView
 )

void ConfigureRenderSource(
 const EZMap16& sourceObject,
 bool keepCurrentView
 )

void ConfigureRenderSource(
 const EZMap32f& sourceObject,
 bool keepCurrentView
 )
```

### Parameters

*sourceObject*

A 3D source (EPointCloud, EMesh, EZMap) to render.

*keepCurrentView*

An optional boolean, use TRUE to keep the current view or FALSE to reset the view and center the new object. The default value resets the view.

### Remarks

For display performance purposes, the object geometry is copied into the viewer.
Subsequent modifications on the object will thus not be visible until a new call to E3DViewer::ConfigureRenderSource has been made.
The initial viewing position looks at the object center.

# E3DViewer::E3DViewer

Creates an E3DViewer object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void E3DViewer(
 int orgX,
 int orgY,
 int width,
 int height,
 int parent
 )
void E3DViewer(
 const E3DViewer& other
 )
```

### Parameters

*orgX*

X coordinate of the top left corner of the viewer window.

*orgY*

Y coordinate of the top left corner of the viewer window.

*width*

Width of the viewer window.

*height*

Height of the viewer window.

*parent*

Handle of the parent window of the viewer. If NULL, the viewer is built as a independent floating window.

*other*

Another E3DViewer.

### Remarks

The origin point (orgX, orgY) defines the offset of the top left corner of the viewer from the top left corner of its parent window client area.
If the window has no parent, it defines the offset from the top left corner of the screen.
If the parent window is too small to contain the viewer, the viewer will be cropped accordingly.

# E3DViewer::GenerateColors

Generates the colors associated to the E3DViewer object to be rendered.
Colors are calculated from point coordinates, several mappings are exposed in EColorRampMode.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void GenerateColors(
 Euresys::Open_eVision_2_10::Easy3D::EColorRampMode mode
 )
```

### Parameters

*mode*

Mode used to compute colors.

### Remarks

The conversion used to generate colors from cloud point coordinates.

# E3DViewer::HideFeatureFor3DObject

Set the E3DObjectFeature of the registered E3DObject at the specified location idx to be not rendered.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void HideFeatureFor3DObject(
 int idx,
 Euresys::Open_eVision_2_10::Easy3D::E3DObjectFeature feature
 )
```

## Parameters

*idx*

the position in the list of registered E3DObject

*feature*

the feature

# E3DViewer::HideFeatureForAll3DObjects

Set the E3DObjectFeature of all the registered E3DObject to be not rendered.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void HideFeatureForAll3DObjects(
 Euresys::Open_eVision_2_10::Easy3D::E3DObjectFeature feature
 )
```

## Parameters

*feature*

the feature

# E3DViewer::GetPointSize

# E3DViewer::SetPointSize

Displays size of the points, in pixels.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
int GetPointSize() const
void SetPointSize(int size)
```

### Remarks

The size of the point (range value is **1** to **5** pixels, and **2** by default).
This value is used only to draw an EPointCloud, not for an EMesh.

# E3DViewer::Register3DObjects

Set the list of E3DObject from which their features can be rendered.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Register3DObjects(
 const std::vector<Euresys::Open_eVision_2_10::Easy3D::E3DObject>& objects
 )
```

### Parameters

*objects*

List of E3DObject

### Remarks

The features that need to be visualize are set with show methods. The registered features have a default style. Previous setted styles is ignored. Remove the currently registered E3DObject.

# E3DViewer::RemoveCurrent3DObjects

Remove all E3DObject currently registered.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void RemoveCurrent3DObjects(
 )
```

# E3DViewer::GetRenderDecimationLevel

# E3DViewer::SetRenderDecimationLevel

Point cloud decimation level during render.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
int GetRenderDecimationLevel() const
void SetRenderDecimationLevel(int decimationLevel)
```

### Remarks

The viewer will only render one point every [Decimation Level] points (**1** by default, and need to be > 0).
This decimation depends on the order of the points in the EPointCloud.

# E3DViewer::GetRenderGrid

# E3DViewer::SetRenderGrid

Enables or disables the display of Grid.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool GetRenderGrid() const
void SetRenderGrid(bool state)
```

### Remarks

Display Grid with true (true by default).

# E3DViewer::GetRenderGridStep

# E3DViewer::SetRenderGridStep

Enables or disables the display of Grid value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
float GetRenderGridStep() const

void SetRenderGridStep(float value)
```

### Remarks

The unit of the grid step value is the same as the one from the EPointCloud.
If value is smaller or equal to 0, Grid step will automatically be computed for 10 step on the X axe. (**0** by default).

# E3DViewer::SetAutoRotate

Enables and configures the auto rotate display.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetAutoRotate(
 float vx,
 float vy,
 float vz
 )
```

### Parameters

*vx*

Rotation speed around axis X.

*vy*

Rotation speed around axis Y.

*vz*

Rotation speed around axis Z.

# E3DViewer::SetFeatureStyleFor3DObject

Set how the E3DObjectFeature of the registered E3DObject at the specified location idx should be rendered.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetFeatureStyleFor3DObject(
 int idx,
 const ERenderStyle& style,
 Euresys::Open_eVision_2_10::Easy3D::E3DObjectFeature feature
 )
```

### Parameters

*idx*

-

*style*

the style

*feature*

the feature

# E3DViewer::SetFeatureStyleForAll3DObjects

Set how the E3DObjectFeature of all the registered E3DObject should be rendered.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetFeatureStyleForAll3DObjects(
 const ERenderStyle& style,
 Euresys::Open_eVision_2_10::Easy3D::E3DObjectFeature feature
 )
```

### Parameters

*style*

the style

*feature*

the feature

# E3DViewer::SetFocus

The viewer window takes the focus.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetFocus(
 )
```

# E3DViewer::SetPosition

Sets the position of the 3D viewer window.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetPosition(
 int orgX,
 int orgY,
 int width,
 int height
 )
```

## Parameters

*orgX*

X coordinate of the top left corner of the viewer window.

*orgY*

Y coordinate of the top left corner of the viewer window.

*width*

Width of the viewer window.

*height*

Height of the viewer window.

# E3DViewer::SetViewAngle

Sets view angle.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetViewAngle(
 float angleX,
 float angleY
 )
```

### Parameters

*angleX*

Rotation around the X axis.

*angleY*

Rotation around the Y axis.

# E3DViewer::SetViewTarget

Sets view target position (by default, the camera position is 'look at center of the point cloud').

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
void SetViewTarget(
 float targetX,
 float targetY,
 float targetZ
 )
```

### Parameters

*targetX*

X axis target position.

*targetY*

-

*targetZ*

-

# E3DViewer::Show

Shows the viewer window.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Show(
 )
```

# E3DViewer::ShowFeatureFor3DObject

Set the E3DObjectFeature of the registered E3DObject at the specified location idx to be rendered.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ShowFeatureFor3DObject(
 int idx,
 Euresys::Open_eVision_2_10::Easy3D::E3DObjectFeature feature
 )
```

### Parameters

*idx*

the position in the list of registered E3DObject

*feature*

the feature

# E3DViewer::ShowFeatureForAll3DObjects

Set the E3DObjectFeature of all the registered E3DObject to be rendered.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ShowFeatureForAll3DObjects(
 Euresys::Open_eVision_2_10::Easy3D::E3DObjectFeature feature
 )
```

### Parameters

*feature*

the feature

# E3DViewer::StopAutoRotate

Stops the auto rotate display.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void StopAutoRotate(
 )
```

# E3DViewer::SetViewDistance

Sets view distance.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetViewDistance(float distance)
```

## Remarks

Distance between the point of the view and the object.

# E3DViewer::GetWireframeMode

# E3DViewer::SetWireframeMode

Enables or disables the display of wireframe triangles.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool GetWireframeMode() const
void SetWireframeMode(bool state)
```

### Remarks

Display wireframe triangles with true (false by default).

# 4.10. EAffineTransformer Class

Manages a 3D coordinates transformation context.

### Remarks

By default, no transformation is done (identity matrix). The transformations are applied in the order in which the calls to AddTransform are done.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| AddAnisotropicScalingTransform | Adds anisotropic scaling to the current E3DTransformMatrix. |
| AddIsotropicScalingTransform | Adds isotropic scaling to the current E3DTransformMatrix. |
| AddOrthographicProjectionTransform | Adds an othographic projection to the current E3DTransformMatrix. |
| AddPerspectiveProjectionTransform | Adds a perspective projection to the current E3DTransformMatrix. |

| | |
|---|---|
| AddRotationXTransform | Adds rotation around the X axis to the current E3DTransformMatrix. |
| AddRotationYTransform | Adds rotation around the Y axis to the current E3DTransformMatrix. |
| AddRotationZTransform | Adds rotation around the Z axis to the current E3DTransformMatrix. |
| AddTransform | Composes a custom transformation with the current E3DTransformMatrix. |
| AddTranslationTransform | Adds translation to the current E3DTransformMatrix. |
| ApplyMatrix | Applies a E3DTransformMatrix to a EPointCloud or a points list.<br>If the second parameter is present, puts the transformed points in another point cloud or points list.<br>With a single parameter, the transformation is performed in place. |
| ApplyTransform | Applies the current transformation to a EPointCloud or a points list.<br>If the second parameter is present, puts the transformed points in another point cloud or points list.<br>With a single parameter, transformation is performed in place. |
| CreateAnisotropicScalingMatrix | Creates an anisotropic scaling E3DTransformMatrix. |
| CreateIdentityMatrix | Creates an indentity (neutral) E3DTransformMatrix. |

| | |
|---|---|
| CreateIsotropicScalingMatrix | Creates an isotropic scaling E3DTransformMatrix. |
| CreateOrthographicProjectionMatrix | Creates an othographic projection E3DTransformMatrix. |
| CreatePerspectiveProjectionMatrix | Creates a perspective projection E3DTransformMatrix. |
| CreateRotationXMatrix | Creates a rotation E3DTransformMatrix around the X axis. |
| CreateRotationYMatrix | Creates a rotation E3DTransformMatrix around the Y axis. |
| CreateRotationZMatrix | Creates a rotation E3DTransformMatrix around the Z axis. |
| CreateTranslationMatrix | Creates a translation E3DTransformMatrix. |
| EAffineTransformer | Creates an EAffineTransformer object. |
| GetTransform | Sets the E3DTransformMatrix that will be used. |
| operator= | Assignment operator. |
| Reset | Resets the transformation E3DTransformMatrix to the identity. |
| SetTransform | Sets the E3DTransformMatrix that will be used. |

# EAffineTransformer::AddAnisotropicScalingTransform

Adds anisotropic scaling to the current E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const E3DTransformMatrix& AddAnisotropicScalingTransform(
 float scaleX,
 float scaleY,
 float scaleZ
 )
```

### Parameters

*scaleX*

Scaling factor along the X axis.

*scaleY*

Scaling factor along the Y axis.

*scaleZ*

Scaling factor along the Z axis.

# EAffineTransformer::AddIsotropicScalingTransform

Adds isotropic scaling to the current E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const E3DTransformMatrix& AddIsotropicScalingTransform(
 float scale
 )
```

### Parameters

*scale*

Scaling factor.

# EAffineTransformer::AddOrthographicProjectionTransform

Adds an othographic projection to the current E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const E3DTransformMatrix& AddOrthographicProjectionTransform(
 float width,
 float height
 )
```

### Parameters

*width*

Width of the viewport.

*height*

Height of the viewport.

# EAffineTransformer::AddPerspectiveProjectionTransform

Adds a perspective projection to the current E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
const E3DTransformMatrix& AddPerspectiveProjectionTransform(
 float distance,
 float width,
 float height
 )
```

### Parameters

*distance*

Distance of the viewport to the origin.

*width*

Width of the viewport.

*height*

Height of the viewport.

# EAffineTransformer::AddRotationXTransform

Adds rotation around the X axis to the current E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
const E3DTransformMatrix& AddRotationXTransform(
 float Angle
 )
```

### Parameters

*Angle*

Rotation angle.

# EAffineTransformer::AddRotationYTransform

Adds rotation around the Y axis to the current E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const E3DTransformMatrix& AddRotationYTransform(
 float Angle
 )
```

### Parameters

*Angle*

Rotation angle.

# EAffineTransformer::AddRotationZTransform

Adds rotation around the Z axis to the current E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const E3DTransformMatrix& AddRotationZTransform(
 float Angle
 )
```

### Parameters

*Angle*

Rotation angle.

# EAffineTransformer::AddTransform

Composes a custom transformation with the current E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const E3DTransformMatrix& AddTransform(
 const E3DTransformMatrix& matrix
 )
```

## Parameters

*matrix*

Transformation matrix.

# EAffineTransformer::AddTranslationTransform

Adds translation to the current E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const E3DTransformMatrix& AddTranslationTransform(
 float dX,
 float dY,
 float dZ
 )
```

## Parameters

*dX*

Translation along the X axis.

*dY*

Translation along the Y axis.

*dZ*

Translation along the Z axis.

# EAffineTransformer::ApplyMatrix

Applies a E3DTransformMatrix to a EPointCloud or a points list.
If the second parameter is present, puts the transformed points in another point cloud or points list.
With a single parameter, the transformation is performed in place.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ApplyMatrix(
 const E3DTransformMatrix& matrix,
 const EPointCloud& cloud,
 EPointCloud& transformedCloud
 )

void ApplyMatrix(
 const E3DTransformMatrix& matrix,
 EPointCloud& cloud
 )

void ApplyMatrix(
 const E3DTransformMatrix& matrix,
 const std::vector<Euresys::Open_eVision_2_10::Easy3D::E3DPoint>& sourcePoints,
 std::vector<Euresys::Open_eVision_2_10::Easy3D::E3DPoint>& transformedPoints
 )
```

## Parameters

*matrix*

Transformation matrix.

*cloud*

Cloud to transform.

*transformedCloud*

Transformed cloud.

*sourcePoints*

Points list to transform.

*transformedPoints*

Transformed points list.

# EAffineTransformer::ApplyTransform

Applies the current transformation to a EPointCloud or a points list.
If the second parameter is present, puts the transformed points in another point cloud or points list.
With a single parameter, transformation is performed in place.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ApplyTransform(
 const EPointCloud& cloud,
 EPointCloud& transformedCloud
 )

void ApplyTransform(
 EPointCloud& cloud
 )

void ApplyTransform(
 const std::vector<Euresys::Open_eVision_2_10::Easy3D::E3DPoint>& sourcePoints,
 std::vector<Euresys::Open_eVision_2_10::Easy3D::E3DPoint>& transformedPoints
 )
```

## Parameters

*cloud*

Cloud to transform.

*transformedCloud*

Transformed cloud.

*sourcePoints*

Points list to transform.

*transformedPoints*

Transformed points list.

# EAffineTransformer::CreateAnisotropicScalingMatrix

Creates an anisotropic scaling E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix CreateAnisotropicScalingMatrix(
 float scaleX,
 float scaleY,
 float scaleZ
 )
```

### Parameters

*scaleX*

Scaling factor along the X axis.

*scaleY*

Scaling factor along the Y axis.

*scaleZ*

Scaling factor along the Z axis.

# EAffineTransformer::CreateIdentityMatrix

Creates an indentity (neutral) E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix CreateIdentityMatrix(
 )
```

# EAffineTransformer::CreateIsotropicScalingMatrix

Creates an isotropic scaling E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix CreateIsotropicScalingMatrix(
 float scale
 )
```

### Parameters

*scale*

Scaling factor.

# EAffineTransformer::CreateOrthographicProjectionMatrix

Creates an othographic projection E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix CreateOrthographicProjectionMatrix(
 float width,
 float height
 )
```

### Parameters

*width*

Width of the viewport.

*height*

Height of the viewport.

# EAffineTransformer::CreatePerspectiveProjectionMatrix

Creates a perspective projection E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix CreatePerspectiveProjectionMatrix(
 float distance,
 float width,
 float height
 )
```

## Parameters

*distance*

Distance of the viewport to the origin.

*width*

Width of the viewport.

*height*

Height of the viewport.

# EAffineTransformer::CreateRotationXMatrix

Creates a rotation E3DTransformMatrix around the X axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
E3DTransformMatrix CreateRotationXMatrix(
 float Angle
 )
```

### Parameters

*Angle*

Rotation angle.

# EAffineTransformer::CreateRotationYMatrix

Creates a rotation E3DTransformMatrix around the Y axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix CreateRotationYMatrix(
 float Angle
 )
```

### Parameters

*Angle*

Rotation angle.

# EAffineTransformer::CreateRotationZMatrix

Creates a rotation E3DTransformMatrix around the Z axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix CreateRotationZMatrix(
 float Angle
 )
```

### Parameters

*Angle*

Rotation angle.

# EAffineTransformer::CreateTranslationMatrix

Creates a translation E3DTransformMatrix.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix CreateTranslationMatrix(
 float dX,
 float dY,
 float dZ
 )
```

### Parameters

*dX*

Translation along the X axis.

*dY*

Translation along the Y axis.

*dZ*

Translation along the Z axis.

# EAffineTransformer::EAffineTransformer

Creates an EAffineTransformer object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EAffineTransformer(
 )
void EAffineTransformer(
 const EAffineTransformer& other
 )
```

### Parameters

*other*

Another EAffineTransformer.

# EAffineTransformer::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EAffineTransformer& operator=(
 const EAffineTransformer& other
 )
```

### Parameters

*other*

Another EAffineTransformer.

## EAffineTransformer::Reset

Resets the transformation E3DTransformMatrix to the identity.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Reset(
 )
```

## EAffineTransformer::GetTransform

## EAffineTransformer::SetTransform

Sets the E3DTransformMatrix that will be used.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const E3DTransformMatrix& GetTransform() const
void SetTransform(const E3DTransformMatrix& matrix)
```

# 4.11. EAngleRectifier Class

-

**Namespace:** Euresys::Open_eVision_2_10

## Methods

Rectify

E-

A

# ngleRectifier::Rectify

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
double Rectify(
 double angle,
 double mean
 )

double Rectify(
 double angle,
 double mean,
 Euresys::Open_eVision_2_10::EAngleUnit currentUnit
 )
```

### Parameters

*angle*

-

*mean*

-

*currentUnit*

-

# 4.12. Easy Class

This class contains static properties and methods specific to the Easy library.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| CheckLicense | Checks if a given license is available. |
| CheckLicenses | Check if at least one license is available. Otherwise, an exception is thrown. |
| CheckOemKey | Checks if the OEM key, if any, matches a given argument. |
| CloseImageGraphicContext | Releases the device context associated to an image. |
| FromRadians | Returns the angle, converted from radians to the current angle unit. |
| GetAngleUnit | Current angular unit. |
| GetBestMatchingImageType | Returns the best matching image type for a given file on disk. |
| GetDongleCount | Get the number of available dongle on the system. |

| | |
|---|---|
| GetDongleInternalSerialNumber | Get the serial number of the selected dongle. |
| GetErrorText | Returns the description associated to a given error code. |
| GetMaxNumberOfProcessingThreads | Maximum number of threads used internally by the Open eVision tools. This number cannot be higher than the number of processor cores available to the system. See Easy::NumberOfAvailableProcessorCores. |
| GetNumberOfAvailableProcessorCores | Number of processor cores available to the system. This is the upper limit for the number of threads usable ibternally by the Open eVision tools. See Easy::MaxNumberOfProcessingThreads |
| GetVersion | Returns a pointer to a **NULL** terminated character string that contains the current version number of Open eVision. |
| OpenImageGraphicContext | Allows to draw in a gray-level or a color image, using any of the Windows device context functions (the image contents is altered, allowing destructive overlays). |
| Render3D | Prepares a three dimensional rendering of an image where the gray-level values are taken for altitudes. |
| RenderColorHistogram | Prepares a three dimensional rendering of the histogram of a color image: the pixels are drawn in the RGB space rather than in the XY plane. |

| | |
|---|---|
| Resize | Resizes an image without interpolation. |
| SetAngleUnit | Current angular unit. |
| SetMaxNumberOfProcessingThreads | Maximum number of threads used internally by the Open eVision tools. This number cannot be higher than the number of processor cores available to the system. See Easy::NumberOfAvailableProcessorCores. |
| SetOemKey | Writes an OEM key value on a dongle. |
| StartTiming | Starts timing, using the system clock or performance counter. |
| StopTiming | Returns the time, in specified time units, elapsed since the last invocation of Easy::StartTiming. |
| Terminate | Method not obligatory, but necessary for close dll cleanly. |
| ToRadians | Returns the angle, converted from current angle unit to radians. |
| TrueTimingResolution | Returns the actual resolution of the timing clock, in ticks per seconds. |

# Easy::GetAngleUnit

# Easy::SetAngleUnit

Current angular unit.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
static Euresys::Open_eVision_2_10::EAngleUnit GetAngleUnit()
static void SetAngleUnit(Euresys::Open_eVision_2_10::EAngleUnit unit)
```

## Remarks

All angles are computed using some angular unit, as well on input as on output. The desired unit can be changed at any time. By default, all angles are given in degrees (**0..360**).

# Easy::CheckLicense

Checks if a given license is available.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool CheckLicense(
 Euresys::Open_eVision_2_10::LicenseFeatures::Features license
 )
```

## Parameters

*license*

The license to check

# Easy::CheckLicenses

Check if at least one license is available. Otherwise, an exception is thrown.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void CheckLicenses(
 )
```

# Easy::CheckOemKey

Checks if the OEM key, if any, matches a given argument.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL CheckOemKey(
 const std::vector<char>& key,
 int index
 )
```

## Parameters

*key*

The expected value of the OEM key

*index*

The index of the dongle where the OEM key is expected. By default, the first dongle found is selected.

## Remarks

The length of the OEM key must be exactly 8 characters.

# Easy::CloseImageGraphicContext

Releases the device context associated to an image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void CloseImageGraphicContext(
 EImageBW8* pImage,
 HDC hDC
 )
void CloseImageGraphicContext(
 EImageC24* pImage,
 HDC hDC
 )
```

### Parameters

*pImage*

Pointer to the target image (must be the same as that passed to Easy::OpenImageGraphicContext).

*hDC*

Handle to a device context that was produced by Easy::OpenImageGraphicContext.

# Easy::GetDongleCount

Get the number of available dongle on the system.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
static OEV_UINT32 GetDongleCount()
```

# Easy::FromRadians

Returns the angle, converted from radians to the current angle unit.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float FromRadians(
 float angle
 )
```

### Parameters

*angle*

Angle to be converted

# Easy::GetBestMatchingImageType

Returns the best matching image type for a given file on disk.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EImageType GetBestMatchingImageType(
 const std::string& path
 )
```

### Parameters

*path*

The path to the file on disk.

# Easy::GetDongleInternalSerialNumber

Get the serial number of the selected dongle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::string GetDongleInternalSerialNumber(
 int index
 )
```

## Parameters

*index*

The index of the dongle.

# Easy::GetErrorText

Returns the description associated to a given error code.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::string GetErrorText(
 Euresys::Open_eVision_2_10::EError error
 )
```

## Parameters

*error*

Error code.

# Easy::GetMaxNumberOfProcessingThreads

# Easy::SetMaxNumberOfProcessingThreads

Maximum number of threads used internally by the Open eVision tools. This number cannot be higher than the number of processor cores available to the system. See Easy::NumberOfAvailableProcessorCores.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
static int GetMaxNumberOfProcessingThreads()
static void SetMaxNumberOfProcessingThreads(int maxNumThreads)
```

# Easy::GetNumberOfAvailableProcessorCores

Number of processor cores available to the system. This is the upper limit for the number of threads usable ibternally by the Open eVision tools. See Easy::MaxNumberOfProcessingThreads

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
static int GetNumberOfAvailableProcessorCores()
```

# Easy::OpenImageGraphicContext

Allows to draw in a gray-level or a color image, using any of the Windows device context functions (the image contents is altered, allowing destructive overlays).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
HDC OpenImageGraphicContext(
 EImageBW8* pImage
 )
HDC OpenImageGraphicContext(
 EImageC24* pImage
 )
```

### Parameters

*pImage*

Pointer to the target image.

### Remarks

The function returns a handle to a device context associated to the image pixel data. When the device context is no more needed, call the Easy::CloseImageGraphicContext function with the same argument.

# Easy::Render3D

Prepares a three dimensional rendering of an image where the gray-level values are taken for altitudes.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void Render3D(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 float phi,
 float psi,
 float xScale,
 float yScale,
 float zScale,
 OEV_INT32 dotSize
 )
void Render3D(
 EROIC24* sourceImage,
 EROIBW8* zImage,
 EROIC24* destinationImage,
 float phi,
 float psi,
 float xScale,
 float yScale,
 float zScale,
 OEV_INT32 dotSize
 )
```

## Parameters

*sourceImage*

Pointer to the source image.

*destinationImage*

Pointer to the destination image.

*phi*

Rotation angle about the X-axis.

*psi*

Rotation angle about the Y-axis.

*xScale*

Magnification factor along X (should remain close to **1**).

*yScale*

Magnification factor along Y (should remain close to **1**).

*zScale*

Magnification factor along Z (should remain close to **1**).

*dotSize*

Size of the rendered dots; allowed values are **1**, **4**, **5** or **9**.

*zImage*

Pointer to the altitude image.

### Remarks

The image is viewed by rotating it about the X-axis, then about the Y-axis. Magnification factors in the three directions (X = width, Y = height, and Z = depth) can be given.
The rendered image appears as independent dots. The dot size can be adjusted so that the surface appears more or less opaque.
The function does not display the rendered image by itself. Rather, it prepares a destination image to be displayed.

# Easy::RenderColorHistogram

Prepares a three dimensional rendering of the histogram of a color image: the pixels are drawn in the RGB space rather than in the XY plane.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void RenderColorHistogram(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 float phi,
 float psi,
 float xScale,
 float yScale,
 float zScale
 )

void RenderColorHistogram(
 EROIC24* sourceImage,
 EROIC24* sysImage,
 EROIC24* destinationImage,
 float phi,
 float psi,
 float xScale,
 float yScale,
 float zScale
 )
```

### Parameters

*sourceImage*

Pointer to the raw source image.

*destinationImage*

Pointer to the destination image.

*phi*

Rotation angle about the X-axis.

*psi*

Rotation angle about the Y-axis.

*xScale*

Magnification factor along X (should remain close to **1**).

*yScale*

Magnification factor along Y (should remain close to **1**).

*zScale*

Magnification factor along Z (should remain close to **1**).

*sysImage*

Pointer to the source image transformed into another color system.

### Remarks

This allows to observe the clustering and dispersion of the RGB values.
The image is viewed by rotating it about the X-axis, then about the Y-axis. Magnification factors in the three directions (X = Red, Y = Green, and Z = Blue) can be given.
In a more advanced version, prepares a three dimensional rendering of the pixels in another system than RGB (EasyColor provides conversion means). However, the raw RGB image must still be provided to allow the display of the pixels in their usual colors.
The rendered image appears as independent dots.
The function does not display the rendered image by itself. Rather, it prepares a destination image to be displayed.

# Easy::Resize

Resizes an image without interpolation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Resize(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )

void Resize(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )

void Resize(
 EROIC15* sourceImage,
 EROIC15* destinationImage
 )

void Resize(
 EROIC16* sourceImage,
 EROIC16* destinationImage
 )

void Resize(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )

void Resize(
 EROIC24A* sourceImage,
 EROIC24A* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

# Easy::SetOemKey

Writes an OEM key value on a dongle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetOemKey(
 const std::vector<char>& key,
 int index
 )
```

### Parameters

*key*

The OEM key value to write

*index*

The index of the dongle where the OEM key must be written. By default, the first dongle found is selected.

### Remarks

The length of the OEM key must be exactly 8 characters. This method raises an EError_CannotWriteOEMKey error if the value cannot be set properly.

# Easy::StartTiming

Starts timing, using the system clock or performance counter.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void StartTiming(
 )
```

# Easy::StopTiming

Returns the time, in specified time units, elapsed since the last invocation of Easy::StartTiming.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 StopTiming(
 OEV_INT32 resolution
 )
```

### Parameters

*resolution*

Temporal resolution, in ticks per second.

# Easy::Terminate

Method not obligatory, but necessary for close dll cleanly.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Terminate(
 )
```

# Easy::ToRadians

Returns the angle, converted from current angle unit to radians.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
float ToRadians(
 float angle
 )
```

### Parameters

*angle*

Angle to be converted.

# Easy::TrueTimingResolution

Returns the actual resolution of the timing clock, in ticks per seconds.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 TrueTimingResolution(
 )
```

### Remarks

Timing granularity is hardware-dependent, but is usually better than 1 Âµs.
This function can be used to select an appropriate timing resolution when using Easy::StopTiming.

# Easy::GetVersion

Returns a pointer to a **NULL** terminated character string that contains the current version number of Open eVision.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
static std::string GetVersion()
```

# 4.13. EasyColor Class

This class contains static properties and methods specific to the EasyColor library.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AlphaBlend | Draws an image over an other. |
| AssignNearestClass | Assigns to every pixel of the source image the nearest class index *plus one* and stores its value in the destination image. |
| AssignNearestClassCenter | Assigns to every pixel of the source image the nearest class center and stores its value in the destination image. |
| BayerToC24 | Converts a Bayer pattern encoded image into a color image. |
| C24ToBayer | Converts a color image into a Bayer pattern encoded image. |
| ClassAverages | Computes the average source pixel colors for every class separately. |

| | |
|---|---|
| ClassVariances | Computes the averages and variances of the image pixel colors for every class separately. |
| Compose | Combines three gray-level images, considered as three color planes, into a color image. |
| Decompose | Extracts the three color planes, considered as gray-level images, from a color image. |
| Dequantize | Convert a quantized value to a unquantized value of a given color system. |
| Format422To444 | Converts a YUV 4:2:2 image to a YUV 4:4:4 image using interpolation. |
| Format444To422 | Converts a YUV 4:4:4 image to a YUV 4:2:2 image using filtering |
| GetCieAB | CIE AB white illuminant |
| GetCieAG | CIE AG white illuminant |
| GetCieAR | CIE AR white illuminant |
| GetCieD50B | CIE D50B white illuminant |

| | |
|---|---|
| GetCieD50G | CIE D50G white illuminant |
| GetCieD50R | CIE D50R white illuminant |
| GetCieD55B | CIE D55B white illuminant |
| GetCieD55G | CIE D55G white illuminant |
| GetCieD55R | CIE D55R white illuminant |
| GetCieD65B | CIE D65B white illuminant |
| GetCieD65G | CIE D65G white illuminant |
| GetCieD65R | CIE D65R white illuminant |
| GetCieFB | CIE FB white illuminant |
| GetCieFG | CIE FG white illuminant |
| GetCieFR | CIE FR white illuminant |
| GetCompensateNtscGamma | NTSC inverse gamma exponent |

| | |
|---|---|
| GetCompensatePalGamma | PAL inverse gamma exponent |
| GetCompensateSmpteGamma | NTSC inverse gamma exponent |
| GetComponent | Extracts one color plane, considered as a gray-level image, from a color image. |
| GetDstQuantization | Quantization mode for output values. |
| GetNtscGamma | NTSC gamma exponent |
| GetPalGamma | PAL gamma exponent |
| GetRgbStandard | RGB definition to be used when converting between RGB and other color systems. |
| GetSmpteGamma | SMPTE gamma exponent |
| GetSrcQuantization | Quantization mode for input values. |
| ImproveClassCenters | Redefines the class centers by computing the average color value of the pixels assigned to each class in the source image. |
| IshToRgb | Convert a color from any system to RGB. |

| | |
|---|---|
| LabToRgb | Convert a color from any system to RGB. |
| LabToXyz | Convert a color from one system to another, including the xyz variant (reduced XYZ). |
| LchToRgb | Convert a color from any system to RGB. |
| LshToRgb | Convert a color from any system to RGB. |
| LuvToRgb | Convert a color from any system to RGB. |
| LuvToXyz | Convert a color from one system to another, including the xyz variant (reduced XYZ). |
| PseudoColor | Applies the mapping defined by the pseudo-color lookup table to transform a gray-level image into a color image. |
| Quantize | Convert an unquantized color of a given color system to a quantized color. |
| RegisterPlanes | Sets a color plane of a color image by using a gray-level image as component. |

| | |
|---|---|
| RgbToIsh | Convert a color from RGB to another system. |
| RgbToLab | Convert a color from RGB to another system. |
| RgbToLch | Convert a color from RGB to another system. |
| RgbToLsh | Convert a color from RGB to another system. |
| RgbToLuv | Convert a color from RGB to another system. |
| RgbToReducedXyz | Convert a color from one system to another, including the xyz variant (reduced XYZ). |
| RgbToVsh | Convert a color from RGB to another system. |
| RgbToXyz | Convert a color from RGB to another system. |
| RgbToYiq | Convert a color from RGB to another system. |
| RgbToYsh | Convert a color from RGB to another system. |
| RgbToYuv | Convert a color from RGB to another system. |

| | |
|---|---|
| SetComponent | Sets a color plane of a color image by using a gray-level image as component. |
| SetDstQuantization | Quantization mode for output values. |
| SetRgbStandard | RGB definition to be used when converting between RGB and other color systems. |
| SetSrcQuantization | Quantization mode for input values. |
| Transform | Applies a color transformation to a specified image. |
| TransformBayer | Converts an image, using the transformation defined by a color lookup. |
| VshToRgb | Convert a color from any system to RGB. |
| XyzToLab | Convert a color from one system to another, including the xyz variant (reduced XYZ). |
| XyzToLuv | Convert a color from one system to another, including the xyz variant (reduced XYZ). |

| | |
|---|---|
| XyzToRgb | Convert a color from any system to RGB. |
| YiqToRgb | Convert a color from any system to RGB. |
| YshToRgb | Convert a color from any system to RGB. |
| YuvToRgb | Convert a color from any system to RGB. |

E

a

# syColor::AlphaBlend

Draws an image over an other.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AlphaBlend(
 EROIC24& source,
 EROIC24& destination,
 double opacity
 )
```

## Parameters

*source*

Foreground image.

*destination*

Background image.

*opacity*

Opacity of the foreground image.

# EasyColor::AssignNearestClass

Assigns to every pixel of the source image the nearest class index *plus one* and stores its value in the destination image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AssignNearestClass(
 EROIC24* sourceImage,
 EROIBW8* destinationImage,
 EC24Vector* classCenters
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination gray-level image/ROI.

*classCenters*

Pointer to the vector of the class centers.

## Remarks

This generates a labeled gray-level image for use with EasyObject (see EImageEncoder and ELabeledImageSegmenter).

**Note.** The class index plus one is stored instead of the class index because EasyObject will never code class 0 objects.
Color image segmentation allows you to decompose a color image in different regions by assigning a "class" (integer index) to every pixel. The nearest neighbor method is used, i.e. for each class a representative center is specified, and a given pixel is associated to the class with the closest center.
Color image segmentation can be used in conjunction with EasyObject to perform blob analysis on the segmented regions.
To use the color segmentation functions, the set of class centers must be specified as a vector of EC24 elements. In this sense, the method is termed supervised clustering. A good way to obtain these values is to compute the average color in an ROI.
Unsupervised clustering is also made available by implementing the so called K-means method that automatically improves an initial choice of class centers.

# EasyColor::AssignNearestClassCenter

Assigns to every pixel of the source image the nearest class center and stores its value in the destination image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AssignNearestClassCenter(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 EC24Vector* classCenters
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*classCenters*

Pointer to the vector of the class centers.

## Remarks

This generates a labeled color image.
Color image segmentation allows you to decompose a color image in different regions by assigning a "class" (integer index) to every pixel. The nearest neighbor method is used, i.e. for each class a representative center is specified, and a given pixel is associated to the class with the closest center.
Color image segmentation can be used in conjunction with EasyObject to perform blob analysis on the segmented regions.
To use the color segmentation functions, the set of class centers must be specified as a vector of EC24 elements. In this sense, the method is termed supervised clustering. A good way to obtain these values is to compute the average color in an ROI.
Unsupervised clustering is also made available by implementing the so called K-means method that automatically improves an initial choice of class centers.

# EasyColor::BayerToC24

Converts a Bayer pattern encoded image into a color image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BayerToC24(
 EROIBW8* sourceImage,
 EROIC24* destinationImage,
 BOOL evenCol,
 BOOL evenRow,
 BOOL interpolate,
 BOOL improved
 )
```

## Parameters

*sourceImage*

Pointer to the Bayer pattern input image/ROI, stored using the 8 bits per pixel format.

*destinationImage*

Pointer to the color output image/ROI.

*evenCol*

**TRUE** if the leftmost image column contains no blue pixels.

*evenRow*

**TRUE** if the topmost image row contains no red pixels.

*interpolate*

Interpolation mode to be used for pixel reconstruction. When **FALSE**, the missing color components are merely copied from northern/western pixels; when **TRUE**, they are computed by averaging from relevant neighbors. By default, interpolation is used.

*improved*

Provides an access to an improved interpolation mode that reduces visible artifacts along object edges. The running time of the improved method is longer. By default, it is not used.

## Remarks

The pattern can be shifted by one pixel horizontally and vertically as needed to deal with a non standard pattern origin. See also Bayer Filter.

# EasyColor::C24ToBayer

Converts a color image into a Bayer pattern encoded image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void C24ToBayer(
 EROIC24* sourceImage,
 EROIBW8* destinationImage,
 BOOL evenCol,
 BOOL evenRow
 )
```

## Parameters

*sourceImage*

Pointer to the color input image/ROI.

*destinationImage*

Pointer to the Bayer pattern output image/ROI, stored using the 8 bits per pixel format.

*evenCol*

**TRUE** if the leftmost destination image column can't contain blue pixels.

*evenRow*

**TRUE** if the topmost destination image row can't contain red pixels.

## Remarks

The pattern can be shifted by one pixel horizontally and vertically as needed to deal with a non standard pattern origin.
See also Bayer Filter.

# EasyColor::GetCieAB

CIE AB white illuminant

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
static float GetCieAB()
```

# EasyColor::GetCieAG

CIE AG white illuminant

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
static float GetCieAG()
```

# EasyColor::GetCieAR

CIE AR white illuminant

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
static float GetCieAR()
```

# EasyColor::GetCieD50B

CIE D50B white illuminant

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
static float GetCieD50B()
```

# EasyColor::GetCieD50G

CIE D50G white illuminant

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
static float GetCieD50G()
```

# EasyColor::GetCieD50R

CIE D50R white illuminant

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
static float GetCieD50R()
```

# EasyColor::GetCieD55B

CIE D55B white illuminant

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
static float GetCieD55B()
```

# EasyColor::GetCieD55G

CIE D55G white illuminant

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
static float GetCieD55G()
```

# EasyColor::GetCieD55R

CIE D55R white illuminant

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
static float GetCieD55R()
```

# EasyColor::GetCieD65B

CIE D65B white illuminant

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
static float GetCieD65B()
```

# EasyColor::GetCieD65G

CIE D65G white illuminant

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
static float GetCieD65G()
```

# EasyColor::GetCieD65R

CIE D65R white illuminant

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
static float GetCieD65R()
```

# EasyColor::GetCieFB

CIE FB white illuminant

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
static float GetCieFB()
```

# EasyColor::GetCieFG

CIE FG white illuminant

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
static float GetCieFG()
```

# EasyColor::GetCieFR

CIE FR white illuminant

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
static float GetCieFR()
```

# EasyColor::ClassAverages

Computes the average source pixel colors for every class separately.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ClassAverages(
 EROIC24* sourceImage,
 EC24Vector* classCenters,
 EColorVector* averages
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*classCenters*

Pointer to the vector of the class centers.

*averages*

Pointer to the vector of the average color values.

### Remarks

This allows measuring the actual average color of the segmented regions.
Color image segmentation allows you to decompose a color image in different regions by assigning a "class" (integer index) to every pixel. The nearest neighbor method is used, i.e. for each class a representative center is specified, and a given pixel is associated to the class with the closest center.
Color image segmentation can be used in conjunction with EasyObject to perform blob analysis on the segmented regions.
To the color segmentation functions, the set of class centers must be specified as a vector of EC24 elements. In this sense, the method is termed supervised clustering. A good way to obtain these values is to compute the average color in an ROI.
Unsupervised clustering is also made available by implementing the so called K-means method that automatically improves an initial choice of class centers.

# EasyColor::ClassVariances

Computes the averages and variances of the image pixel colors for every class separately.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ClassVariances(
 EROIC24* sourceImage,
 EC24Vector* classCenters,
 EColorVector* averages,
 EColorVector* variances
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*classCenters*

Pointer to the vector of the class centers.

*averages*

Pointer to the vector of the average color values.

*variances*

Pointer to the vector of the variance color values.

### Remarks

This allows quantifying the homogeneity of the segmented regions.
Color image segmentation allows you to decompose a color image in different regions by assigning a "class" (integer index) to every pixel. The nearest neighbor method is used, i.e. for each class a representative center is specified, and a given pixel is associated to the class with the closest center. Color image segmentation can be used in conjunction with EasyObject to perform blob analysis on the segmented regions.
To the color segmentation functions, the set of class centers must be specified as a vector of EC24 elements. In this sense, the method is termed supervised clustering. A good way to obtain these values is to compute the average color in an ROI.
Unsupervised clustering is also made available by implementing the so called K-means method that automatically improves an initial choice of class centers.

# EasyColor::GetCompensateNtscGamma

NTSC inverse gamma exponent

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
static float GetCompensateNtscGamma()
```

# EasyColor::GetCompensatePalGamma

PAL inverse gamma exponent

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
static float GetCompensatePalGamma()
```

# EasyColor::GetCompensateSmpteGamma

NTSC inverse gamma exponent

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
static float GetCompensateSmpteGamma()
```

# EasyColor::Compose

Combines three gray-level images, considered as three color planes, into a color image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Compose(
 EROIBW8* sourceImageOfColor0,
 EROIBW8* sourceImageOfColor1,
 EROIBW8* sourceImageOfColor2,
 EROIC24* colorDestinationImage,
 EColorLookup* lookup
 )

void Compose(
 EROIBW16* sourceImageOfColor0,
 EROIBW16* sourceImageOfColor1,
 EROIBW16* sourceImageOfColor2,
 EROIC48* colorDestinationImage
 )
```

### Parameters

*sourceImageOfColor0*

Pointers to the three input gray-level component images/ROIs.

*sourceImageOfColor1*

Pointers to the three input gray-level component images/ROIs.

*sourceImageOfColor2*

Pointers to the three input gray-level component images/ROIs.

*colorDestinationImage*

Pointer to the output color image/ROI.

*lookup*

Pointer to the color lookup table, or **NULL**.

### Remarks

If a color lookup is used, the resulting image undergoes the corresponding color transform. This way, it is possible to build an RGB image from the color planes of another system, or conversely.
A color image can be seen as a set of three color planes, each corresponding to a color component. The color planes are themselves continuous tone images. An EImageC24 contains three EImageBW8.

# EasyColor::Decompose

Extracts the three color planes, considered as gray-level images, from a color image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Decompose(
 EROIC24* colorSourceImage,
 EROIBW8* destinationImageOfColor0,
 EROIBW8* destinationImageOfColor1,
 EROIBW8* destinationImageOfColor2,
 EColorLookup* lookup
 )

void Decompose(
 EROIC48* colorSourceImage,
 EROIBW16* destinationImageOfColor0,
 EROIBW16* destinationImageOfColor1,
 EROIBW16* destinationImageOfColor2
 )
```

### Parameters

*colorSourceImage*

Pointer to the input color image/ROI.

*destinationImageOfColor0*

Pointers to the three output gray level component images/ROIs.

*destinationImageOfColor1*

Pointers to the three output gray level component images/ROIs.

*destinationImageOfColor2*

Pointers to the three output gray level component images/ROIs.

*lookup*

Pointer to the color lookup table, or **NULL**.

### Remarks

If a color lookup is used, the source image undergoes the corresponding color transform. This way, it is possible to get the RGB components from an image of another system, of conversely.
A color image can be seen as a set of three color planes, each corresponding to a color component. The color planes are themselves continuous tone images. An EImageC24 contains three EImageBW8.

# EasyColor::Dequantize

Convert a quantized value to a unquantized value of a given color system.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Dequantize(
 EC24 colorIn,
 ERGB& colorOut
 )

void Dequantize(
 EC24 colorIn,
 EXYZ& colorOut
 )

void Dequantize(
 EC24 colorIn,
 EYUV& colorOut
 )

void Dequantize(
 EC24 colorIn,
 EYIQ& colorOut
 )

void Dequantize(
 EC24 colorIn,
 ELSH& colorOut
 )

void Dequantize(
 EC24 colorIn,
 EVSH& colorOut
 )

void Dequantize(
 EC24 colorIn,
 EISH& colorOut
 )

void Dequantize(
 EC24 colorIn,
 EYSH& colorOut
 )

void Dequantize(
 EC24 colorIn,
 ELAB& colorOut
 )
```

```
void Dequantize(
 EC24 colorIn,
 ELCH& colorOut
 )

void Dequantize(
 EC24 colorIn,
 ELUV& colorOut
 )
```

### Parameters

*colorIn*

Input quantized color.

*colorOut*

Output unquantized color, as defined by the corresponding structure.

### Remarks

Quantization is the process that transforms a continuous value, usually represented as a floating-point value in the **[0..1]** interval, into a discrete one, usually represented as an integer in the **[0..255]** interval.
Dequantization is the reverse process.

# EasyColor::GetDstQuantization

# EasyColor::SetDstQuantization

Quantization mode for output values.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
static Euresys::Open_eVision_2_10::EColorQuantization GetDstQuantization()

static void SetDstQuantization(Euresys::Open_eVision_2_10::EColorQuantization
quantization)
```

### Remarks

These settings remain permanent and influence the relevant quantized and unquantized color conversions (during lookup table initialization or image color transformation).

# EasyColor::Format422To444

Converts a YUV 4:2:2 image to a YUV 4:4:4 image using interpolation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Format422To444(
 EROIBW16* sourceImage,
 EROIC24* destinationImage,
 BOOL yFirst
 )
```

### Parameters

*sourceImage*

Pointer to the input image/ROI, stored using the 16 bits per pixel format.

*destinationImage*

Pointer to the output image/ROI.

*yFirst*

Flag indicating if the format is YUYVYUYV (**TRUE**) or UYVYUYVY (**FALSE**).

### Remarks

In the YUV system, it has been established that sub-sampling the chroma components does not degrade the visual image quality. The 4:4:4 format uses three bytes of information by pixel. The 4:2:2 format is such that only the U and V chroma of the even pixels are kept and they are stored with the even and odd luma, as follows: Thus, only two bytes per pixel are required. $Y_{[even]}$ $U_{[even]}$ $Y_{[odd]}$ $V_{[even]}$

# EasyColor::Format444To422

Converts a YUV 4:4:4 image to a YUV 4:2:2 image using filtering

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Format444To422(
 EROIC24* sourceImage,
 EROIBW16* destinationImage,
 BOOL yFirst
 )
```

### Parameters

*sourceImage*

Pointer to the input image/ROI.

*destinationImage*

Pointer to the output image/ROI, stored using the 16 bits per pixel format.

*yFirst*

Flag indicating if the format is YUYVYUYV (**TRUE**) or UYVYUYVY (**FALSE**).

### Remarks

In the YUV system, it has been established that sub-sampling the chroma components does not degrade the visual image quality. The 4:4:4 format uses three bytes of information by pixel. The 4:2:2 format is such that only the U and V chroma of the even pixels are kept and they are stored with the even and odd luma, as follows: Thus, only two bytes per pixel are required. $Y_{[even]}$ $U_{[even]}$ $Y_{[odd]}$ $V_{[even]}$

# EasyColor::GetComponent

Extracts one color plane, considered as a gray-level image, from a color image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void GetComponent(
 const EROIC24* colorSourceImage,
 EROIBW8* bWDestinationImage,
 OEV_UINT32 component,
 EColorLookup* lookup
 )
void GetComponent(
 EROIC48* colorSourceImage,
 EROIBW16* bWDestinationImage,
 OEV_UINT32 component
 )
```

### Parameters

*colorSourceImage*

Pointer to the input color image/ROI.

*bWDestinationImage*

Pointers to the output gray-level component image/ROI.

*component*

Color component index (**0**, **1**, or **2**).

*lookup*

Pointer to the color lookup table, or **NULL**.

# EasyColor::ImproveClassCenters

Redefines the class centers by computing the average color value of the pixels assigned to each class in the source image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ImproveClassCenters(
 EROIC24* sourceImage,
 EC24Vector* classCenters
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*classCenters*

Pointer to the vector of the class centers.

### Remarks

This implements a step of the K-means method for unsupervised clustering.
Color image segmentation allows you to decompose a color image in different regions by assigning a "class" (integer index) to every pixel. The nearest neighbor method is used, i.e. for each class a representative center is specified, and a given pixel is associated to the class with the closest center.
Color image segmentation can be used in conjunction with EasyObject to perform blob analysis on the segmented regions.

# EasyColor::IshToRgb

Convert a color from any system to RGB.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void IshToRgb(
 EISH colorIn,
 ERGB& colorOut
 )
void IshToRgb(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::LabToRgb

Convert a color from any system to RGB.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void LabToRgb(
 ELAB colorIn,
 ERGB& colorOut
 )
void LabToRgb(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::LabToXyz

Convert a color from one system to another, including the xyz variant (reduced XYZ).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void LabToXyz(
 ELAB colorIn,
 EXYZ& colorOut
 )
void LabToXyz(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::LchToRgb

Convert a color from any system to RGB.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void LchToRgb(
 ELCH colorIn,
 ERGB& colorOut
 )
```

```
void LchToRgb(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::LshToRgb

Convert a color from any system to RGB.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void LshToRgb(
 ELSH colorIn,
 ERGB& colorOut
 )

void LshToRgb(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::LuvToRgb

Convert a color from any system to RGB.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void LuvToRgb(
 ELUV colorIn,
 ERGB& colorOut
 )
void LuvToRgb(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::LuvToXyz

Convert a color from one system to another, including the xyz variant (reduced XYZ).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void LuvToXyz(
 ELUV colorIn,
 EXYZ& colorOut
 )
void LuvToXyz(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::GetNtscGamma

NTSC gamma exponent

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
static float GetNtscGamma()
```

# EasyColor::GetPalGamma

PAL gamma exponent

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
static float GetPalGamma()
```

# EasyColor::PseudoColor

Applies the mapping defined by the pseudo-color lookup table to transform a gray-level image into a color image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void PseudoColor(
 EROIBW8* sourceImage,
 EROIC24* destinationImage,
 EPseudoColorLookup* lookup
 )
```

## Parameters

*sourceImage*

Pointer to the source gray-level image.

*destinationImage*

Pointer to the destination color image.

*lookup*

Pointer to the pseudo-color lookup table.

### Remarks

Pseudo-coloring is a convenient way to display gray-level images with enhanced contrast: a shade of colors is associated to the shade of gray-level values. A simple way to define the shade of colors is to specify a path in color space.
In order to use pseudo-coloring, a special lookup table is used: EPseudoColorLookup. It handles the mapping between the gray-level and color values.

# EasyColor::Quantize

Convert an unquantized color of a given color system to a quantized color.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Quantize(
 ERGB colorIn,
 EC24& colorOut
 )
void Quantize(
 EXYZ colorIn,
 EC24& colorOut
 )
void Quantize(
 EYUV colorIn,
 EC24& colorOut
 )
void Quantize(
 EYIQ colorIn,
 EC24& colorOut
 )
void Quantize(
 ELSH colorIn,
 EC24& colorOut
 )
```

```
void Quantize(
 EVSH colorIn,
 EC24& colorOut
 )

void Quantize(
 EISH colorIn,
 EC24& colorOut
 )

void Quantize(
 EYSH colorIn,
 EC24& colorOut
 )

void Quantize(
 ELAB colorIn,
 EC24& colorOut
 )

void Quantize(
 ELCH colorIn,
 EC24& colorOut
 )

void Quantize(
 ELUV colorIn,
 EC24& colorOut
 )
```

## Parameters

*colorIn*

Input unquantized color, as defined by the corresponding structure.

*colorOut*

Output quantized color.

## Remarks

Quantization is the process that transforms a continuous value, usually represented as a floating-point value in the **[0..1]** interval, into a discrete one, usually represented as an integer in the **[0..255]** interval.
Dequantization is the reverse process.

# EasyColor::RegisterPlanes

Sets a color plane of a color image by using a gray-level image as component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RegisterPlanes(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_INT32 rShiftX,
 OEV_INT32 gShiftX,
 OEV_INT32 bShiftX,
 OEV_INT32 rShiftY,
 OEV_INT32 gShiftY,
 OEV_INT32 bShiftY
 )
```

## Parameters

*sourceImage*

Pointers to the input image/ROI.

*destinationImage*

Pointer to the output image/ROI.

*rShiftX*

Horizontal shifting of the first plane (the red one in case of an RGB image), expressed in pixels.

*gShiftX*

Horizontal shifting of the second plane (the green one in case of an RGB image), expressed in pixels.

*bShiftX*

Horizontal shifting of the third plane (the blue one in case of an RGB image), expressed in pixels.

*rShiftY*

Vertical shifting of the first plane (the red one in case of an RGB image), expressed in pixels.

*gShiftY*

Vertical shifting of the second plane (the green one in case of an RGB image), expressed in pixels.

*bShiftY*

Vertical shifting of the third plane (the blue one in case of an RGB image), expressed in pixels.

### Remarks

A color image can be seen as a set of three color planes, each corresponding to a color component. The color planes are themselves continuous tone images. An EImageC24 contains three EImageBW8.

# EasyColor::GetRgbStandard

# EasyColor::SetRgbStandard

RGB definition to be used when converting between RGB and other color systems.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
static Euresys::Open_eVision_2_10::ERgbStandard GetRgbStandard()
static void SetRgbStandard(Euresys::Open_eVision_2_10::ERgbStandard rgbStandard)
```

### Remarks

Some variant of the color systems can be used. The EasyColor::SrcQuantization and EasyColor::DstQuantization functions are used to activate them.
These settings remain permanent and influence the relevant quantized and unquantized color conversions (during lookup table initialization or image color transformation).

# EasyColor::RgbToIsh

Convert a color from RGB to another system.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void RgbToIsh(
 ERGB colorIn,
 EISH& colorOut
 )

void RgbToIsh(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color, as defined by the corresponding structure.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::RgbToLab

Convert a color from RGB to another system.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void RgbToLab(
 ERGB colorIn,
 ELAB& colorOut
 )

void RgbToLab(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color, as defined by the corresponding structure.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::RgbToLch

Convert a color from RGB to another system.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RgbToLch(
 ERGB colorIn,
 ELCH& colorOut
 )
void RgbToLch(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color, as defined by the corresponding structure.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::RgbToLsh

Convert a color from RGB to another system.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RgbToLsh(
 ERGB colorIn,
 ELSH& colorOut
 )
void RgbToLsh(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color, as defined by the corresponding structure.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::RgbToLuv

Convert a color from RGB to another system.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RgbToLuv(
 ERGB colorIn,
 ELUV& colorOut
 )
void RgbToLuv(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color, as defined by the corresponding structure.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::RgbToReducedXyz

Convert a color from one system to another, including the xyz variant (reduced XYZ).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RgbToReducedXyz(
 ERGB colorIn,
 EXYZ& colorOut
 )
```

```
void RgbToReducedXyz(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::RgbToVsh

Convert a color from RGB to another system.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void RgbToVsh(
 ERGB colorIn,
 EVSH& colorOut
 )

void RgbToVsh(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color, as defined by the corresponding structure.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::RgbToXyz

Convert a color from RGB to another system.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RgbToXyz(
 ERGB colorIn,
 EXYZ& colorOut
 )
void RgbToXyz(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color, as defined by the corresponding structure.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::RgbToYiq

Convert a color from RGB to another system.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RgbToYiq(
 ERGB colorIn,
 EYIQ& colorOut
 )
void RgbToYiq(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color, as defined by the corresponding structure.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::RgbToYsh

Convert a color from RGB to another system.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RgbToYsh(
 ERGB colorIn,
 EYSH& colorOut
 )
```

```
void RgbToYsh(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color, as defined by the corresponding structure.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::RgbToYuv

Convert a color from RGB to another system.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RgbToYuv(
 ERGB colorIn,
 EYUV& colorOut
 )
void RgbToYuv(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color, as defined by the corresponding structure.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::SetComponent

Sets a color plane of a color image by using a gray-level image as component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetComponent(
 EROIBW8* bWSourceImage,
 EROIC24* colorDestinationImage,
 OEV_UINT32 component
 )

void SetComponent(
 EROIBW16* bWSourceImage,
 EROIC48* colorDestinationImage,
 OEV_UINT32 component
 )
```

### Parameters

*bWSourceImage*

Pointers to the input gray level component image/ROI.

*colorDestinationImage*

Pointer to the output color image/ROI.

*component*

Color component index (**0**, **1**, or **2**).

### Remarks

A color image can be seen as a set of three color planes, each corresponding to a color component. The color planes are themselves continuous tone images. An EImageC24 contains three EImageBW8.

# EasyColor::GetSmpteGamma

SMPTE gamma exponent

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
static float GetSmpteGamma()
```

# EasyColor::GetSrcQuantization

# EasyColor::SetSrcQuantization

Quantization mode for input values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
static Euresys::Open_eVision_2_10::EColorQuantization GetSrcQuantization()
static void SetSrcQuantization(Euresys::Open_eVision_2_10::EColorQuantization
quantization)
```

## Remarks

These settings remain permanent and influence the relevant quantized and unquantized color conversions (during lookup table initialization or image color transformation).

# EasyColor::Transform

Applies a color transformation to a specified image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Transform(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 EColorLookup* lookup
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*lookup*

Pointer to the color lookup.

### Remarks

In the first case, the transformation is defined by a color lookup. See Initialization (EColorLookup).
In the two other cases, the user defines a quantized or unquantized color transformation. No intermediate color lookup table is used.
A color image can be seen as a set of three color planes, each corresponding to a color component. The color planes are themselves continuous tone images. An EImageC24 contains three EImageBW8.

# EasyColor::TransformBayer

Converts an image, using the transformation defined by a color lookup.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void TransformBayer(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 EColorLookup* lookup,
 BOOL evenCol,
 BOOL evenRow
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI. This image must be encoded using the Bayer color pattern.

*destinationImage*

Pointer to the destination image/ROI. This image must be encoded using the Bayer color pattern.

*lookup*

Pointer to the color lookup table holding the color adjustment transform. The lookup table must be previously set up by EColorLookup::WhiteBalance method (no other transforms are supported).

*evenCol*

**TRUE** if the leftmost destination image column can't contain blue pixels.

*evenRow*

**TRUE** if the topmost destination image row can't contain red pixels.

### Remarks

By contrast with EasyColor::Transform, the transformation is applied directly to Bayer-encoded data. This allows efficient processing to take place before conversion to the **C24** format.

# EasyColor::VshToRgb

Convert a color from any system to RGB.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void VshToRgb(
 EVSH colorIn,
 ERGB& colorOut
 )

void VshToRgb(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::XyzToLab

Convert a color from one system to another, including the xyz variant (reduced XYZ).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void XyzToLab(
 EXYZ colorIn,
 ELAB& colorOut
 )

void XyzToLab(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::XyzToLuv

Convert a color from one system to another, including the xyz variant (reduced XYZ).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void XyzToLuv(
 EXYZ colorIn,
 ELUV& colorOut
 )
void XyzToLuv(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::XyzToRgb

Convert a color from any system to RGB.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void XyzToRgb(
 EXYZ colorIn,
 ERGB& colorOut
 )
void XyzToRgb(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::YiqToRgb

Convert a color from any system to RGB.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void YiqToRgb(
 EYIQ colorIn,
 ERGB& colorOut
 )
void YiqToRgb(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::YshToRgb

Convert a color from any system to RGB.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void YshToRgb(
 EYSH colorIn,
 ERGB& colorOut
 )
```

```
void YshToRgb(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# EasyColor::YuvToRgb

Convert a color from any system to RGB.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void YuvToRgb(
 EYUV colorIn,
 ERGB& colorOut
 )
void YuvToRgb(
 EC24 colorIn,
 EC24& colorOut
 )
```

### Parameters

*colorIn*

Input color.

*colorOut*

Output color.

### Remarks

These functions transform the color components (of a pixel) expressed in some color system to the corresponding components in another system.

# 4.14. EasyImage Class

This class contains static properties and methods specific to the EasyImage library.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AdaptiveThreshold | Preforms a locally adaptive threshold on the source image. |
| AlphaBlend | Draws an image over an other. |
| AnalyseHistogram | Returns a floating-point statistical parameter extracted from a range of gray levels in an image histogram (most/least frequent value/frequency, min/max value, count, average, standard deviation). |
| AnalyseHistogramBW16 | Returns a floating-point statistical parameter extracted from a range of gray levels in an image histogram (most/least frequent value/frequency, min/max value, count, average, standard deviation). |
| Area | Counts the pixels whose values are above (or on) a threshold. |

| | |
|---|---|
| AreaDoubleThreshold | Counts the pixels whose values are comprised between (or on) two thresholds. |
| ArgumentImage | Prepares a lookup-table image for use for gradient argument computation. |
| AutoThreshold | Returns a suitable threshold value for a gray-level image binarization. |
| BiLevelBlackTopHatBox | Performs a top-hat filtering on a bilevel image (closed image minus source image) on a rectangular kernel. |
| BiLevelBlackTopHatDisk | Performs a top-hat filtering on a bilevel image (closed image minus source image) on a quasi-circular kernel. |
| BiLevelCloseBox | Performs a closing on a bilevel image (dilation followed by erosion) on a rectangular kernel. |
| BiLevelCloseDisk | Performs a closing on a bilevel image (dilation followed by erosion) on a quasi-circular kernel. |
| BiLevelDilateBox | Performs a dilation on a bilevel image (maximum of the pixel values in a defined neighborhood) on a rectangular kernel.<br>For bilevel images, this maximum can either be 0 (if all pixels are black in the given neighborhood), or the maximum possible pixel value. |

| | |
|---|---|
| BiLevelDilateDisk | Performs a dilation on a bilevel image (maximum of the pixel values in a defined neighborhood) on a quasi-circular kernel.<br>For bilevel images, this maximum can either be 0 (if all pixels are black in the given neighborhood), or the maximum possible pixel value. |
| BiLevelErodeBox | Performs an erosion on a bilevel image (minimum of the pixel values in a defined neighborhood) on a rectangular kernel.<br>For bilevel images, this minimum can either be 0, or the maximum possible pixel value (if all pixels are white in the given neighborhood) |
| BiLevelErodeDisk | Performs an erosion on a bilevel image (minimum of the pixel values in a defined neighborhood) on a quasi-circular kernel.<br>For bilevel images, this minimum can either be 0, or the maximum possible pixel value (if all pixels are white in the given neighborhood) |
| BiLevelMedian | Applies a median filter to a bilevel image (median of the gray values in a 3x3 neighborhood). |
| BiLevelMorphoGradientBox | Computes the morphological gradient of a bilevel image using a rectangular kernel. |
| BiLevelMorphoGradientDisk | Computes the morphological gradient of a bilevel image using a quasi-circular kernel. |
| BiLevelOpenBox | Performs an opening on a bilevel image (erosion followed by dilation) on a rectangular kernel. |

| | |
|---|---|
| BiLevelOpenDisk | Performs an opening on a bilevel image (erosion followed by dilation) on a quasi-circular kernel. |
| BiLevelThick | Applies a thickening operation on a bilevel image, using a 3x3 kernel. |
| BiLevelThin | Applies a thinning operation on a bilevel image, using a 3x3 kernel. |
| BiLevelWhiteTopHatBox | Performs a top-hat filtering on a bilevel image (source image minus opened image) on a rectangular kernel. |
| BiLevelWhiteTopHatDisk | Performs a top-hat filtering on a bilevel image (source image minus opened image) on a quasi-circular kernel. |
| BinaryMoments | Computes the zero-th, first or second order moments on the binarized image, i.e. with a unit weight for those pixels with a value above or equal to the threshold, and zero otherwise. |
| BlackTopHatBox | Performs a top-hat filtering on an image (closed image minus source image) on a rectangular kernel. |
| BlackTopHatDisk | Performs a top-hat filtering on an image (closed image minus source image) on a quasi-circular kernel. |

| | |
|---|---|
| CloseBox | Performs a closing on an image (dilation followed by erosion) on a rectangular kernel. |
| CloseDisk | Performs a closing on an image (dilation followed by erosion) on a quasi-circular kernel. |
| Contour | Follows the *contour* of an object. |
| Convert | Transforms the contents of an image to an image of another type. |
| ConvertTo422 | Turns an 8-bit gray-level image into a YUV 4:2:2 encoded color image. |
| ConvolGaussian | Applies Gaussian filtering (binomial weights) in rectangular kernel of odd size. |
| ConvolGradient | Extracts the edges of an image by summing the absolute values of the Gradient X and Gradient Y derivatives and stores the absolute value (magnitude) of the result in the destination image. |
| ConvolGradientX | Derives an image along X using a Gradient kernel and stores the absolute value (magnitude) of the result in the destination image. |

| | |
|---|---|
| ConvolGradientY | Derives an image along Y using a Gradient kernel and stores the absolute value (magnitude) of the result in the destination image. |
| ConvolHighpass1 | Filters an image using a 3x3 high-pass kernel and stores the absolute value (magnitude) of the result in the destination image. |
| ConvolHighpass2 | Filters an image using a 3x3 high-pass kernel and stores the absolute value (magnitude) of the result in the destination image. |
| ConvolKernel | Performs a convolution in image space, i.e. applies a convolution kernel. |
| ConvolLaplacian4 | Takes the second derivative of an image using a 4-neighbor Laplacian kernel and stores the absolute value (magnitude) of the result in the destination image. |
| ConvolLaplacian8 | Takes the second derivative of an image using an 8-neighbor Laplacian kernel and stores the absolute value (magnitude) of the result in the destination image. |
| ConvolLaplacianX | Takes the horizontal second derivative and stores the absolute value (magnitude) of the result in the destination image. |
| ConvolLaplacianY | Takes the vertical second derivative and stores the absolute value (magnitude) of the result in the destination image. |

| | |
|---|---|
| ConvolLowpass1 | Filters an image using a 3x3 low-pass kernel and stores the absolute value (magnitude) of the result in the destination image. |
| ConvolLowpass2 | Filters an image using a 3x3 low-pass kernel and stores the absolute value (magnitude) of the result in the destination image. |
| ConvolLowpass3 | Filters an image using a 3x3 low-pass kernel. |
| ConvolPrewitt | Extracts the edges of an image by summing the absolute values of the Prewitt X and Prewitt Y derivatives and stores the absolute value (magnitude) of the result in the destination image. |
| ConvolPrewittX | Derives an image along X using a Prewitt kernel and stores the absolute value (magnitude) of the result in the destination image. |
| ConvolPrewittY | Derives an image along Y using a Prewitt kernel and stores the absolute value (magnitude) of the result in the destination image. |
| ConvolRoberts | The Roberts edge extraction filter is based on a 2x2 kernel. |
| ConvolSobel | Extracts the edges of an image by summing the absolute values of the Sobel X and Sobel Y derivatives. |
| ConvolSobelX | Derives an image along X using a Sobel kernel and stores the absolute value (magnitude) of the result in the destination image. |

| | |
|---|---|
| ConvolSobelY | Derives an image along Y using a Sobel kernel and stores the absolute value (magnitude) of the result in the destination image. |
| ConvolSymmetricKernel | Performs a convolution in image space, i.e. applies a square symmetric convolution kernel of size 3x3, 5x5 or 7x7. |
| ConvolUniform | Applies strong low-pass filtering to an image by using a uniform rectangular kernel of odd size. |
| Copy | Copies a source image or a constant in a destination image. |
| CumulateHistogram | Cumulates histogram values in another histogram. |
| DilateBox | Performs a dilation on an image (maximum of the gray values in a defined neighborhood) on a rectangular kernel. |
| DilateDisk | Performs a dilation on an image (maximum of the gray values in a defined neighborhood) on a quasi-circular kernel. |
| Distance | Computes the morphological distance function on a binary image (0 for black, non 0 for white). |
| DoubleThreshold | Converts an image by setting all pixels below the low threshold to a low value, all pixels above the high threshold to a high value, and the remaining pixels to an intermediate value. |

| | |
|---|---|
| Equalize | Equalizes an image histogram (the gray levels are re-mapped so that the histogram becomes as close to uniform as possible). |
| ErodeBox | Performs an erosion on an image (minimum of the gray values in a defined neighborhood) on a rectangular kernel. |
| ErodeDisk | Performs an erosion on an image (minimum of the gray values in a defined neighborhood) on a quasi-circular kernel. |
| Focusing | Returns a measure of the focusing of an image by computing the total gradient energy. |
| Gain | Transforms an image, applying a gain and offset to all pixels. |
| GainOffset | Transforms an image, applying a gain and offset to all pixels. |
| GetFrame | Extracts the frame of given parity from an image. |
| GetOverlayColor | Gets/Sets the color of the overlay in the destination image when a **BW8** Image is used as overlay source image in functions. |
| GetProfilePeaks | Detects peaks in a gray-level profile. Maxima as well as minima are considered. |

| | |
|---|---|
| GradientScalar | Computes the (scalar) gradient image derived from a given source image. |
| GravityCenter | Computes the coordinates of the gravity center of an image, i.e. the average coordinates of the pixels above (or on) the threshold. |
| HDRFusion | Fuses two images using HDR principles. |
| Histogram | Computes the histogram of an image (count of each gray-level value). |
| HistogramThreshold | Determines an appropriate threshold level based on the histogram contents, using an automatic threshold mode. |
| HistogramThresholdBW16 | Determines an appropriate threshold level based on the histogram contents, using an automatic threshold mode. |
| HitAndMiss | Apply the morphological hit-and-miss transform to detect a particular pattern of foreground and background pixels in a BW8, BW16 or C24 image/ROI. |
| HorizontalMirror | Mirrors an image horizontally (the columns are swapped). |

| | |
|---|---|
| ImageToLineSegment | Copies the pixel values along a given line segment (arbitrarily oriented) to a vector. |
| ImageToPath | Given a path described by coordinates in a path vector, copies the corresponding pixel values into the same vector. |
| IsodataThreshold | Computes a suitable threshold value for a histogram. |
| IsodataThresholdBW16 | Computes a suitable threshold value for a histogram. |
| LinearTransform | Applies a general affine transformation. |
| LineSegmentToImage | Copies the pixel values from a vector or a constant to the pixels of a given line segment (arbitrarily oriented). |
| LocalAverage | Computes the average in a rectangular window centered on every pixel. |
| LocalDeviation | Computes the standard deviation in a rectangular window centered on every pixel. |
| Lut | Transforms the gray levels of an image, using a lookup table stored in a vector (of unsigned values). |

| | |
|---|---|
| MatchFrames | Determines the optimal shift amplitude by comparing two successive lines of the image. |
| Median | Applies a median filter to an image (median of the gray values in a 3x3 neighborhood). |
| ModulusImage | Prepares a lookup-table image for use for gradient magnitude computation. |
| MorphoGradientBox | Computes the morphological gradient of an image using a rectangular kernel. |
| MorphoGradientDisk | Computes the morphological gradient of an image using a quasi-circular kernel. |
| Normalize | Normalizes an image, i.e. applies a linear transform to the gray levels so that their average and standard deviation are imposed. |
| Offset | Transforms an image, applying a gain and offset to all pixels. |
| OpenBox | Performs an opening on an image (erosion followed by dilation) on a rectangular kernel. |

| | |
|---|---|
| OpenDisk | Performs an opening on an image (erosion followed by dilation) on a quasi-circular kernel. |
| Oper | Applies the desired arithmetic or logic pixel-wise operator between two images or constants. |
| Overlay | Overlays an image on the top of a color image, at a given position. |
| PathToImage | Given a path described by coordinates in a path vector, copies the pixel values from the path vector to the corresponding image pixels. |
| PixelAverage | Computes the average pixel value in a gray-level or color image. |
| PixelCompare | Counts the number of pixels differing between two images. |
| PixelCount | Counts the pixels in the three value classes separated by two thresholds. |
| PixelMax | Computes the maximum gray-level value in an image. |
| PixelMaxBW16 | Computes the maximum gray-level value in an image. |
| PixelMaxBW8 | Computes the maximum gray-level value in an image. |

| | |
|---|---|
| PixelMin | Computes the minimum gray-level value in an image. |
| PixelMinBW16 | Computes the minimum gray-level value in an image. |
| PixelMinBW8 | Computes the minimum gray-level value in an image. |
| PixelStat | Computes the minimum, maximum and average gray-level values in an image. |
| PixelStatBW16 | Computes the minimum, maximum and average gray-level values in an image. |
| PixelStatBW8 | Computes the minimum, maximum and average gray-level values in an image. |
| PixelStdDev | Computes the average gray-level or color value in an image, the standard deviation of the color components, and the correlation between the color components (in the case of color images). |
| PixelVariance | For a gray-level image, computes the mean and variance of the pixel values. |
| ProfileDerivative | Computes the first derivative of a profile extracted from a gray-level image. |
| ProjectOnAColumn | Projects an image horizontally onto a column. |

| | |
|---|---|
| ProjectOnARow | Projects an image vertically onto a row. |
| RealignFrame | Shifts one frame of the image horizontally. |
| RebuildFrame | Rebuilds one frame of the image by interpolation between the lines of the other frame. |
| RecursiveAverage | Applies stronger noise reduction to small variations and conversely. |
| Register | Registers an image by realigning one, two or three pivot points to reference positions. |
| RmsNoise | Computes the root-mean-square amplitude of noise, by comparing a given image to a reference image. |
| ScaleRotate | Re-scales an image by an arbitrary factor and/or rotates it by an arbitrary angle. |
| SetCircleWarp | Prepares suitable warp images for use with function EasyImage::Warp to unwarp a circular ring-wedge shape into a straight rectangle. |
| SetFrame | Replaces the frame of given parity in an image. |
| SetOverlayColor | Gets/Sets the color of the overlay in the destination image when a **BW8** Image is used as overlay source image in functions. |

| | |
|---|---|
| SetRecursiveAverageLUT | Pre-compute the required non-linear transfer function for noise reduction by recursive temporal averaging. |
| SetupEqualize | Prepares a lookup-table for image equalization, using an image histogram. |
| Shrink | Resizes an image to a smaller size. Pre-filtering is applied to avoid aliasing. |
| SignalNoiseRatio | Computes the signal to noise ratio, in dB, by comparing a given image to a reference image. |
| SwapFrames | Interchanges the even and odd rows of an image. |
| Thick | Applies a thickening operation on an image, using a 3x3 kernel. |
| Thin | Applies a thinning operation on an image, using a 3x3 kernel. |
| ThreeLevelsMinResidueThreshold | Computes the two threshold values used to separate the pixels of an image in three classes. |
| Threshold | Binarize an image by setting pixels to two different possible values in a destination image, according to their value in a source image. |

| | |
|---|---|
| TwoLevelsMinResidueThreshold | Computes the threshold value used to separate the pixels of an image in two classes. |
| Uniformize | Shading correction is the process of transforming the gray or color component values of an image, using one or two reference images or vectors. |
| VerticalMirror | Mirrors an image vertically (the rows are swapped). |
| Warp | Transforms an image so that each pixels are moved to the locations specified in the "warp" images used as look-up tables. |
| WeightedMoments | Computes the zero-th, first, second, third or fourth order weighted moments on the gray-level image. The weight of a pixel is its gray-level value. |
| WhiteTopHatBox | Performs a top-hat filtering on an image (source image minus opened image) on a rectangular kernel. |
| WhiteTopHatDisk | Performs a top-hat filtering on an image (source image minus opened image) on a quasi-circular kernel. |

# EasyImage::AdaptiveThreshold

Preforms a locally adaptive threshold on the source image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AdaptiveThreshold(
 const EROIBW8* src,
 EROIBW8* dst,
 Euresys::Open_eVision_2_10::EAdaptiveThresholdMethod method,
 int halfKernelSize,
 int constant
 )
```

## Parameters

*src*

-

*dst*

-

*method*

The thresholding mode, as defined by the enumeration EAdaptiveThresholdMethod.

*halfKernelSize*

Half width of the kernel rounded down

*constant*

Constant offset applied to the threshold value. By default (argument omitted) **0**, i.e. no change.

## Remarks

Kernel size is always odd.

# EasyImage::AlphaBlend

Draws an image over an other.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AlphaBlend(
 const EROIBW8& sourceImage,
 EROIBW8& destinationImage,
 double opacity
 )
```

### Parameters

*sourceImage*

Foreground image.

*destinationImage*

Background image.

*opacity*

Opacity of the foreground image.

# EasyImage::AnalyseHistogram

Returns a floating-point statistical parameter extracted from a range of gray levels in an image histogram (most/least frequent value/frequency, min/max value, count, average, standard deviation).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float AnalyseHistogram(
 EBWHistogramVector* histogram,
 Euresys::Open_eVision_2_10::EHistogramFeature operation,
 OEV_INT32 minimumIndex,
 OEV_INT32 maximumIndex
 )
```

### Parameters

*histogram*

Pointer to the histogram vector.

*operation*

Parameter to be computed, as defined by EHistogramFeature.

*minimumIndex*

Starting index of the gray-level range.

*maximumIndex*

Ending index of the gray-level range.

# EasyImage::AnalyseHistogramBW16

Returns a floating-point statistical parameter extracted from a range of gray levels in an image histogram (most/least frequent value/frequency, min/max value, count, average, standard deviation).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float AnalyseHistogramBW16(
 EBWHistogramVector* histogram,
 Euresys::Open_eVision_2_10::EHistogramFeature operation,
 OEV_INT32 minimumIndex,
 OEV_INT32 maximumIndex
 )
```

## Parameters

*histogram*

Pointer to the histogram vector.

*operation*

Parameter to be computed, as defined by EHistogramFeature.

*minimumIndex*

Starting index of the gray-level range.

*maximumIndex*

Ending index of the gray-level range.

# EasyImage::Area

Counts the pixels whose values are above (or on) a threshold.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Area(
 const EROIBW8* sourceImage,
 EBW8 threshold,
 OEV_INT32& numberOfPixelsAboveThreshold
 )

void Area(
 const EROIBW16* sourceImage,
 EBW16 threshold,
 OEV_INT32& numberOfPixelsAboveThreshold
 )

void Area(
 const EROIBW8* sourceImage,
 const ERegion& region,
 EBW8 threshold,
 OEV_INT32& numberOfPixelsAboveThreshold
 )

void Area(
 const EROIBW16* sourceImage,
 const ERegion& region,
 EBW16 threshold,
 OEV_INT32& numberOfPixelsAboveThreshold
 )

void Area(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 EBW8 threshold,
 OEV_INT32& numberOfPixelsAboveThreshold
 )
```

```
void Area(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 EBW16 threshold,
 OEV_INT32& numberOfPixelsAboveThreshold
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*threshold*

The pixel thresholding value used to count the pixels

*numberOfPixelsAboveThreshold*

Reference to the count of pixels above or equal to the threshold.

*region*

Pointer to a region to apply the function only on a particular region in the image.

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

# EasyImage::AreaDoubleThreshold

Counts the pixels whose values are comprised between (or on) two thresholds.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AreaDoubleThreshold(
 const EROIBW8* sourceImage,
 EBW8 lowThreshold,
 EBW8 highThreshold,
 OEV_INT32& numberOfPixelsBetweenThresholds
 )
```

```
void AreaDoubleThreshold(
 const EROIBW16* sourceImage,
 EBW16 lowThreshold,
 EBW16 highThreshold,
 OEV_INT32& numberOfPixelsBetweenThresholds
 )

void AreaDoubleThreshold(
 const EROIBW8* sourceImage,
 const ERegion& region,
 EBW8 lowThreshold,
 EBW8 highThreshold,
 OEV_INT32& numberOfPixelsBetweenThresholds
 )

void AreaDoubleThreshold(
 const EROIBW16* sourceImage,
 const ERegion& region,
 EBW16 lowThreshold,
 EBW16 highThreshold,
 OEV_INT32& numberOfPixelsBetweenThresholds
 )

void AreaDoubleThreshold(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 EBW8 lowThreshold,
 EBW8 highThreshold,
 OEV_INT32& numberOfPixelsBetweenThresholds
 )

void AreaDoubleThreshold(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 EBW16 lowThreshold,
 EBW16 highThreshold,
 OEV_INT32& numberOfPixelsBetweenThresholds
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*lowThreshold*

Inferior threshold.

*highThreshold*

Superior threshold.

*numberOfPixelsBetweenThresholds*

Reference to the count of pixels that are above or equal to the inferior threshold, and strictly below the superior threshold.

*region*

Pointer to a region to apply the function only on a particular region in the image.

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

# EasyImage::ArgumentImage

Prepares a lookup-table image for use for gradient argument computation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ArgumentImage(
 EImageBW8* destinationImage,
 EBW8 phase,
 float period
 )
void ArgumentImage(
 EImageBW8* destinationImage
 )
void ArgumentImage(
 EImageBW8* destinationImage,
 EBW8 phase
 )
```

## Parameters

*destinationImage*

Pointer to the destination image.

*phase*

Argument value corresponding to the horizontal direction, in 256-th (65,536-th) of the period (by default, **phase = 0**).

*period*

Range of argument values corresponding to the **0..255** (**0..65535**) interval, in the current angle unit (by default, **period = 0**).

### Remarks

The scale and phase of the gradient argument can be adjusted. The argument angles are counted clockwise on a **0..255** scale in the **BW8** context and on a **0..65535** scale in the **BW16** one, corresponding to a specified range (full turn by default, specified period otherwise). The argument phase is counted on a **0..255** scale or on a **0..65535** scale too. Angle values outside the **0..255** (**0..65535**) interval are wrapped. The period length is given in the current angle unit. EasyImage::ArgumentImage sets a lookup-table image for use with function EasyImage::GradientScalar, ready to compute the argument of the gradient in the source image, i.e. its direction. The argument will be returned as a value in range **0..255** suitable for storage in an EImageBW8 or as a value in the range **0..65535** suitable for storage in an EImageBW16. The phase of the argument can be adjusted.

# EasyImage::AutoThreshold

Returns a suitable threshold value for a gray-level image binarization.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EBW8 AutoThreshold(
 const EROIBW8* sourceImage,
 Euresys::Open_eVision_2_10::EThresholdMode thresholdMode,
 float relativeThresholdMode
 )

EBW16 AutoThreshold(
 const EROIBW16* sourceImage,
 Euresys::Open_eVision_2_10::EThresholdMode thresholdMode,
 float relativeThresholdMode
 )

EBW8 AutoThreshold(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 Euresys::Open_eVision_2_10::EThresholdMode thresholdMode,
 float relativeThresholdMode
 )
```

```
EBW16 AutoThreshold(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 Euresys::Open_eVision_2_10::EThresholdMode thresholdMode,
 float relativeThresholdMode
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*thresholdMode*

The thresholding mode, as defined by the enumeration EThresholdMode. To use absolute thresholding, use directly the threshold value instead.

*relativeThresholdMode*

Fraction of the image pixels that will be set below the threshold. Only used when the threshold value is EThresholdMode_Relative (by default, **relativeThresholdMode = 0.5**).

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

### Remarks

Several modes are available: absolute (the threshold value is given readily in the **thresholdMode** parameter), relative (the threshold value is computed to obtain a desired fraction of the image pixels) or automatic (using three different criteria).

It is possible that, in the automatic or relative thresholding modes, the computed threshold exceeds the dynamic range of the return type. In this case, the value is clipped to the maximum value that is representable in the return type.

# EasyImage::BiLevelBlackTopHatBox

Performs a top-hat filtering on a bilevel image (closed image minus source image) on a rectangular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BiLevelBlackTopHatBox(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. Must not be the same as the source image.

*halfOfKernelWidth*

Half of the box width minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

*halfOfKernelHeight*

Half of the box height minus one (by default, same as **halfOfKernelWidth**; **0** is allowed).

### Remarks

This filter enhances the thin black features.

# EasyImage::BiLevelBlackTopHatDisk

Performs a top-hat filtering on a bilevel image (closed image minus source image) on a quasi-circular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BiLevelBlackTopHatDisk(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. Must not be the same as the source image.

*halfOfKernelWidth*

Half width of the kernel minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

### Remarks

This filter enhances the thin black features.

# EasyImage::BiLevelCloseBox

Performs a closing on a bilevel image (dilation followed by erosion) on a rectangular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BiLevelCloseBox(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half of the box width minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

*halfOfKernelHeight*

Half of the box height minus one (by default, same as **halfOfKernelWidth**; **0** is allowed).

# EasyImage::BiLevelCloseDisk

Performs a closing on a bilevel image (dilation followed by erosion) on a quasi-circular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BiLevelCloseDisk(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half width of the kernel minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

# EasyImage::BiLevelDilateBox

Performs a dilation on a bilevel image (maximum of the pixel values in a defined neighborhood) on a rectangular kernel.
For bilevel images, this maximum can either be 0 (if all pixels are black in the given neighborhood), or the maximum possible pixel value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BiLevelDilateBox(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half of the box width minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

*halfOfKernelHeight*

Half of the box height minus one (by default, same as **halfOfKernelWidth**; **0** is allowed).

# EasyImage::BiLevelDilateDisk

Performs a dilation on a bilevel image (maximum of the pixel values in a defined neighborhood) on a quasi-circular kernel.
For bilevel images, this maximum can either be 0 (if all pixels are black in the given neighborhood), or the maximum possible pixel value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BiLevelDilateDisk(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half width of the kernel minus one (by default, **halfOfKernelWidth =1**; **0** is allowed).

# EasyImage::BiLevelErodeBox

Performs an erosion on a bilevel image (minimum of the pixel values in a defined neighborhood) on a rectangular kernel.
For bilevel images, this minimum can either be 0, or the maximum possible pixel value (if all pixels are white in the given neighborhood)

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BiLevelErodeBox(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half of the box width minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

*halfOfKernelHeight*

Half of the box height minus one (by default, same as **halfOfKernelWidth**; **0** is allowed).

# EasyImage::BiLevelErodeDisk

Performs an erosion on a bilevel image (minimum of the pixel values in a defined neighborhood) on a quasi-circular kernel.
For bilevel images, this minimum can either be 0, or the maximum possible pixel value (if all pixels are white in the given neighborhood)

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BiLevelErodeDisk(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half width of the kernel minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

# EasyImage::BiLevelMedian

Applies a median filter to a bilevel image (median of the gray values in a 3x3 neighborhood).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BiLevelMedian(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. Must not be the same as source image.

# EasyImage::BiLevelMorphoGradientBox

Computes the morphological gradient of a bilevel image using a rectangular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BiLevelMorphoGradientBox(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half of the box width minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

*halfOfKernelHeight*

Half of the box height minus one (by default, same as **halfOfKernelWidth**; **0** is allowed).

### Remarks

The morphological gradient is the difference between the dilation and the erosion of the image, using the same structuring element.
The kernel size is a pair of odd numbers; they must be halved before they are passed. For instance, a 3x5 kernel is passed as 1x2.

# EasyImage::BiLevelMorphoGradientDisk

Computes the morphological gradient of a bilevel image using a quasi-circular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BiLevelMorphoGradientDisk(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half width of the kernel minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

### Remarks

The morphological gradient is the difference between the dilation and the erosion of the image, using the same structuring element.

# EasyImage::BiLevelOpenBox

Performs an opening on a bilevel image (erosion followed by dilation) on a rectangular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BiLevelOpenBox(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

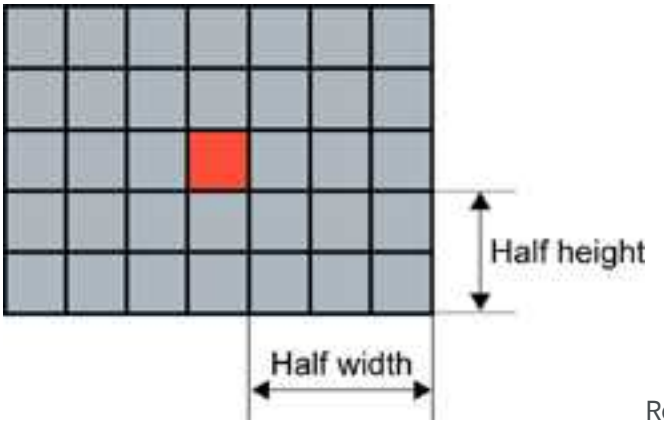
*halfOfKernelWidth*

Half of the box width minus one, as shown on the picture below (by default, **halfOfKernelWidth =1**; **0** is allowed).

*halfOfKernelHeight*

Half of the box height minus one, as shown on the picture below (by default, same as **halfOfKernelWidth**; **0** is allowed).

### Remarks




Rectangular kernel of half width = 3 and half height = 2

# EasyImage::BiLevelOpenDisk

Performs an opening on a bilevel image (erosion followed by dilation) on a quasi-circular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BiLevelOpenDisk(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half width of the kernel minus one, as shown on the picture below (by default, **halfOfKernelWidth = 1**; **0** is allowed).

# EasyImage::BiLevelThick

Applies a thickening operation on a bilevel image, using a 3x3 kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BiLevelThick(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 EKernel* thickeningKernel,
 Euresys::Open_eVision_2_10::EKernelRotation rotationMode,
 OEV_INT32& numberOfIterations
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. Must not be the same as source image.

*thickeningKernel*

Pointer to the thickening kernel.

*rotationMode*

Rotation mode, as defined by EKernelRotation.

*numberOfIterations*

Number of iterations to apply. **0** indicates stop when convergence is reached. Upon return, gives the number of passes actually performed. If the rotation mode is set to either EKernelRotation_Clockwise or EKernelRotation_Anticlockwise, a pass comprises eight kernel rotations.

## Remarks

The thickening kernel coefficients must be **0** (matching black pixel, value 0), **1** (matching non black pixel, value > 0) or **-1** (don't care). When a match is found between the kernel coefficients and the neighborhood of a pixel, the pixel value is set to **255**.

# EasyImage::BiLevelThin

Applies a thinning operation on a bilevel image, using a 3x3 kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BiLevelThin(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 EKernel* thinningKernel,
 Euresys::Open_eVision_2_10::EKernelRotation rotationMode,
 OEV_INT32& numberOfIterations
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. Must not be the same as source image.

*thinningKernel*

Pointer to the thinning kernel.

*rotationMode*

Rotation mode, as defined by EKernelRotation.

*numberOfIterations*

Number of iterations to apply. **0** indicates stop when convergence is reached. Upon return, gives the number of passes actually performed. If the rotation mode is set to either EKernelRotation_Clockwise or EKernelRotation_Anticlockwise, a pass comprises eight kernel rotations.

## Remarks

The thinning kernel coefficients must be **0** (matching black pixel, value 0), **1** (matching non black pixel, value > 0) or **-1** (don't care). When a match is found between the kernel coefficients and the neighborhood of a pixel, the pixel value is set to 0.

# EasyImage::BiLevelWhiteTopHatBox

Performs a top-hat filtering on a bilevel image (source image minus opened image) on a rectangular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BiLevelWhiteTopHatBox(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. Must not be the same as the source image.

*halfOfKernelWidth*

Half of the box width minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

*halfOfKernelHeight*

Half of the box height minus one (by default, same as **halfOfKernelWidth**; **0** is allowed).

### Remarks

This filter enhances the thin white features.

# EasyImage::BiLevelWhiteTopHatDisk

Performs a top-hat filtering on a bilevel image (source image minus opened image) on a quasi-circular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BiLevelWhiteTopHatDisk(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. Must not be the same as the source image.

*halfOfKernelWidth*

Half width of the kernel minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

### Remarks

This filter enhances the thin white features.

# EasyImage::BinaryMoments

Computes the zero-th, first or second order moments on the binarized image, i.e. with a unit weight for those pixels with a value above or equal to the threshold, and zero otherwise.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BinaryMoments(
 const EROIBW8* sourceImage,
 OEV_UINT32 threshold,
 float& M,
 float& Mx,
 float& My
 )
```

```
void BinaryMoments(
 const EROIBW16* sourceImage,
 OEV_UINT32 threshold,
 float& M,
 float& Mx,
 float& My
 )

void BinaryMoments(
 const EROIBW8* sourceImage,
 const ERegion& region,
 OEV_UINT32 threshold,
 float& M,
 float& Mx,
 float& My
 )

void BinaryMoments(
 const EROIBW16* sourceImage,
 const ERegion& region,
 OEV_UINT32 threshold,
 float& M,
 float& Mx,
 float& My
 )

void BinaryMoments(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 OEV_UINT32 threshold,
 float& M,
 float& Mx,
 float& My
 )

void BinaryMoments(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 OEV_UINT32 threshold,
 float& M,
 float& Mx,
 float& My
 )
```

```
void BinaryMoments(
 const EROIBW8* sourceImage,
 OEV_UINT32 threshold,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy
 )

void BinaryMoments(
 const EROIBW16* sourceImage,
 OEV_UINT32 threshold,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy
 )

void BinaryMoments(
 const EROIBW8* sourceImageconst,
 const ERegion& region,
 OEV_UINT32 threshold,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy
 )

void BinaryMoments(
 const EROIBW16* sourceImage,
 const ERegion& region,
 OEV_UINT32 threshold,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy
 )
```

```
void BinaryMoments(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 OEV_UINT32 threshold,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy
 )
void BinaryMoments(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 OEV_UINT32 threshold,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*threshold*

Binarization threshold.

*M*

Reference to the zero-th order moment (area).

*Mx*

Reference to the first-order, uncentered moments (weighted sum of abscissas).

*My*

Reference to the first-order, uncentered moments (weighted sum of ordinates).

*region*

Pointer to a region to apply the function only on a particular region in the image.

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

*Mxx*

Reference to the second-order, uncentered moments (weighted sum of squared abscissas).

*Mxy*

Reference to the second-order, uncentered moments (weighted sum of cross-product of abscissas and ordinates).

*Myy*

Reference to the second-order, uncentered moments (weighted sum of squared ordinates).

*sourceImageconst*

-

# EasyImage::BlackTopHatBox

Performs a top-hat filtering on an image (closed image minus source image) on a rectangular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BlackTopHatBox(
 EROIBW1* sourceImage,
 EROIBW1* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
void BlackTopHatBox(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
void BlackTopHatBox(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
```

```
void BlackTopHatBox(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. Must not be the same as the source image.

*halfOfKernelWidth*

Half of the box width minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

*halfOfKernelHeight*

Half of the box height minus one (by default, same as **halfOfKernelWidth**; **0** is allowed).

### Remarks

This filter enhances the thin black features.

# EasyImage::BlackTopHatDisk

Performs a top-hat filtering on an image (closed image minus source image) on a quasi-circular kernel.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void BlackTopHatDisk(
 EROIBW1* sourceImage,
 EROIBW1* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
```

```
void BlackTopHatDisk(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
void BlackTopHatDisk(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
void BlackTopHatDisk(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. Must not be the same as the source image.

*halfOfKernelWidth*

Half width of the kernel minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

### Remarks

This filter enhances the thin black features.

# EasyImage::CloseBox

Performs a closing on an image (dilation followed by erosion) on a rectangular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void CloseBox(
 EROIBW1* sourceImage,
 EROIBW1* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )

void CloseBox(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )

void CloseBox(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )

void CloseBox(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half of the box width minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

*halfOfKernelHeight*

Half of the box height minus one (by default, same as **halfOfKernelWidth**; **0** is allowed).

# EasyImage::CloseDisk

Performs a closing on an image (dilation followed by erosion) on a quasi-circular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void CloseDisk(
 EROIBW1* sourceImage,
 EROIBW1* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
void CloseDisk(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
void CloseDisk(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
void CloseDisk(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half width of the kernel minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

# EasyImage::Contour

Follows the *contour* of an object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Contour(
 EROIBW8* sourceImage,
 Euresys::Open_eVision_2_10::EContourMode contourMode,
 OEV_INT32 startX,
 OEV_INT32 startY,
 Euresys::Open_eVision_2_10::EContourThreshold thresholdMode,
 OEV_UINT32 threshold,
 Euresys::Open_eVision_2_10::EConnexity connexity,
 EPathVector* path
 )

void Contour(
 EROIBW16* sourceImage,
 Euresys::Open_eVision_2_10::EContourMode contourMode,
 OEV_INT32 startX,
 OEV_INT32 startY,
 Euresys::Open_eVision_2_10::EContourThreshold thresholdMode,
 OEV_UINT32 threshold,
 Euresys::Open_eVision_2_10::EConnexity connexity,
 EPathVector* path
 )

void Contour(
 EROIBW8* sourceImage,
 Euresys::Open_eVision_2_10::EContourMode contourMode,
 OEV_INT32 startX,
 OEV_INT32 startY,
 Euresys::Open_eVision_2_10::EContourThreshold thresholdMode,
 OEV_UINT32 threshold,
 Euresys::Open_eVision_2_10::EConnexity connexity,
 EBW8PathVector* path,
 BOOL freeman
 )
```

```
void Contour(
 EROIBW16* sourceImage,
 Euresys::Open_eVision_2_10::EContourMode contourMode,
 OEV_INT32 startX,
 OEV_INT32 startY,
 Euresys::Open_eVision_2_10::EContourThreshold thresholdMode,
 OEV_UINT32 threshold,
 Euresys::Open_eVision_2_10::EConnexity connexity,
 EBW16PathVector* path,
 BOOL freeman
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*contourMode*

Traversal mode, as defined by EContourMode.

*startX*

Start point abscissa.

*startY*

Start point ordinate.

*thresholdMode*

Thresholding mode as defined by EThresholdMode.

*threshold*

Threshold level.

*connexity*

Contour connexity, as defined by EConnexity.

*path*

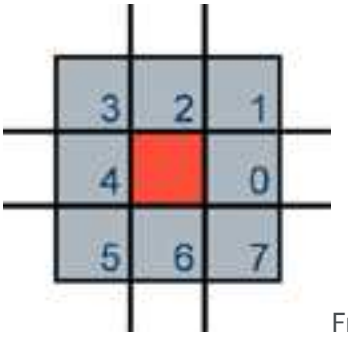
Pointer to the destination vector.

*freeman*

Specifies if Freeman codes are to be retrieved rather than pixel values.

### Remarks

A threshold is applied so that objects become blobs. The contour is a closed or not (see property **Get/SetClosed**) connected path, forming the boundary of the blob.
When destination vector is an EBW8PathVector or a EBW16PathVector, this vector can contain two different information.
If the **bFreeman** argument is **FALSE**, which is the default value, member **m_bw8(16)Pixel** in the vector elements contains the gray-level value of the contour pixels. If it is **TRUE**, the member instead gives the Freeman code leading from a pixel to next. The Freeman codes are numbered from 0 in the horizontal direction and incremented anticlockwise.



Freeman code, leading from a pixel to another adjacent pixel

# EasyImage::Convert

Transforms the contents of an image to an image of another type.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Convert(
 EROIC24* sourceImage,
 EROIBW8* destinationImage
 )
void Convert(
 EROIBW8* sourceImage,
 EROIC24* destinationImage
 )
```

```
void Convert(
 EROIC24* sourceImage,
 EROIC15* destinationImage
 )

void Convert(
 EROIC15* sourceImage,
 EROIC24* destinationImage
 )

void Convert(
 EROIBW8* sourceImage,
 EROIC15* destinationImage
 )

void Convert(
 EROIC15* sourceImage,
 EROIBW8* destinationImage
 )

void Convert(
 EROIC24* sourceImage,
 EROIC16* destinationImage
 )

void Convert(
 EROIC16* sourceImage,
 EROIC24* destinationImage
 )

void Convert(
 EROIBW8* sourceImage,
 EROIC16* destinationImage
 )

void Convert(
 EROIC16* sourceImage,
 EROIBW8* destinationImage
 )

void Convert(
 EROIC24* sourceImage,
 EROIC24A* destinationImage
 )
```

```
void Convert(
 EROIC24A* sourceImage,
 EROIC24* destinationImage
 )

void Convert(
 EROIBW8* sourceImage,
 EROIBW1* destinationImage
 )

void Convert(
 EROIBW16* sourceImage,
 EROIBW1* destinationImage
 )

void Convert(
 EROIBW32* sourceImage,
 EROIBW1* destinationImage
 )

void Convert(
 EROIBW32* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 rightShift
 )

void Convert(
 EROIBW32* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 rightShift
 )

void Convert(
 EROIBW16* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 rightShift
 )

void Convert(
 EROIBW8* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 leftShift
 )
```

```
void Convert(
 EROIBW8* sourceImage,
 EROIBW32* destinationImage,
 OEV_UINT32 leftShift
 )

void Convert(
 EROIBW16* sourceImage,
 EROIBW32* destinationImage,
 OEV_UINT32 leftShift
 )

void Convert(
 EROIC24* sourceImage,
 EROIBW8* sourceImageAlpha,
 EROIC24A* destinationImage
 )

void Convert(
 EROIC24A* sourceImage,
 EROIC24* destinationImage,
 EROIBW8* destinationImageAlpha
 )

void Convert(
 EROIBW1* sourceImage,
 EROIBW8* destinationImage,
 EBW8 highValue
 )

void Convert(
 EROIBW1* sourceImage,
 EROIBW8* destinationImage
 )

void Convert(
 EROIBW1* sourceImage,
 EROIBW16* destinationImage,
 EBW16 highValue
 )

void Convert(
 EROIBW1* sourceImage,
 EROIBW16* destinationImage
 )
```

```
void Convert(
 EROIBW1* sourceImage,
 EROIBW32* destinationImage,
 EBW32 highValue
 )

void Convert(
 EROIBW1* sourceImage,
 EROIBW32* destinationImage
 )

void Convert(
 EROIC48* sourceImage,
 EROIC24* destinationImage,
 OEV_UINT32 rightShift
 )

void Convert(
 EROIC24* sourceImage,
 EROIC48* destinationImage,
 OEV_UINT32 leftShift
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*rightShift*

Right shift amplitude. By default, left justified data is assumed.

*leftShift*

Left shift amplitude. By default, left justified data is assumed.

*sourceImageAlpha*

Pointer to the source alpha component (EImageBW8/EROIBW8).

*destinationImageAlpha*

Pointer to the destination alpha component (EImageBW8/EROIBW8).

*highValue*

In the case of black and white source images/ROIs, indicates to which gray level the value **1** should be mapped. By default, **1** is mapped to the highest allowed value for the destination image/ROI.

### Remarks

Conversion to a black and white image (BW1)
Turns an 8-bit gray-level image into a black and white image.
Turns a 16-bit gray-level image into a black and white image.
Turns a 32-bit gray-level image into a black and white image.
Source pixels whose values is 0 are converted to black. All other source pixel values are converted to white.
Conversion to a 8-bit gray-level image (BW8)
Turns a black and white image into an 8-bit gray-level image.
Turns a 16-bit gray-level image into an 8-bit gray-level image. A right shift can be applied to the 16-bit data to adjust the magnitude, depending on how the 16-bit data is organized. For instance, if the source image holds 10 significant bits right justified, a right shift of 2 is required to drop the 2 low order bits; if the source image holds 12 bits left justified, a right shift of 8 is required and the 4 low order bits will be truncated.
Turns an EC15, EC16 or EC24 color image into an EBW8 gray-level image. The 3 color components are simply averaged, giving the intensity component of the ISH color system.
Conversion to a 16-bit gray-level image (BW16)
Turns a black and white image into a 16-bit gray-level image.
Turns an 8-bit gray-level image into a 16-bit gray-level image. A left shift can be applied to the 8-bit data to adjust the magnitude, depending on how the 16-bit data is organized. For instance, if the destination image holds 10 significant bits right justified, a shift of 2 is required; if the destination image holds 12 bits left justified, a shift of 8 is required.
Conversion to a 32-bit gray-level image (BW32)
Turns a black and white image into a 32-bit gray-level image.
Conversion to color images
Turns an 8-bit gray-level image into a true color equivalent. The color components are all set equal to the corresponding gray-level value.
Converts between standard and Windows' packing RGB color formats. When converting from an EC24 image to a EC15 or EC16 one, only the 5 (or 6) most significant bits of each color component are retained.
Converts between RGB 24-bit color image and RGB32 (also known as RGBA) color image. When converting from EC24 to EC24A, you can choose to provide or not the alpha component. On the other hand, when converting from EC24A to EC24, you can choose to conserve or not the alpha component. The alpha component is retrieved and set using an EImageBW8/EROIBW8.

# EasyImage::ConvertTo422

Turns an 8-bit gray-level image into a YUV 4:2:2 encoded color image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ConvertTo422(
 EROIBW8* sourceImage,
 EROIBW16* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

### Remarks

The Y component is set to the corresponding gray-level values, while the U and V components are set to **128** (achromatic light).

# EasyImage::ConvolGaussian

Applies Gaussian filtering (binomial weights) in rectangular kernel of odd size.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ConvolGaussian(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
```

```
void ConvolGaussian(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )

void ConvolGaussian(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )

void ConvolGaussian(
 const EBW8Vector* sourceImage,
 EBW8Vector* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )

void ConvolGaussian(
 const EBW16Vector* sourceImage,
 EBW16Vector* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half of the box width minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

*halfOfKernelHeight*

Half of the box height minus one (by default, same as **halfOfKernelHeight**; **0** is allowed).

# EasyImage::ConvolGradient

Extracts the edges of an image by summing the absolute values of the Gradient X and Gradient Y derivatives and stores the absolute value (magnitude) of the result in the destination image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ConvolGradient(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )
void ConvolGradient(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )
void ConvolGradient(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

# EasyImage::ConvolGradientX

Derives an image along X using a Gradient kernel and stores the absolute value (magnitude) of the result in the destination image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ConvolGradientX(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )

void ConvolGradientX(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )

void ConvolGradientX(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

### Remarks

Filtering kernel: 0 0 0-1 0 1 0 0 0

# EasyImage::ConvolGradientY

Derives an image along Y using a Gradient kernel and stores the absolute value (magnitude) of the result in the destination image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ConvolGradientY(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )
void ConvolGradientY(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )
void ConvolGradientY(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

### Remarks

Filtering kernel: 0 -1 00  0 00  1 0

# EasyImage::ConvolHighpass1

Filters an image using a 3x3 high-pass kernel and stores the absolute value (magnitude) of the result in the destination image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void ConvolHighpass1(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )
void ConvolHighpass1(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )
void ConvolHighpass1(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

### Remarks

Filtering kernel: 0 -1 0-1 5 -1 0 -1 0

# EasyImage::ConvolHighpass2

Filters an image using a 3x3 high-pass kernel and stores the absolute value (magnitude) of the result in the destination image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void ConvolHighpass2(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )

void ConvolHighpass2(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )

void ConvolHighpass2(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

### Remarks

Filtering kernel: -1 -1 -1-1  9 -1-1 -1 -1

# EasyImage::ConvolKernel

Performs a convolution in image space, i.e. applies a convolution kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void ConvolKernel(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 EKernel* kernel
 )
void ConvolKernel(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 EKernel* kernel
 )
void ConvolKernel(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 EKernel* kernel
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*kernel*

Pointer to the convolution kernel.

### Remarks

Please note that **sourceImage** and **destinationImage** must be different objects.

# EasyImage::ConvolLaplacian4

Takes the second derivative of an image using a 4-neighbor Laplacian kernel and stores the absolute value (magnitude) of the result in the destination image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ConvolLaplacian4(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )

void ConvolLaplacian4(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )

void ConvolLaplacian4(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

### Remarks

Filtering kernel: 0  1 01 -4 10  1 0

# EasyImage::ConvolLaplacian8

Takes the second derivative of an image using an 8-neighbor Laplacian kernel and stores the absolute value (magnitude) of the result in the destination image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ConvolLaplacian8(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )

void ConvolLaplacian8(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )

void ConvolLaplacian8(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

### Remarks

Filtering kernel: 1  1 11 -8 11  1 1

# EasyImage::ConvolLaplacianX

Takes the horizontal second derivative and stores the absolute value (magnitude) of the result in the destination image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void ConvolLaplacianX(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )

void ConvolLaplacianX(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )

void ConvolLaplacianX(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

### Remarks

Filtering kernel: 1 -2 1

# EasyImage::ConvolLaplacianY

Takes the vertical second derivative and stores the absolute value (magnitude) of the result in the destination image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ConvolLaplacianY(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )
```

```
void ConvolLaplacianY(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )

void ConvolLaplacianY(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

### Remarks

Filtering kernel: 1-2 1

# EasyImage::ConvolLowpass1

Filters an image using a 3x3 low-pass kernel and stores the absolute value (magnitude) of the result in the destination image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ConvolLowpass1(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )
```

```
void ConvolLowpass1(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )

void ConvolLowpass1(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

### Remarks

Filtering kernel: 1 1 11 1 11 1 1

# EasyImage::ConvolLowpass2

Filters an image using a 3x3 low-pass kernel and stores the absolute value (magnitude) of the result in the destination image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ConvolLowpass2(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )
```

```
void ConvolLowpass2(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )

void ConvolLowpass2(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

### Remarks

Filtering kernel: 1 1 11 0 11 1 1

# EasyImage::ConvolLowpass3

Filters an image using a 3x3 low-pass kernel.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ConvolLowpass3(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )

void ConvolLowpass3(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )
```

```
void ConvolLowpass3(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

### Remarks

Filtering kernel: 1 2 12 4 21 2 1

# EasyImage::ConvolPrewitt

Extracts the edges of an image by summing the absolute values of the Prewitt X and Prewitt Y derivatives and stores the absolute value (magnitude) of the result in the destination image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ConvolPrewitt(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )

void ConvolPrewitt(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )

void ConvolPrewitt(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

# EasyImage::ConvolPrewittX

Derives an image along X using a Prewitt kernel and stores the absolute value (magnitude) of the result in the destination image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ConvolPrewittX(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )
void ConvolPrewittX(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )
void ConvolPrewittX(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

### Remarks

Filtering kernel: -1 0 1-1 0 1-1 0 1

# EasyImage::ConvolPrewittY

Derives an image along Y using a Prewitt kernel and stores the absolute value (magnitude) of the result in the destination image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ConvolPrewittY(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )
void ConvolPrewittY(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )
void ConvolPrewittY(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

### Remarks

Filtering kernel: -1 -1 -1 0 0 0 1 1 1

# EasyImage::ConvolRoberts

The Roberts edge extraction filter is based on a 2x2 kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ConvolRoberts(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )
void ConvolRoberts(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )
void ConvolRoberts(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

### Remarks

It computes the sum of absolute differences of the pixel values in the diagonal directions.

# EasyImage::ConvolSobel

Extracts the edges of an image by summing the absolute values of the Sobel X and Sobel Y derivatives.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void ConvolSobel(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )
void ConvolSobel(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )
void ConvolSobel(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

# EasyImage::ConvolSobelX

Derives an image along X using a Sobel kernel and stores the absolute value (magnitude) of the result in the destination image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ConvolSobelX(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )
```

```
void ConvolSobelX(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )

void ConvolSobelX(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

### Remarks

Filtering kernel: -1 0 1-2 0 2-1 0 1

# EasyImage::ConvolSobelY

Derives an image along Y using a Sobel kernel and stores the absolute value (magnitude) of the result in the destination image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ConvolSobelY(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )
```

```
void ConvolSobelY(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )

void ConvolSobelY(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

### Remarks

Filtering kernel: -1 -2 -1 0 0 0 1 2 1

# EasyImage::ConvolSymmetricKernel

Performs a convolution in image space, i.e. applies a square symmetric convolution kernel of size 3x3, 5x5 or 7x7.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ConvolSymmetricKernel(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 EKernel* kernel
 )
```

```
void ConvolSymmetricKernel(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 EKernel* kernel
 )
void ConvolSymmetricKernel(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 EKernel* kernel
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*kernel*

Pointer to the convolution kernel.

### Remarks

This function is a synonym for EasyImage::ConvolKernel.

# EasyImage::ConvolUniform

Applies strong low-pass filtering to an image by using a uniform rectangular kernel of odd size.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void ConvolUniform(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )

void ConvolUniform(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )

void ConvolUniform(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )

void ConvolUniform(
 EBW8Vector* sourceVector,
 EBW8Vector* destinationVector,
 OEV_UINT32 halfOfKernelWidth
 )

void ConvolUniform(
 EBW16Vector* sourceVector,
 EBW16Vector* destinationVector,
 OEV_UINT32 halfOfKernelWidth
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image) and the default value for **un32HalfWidth** (**1**) has to be used.

*halfOfKernelWidth*

Half of the box width minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

*halfOfKernelHeight*

Half of the box height minus one (by default, same as **halfOfKernelWidth**; **0** is allowed).

*sourceVector*

Pointer to the source vector.

*destinationVector*

Pointer to the destination vector. If **NULL** (default), this operation is destructive (i.e. applied to the source vector) and the default value for **un32HalfWidth** (**1**) has to be used.

### Remarks

This filter replaces every pixel values by the arithmetic mean of the neighboring values in a rectangular window. To handle pixels along edges, the source pixels are replicated outwards as many times as required.
A very nice feature of this function is that its running time does not depend on the kernel size!

# EasyImage::Copy

Copies a source image or a constant in a destination image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Copy(
 const EROIBW1* sourceImage,
 EROIBW1* destinationImage
 )

void Copy(
 const EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )

void Copy(
 const EROIBW32* sourceImage,
 EROIBW32* destinationImage
 )

void Copy(
 const EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

```
void Copy(
 const EROIC15* sourceImage,
 EROIC15* destinationImage
 )

void Copy(
 const EROIC16* sourceImage,
 EROIC16* destinationImage
 )

void Copy(
 const EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )

void Copy(
 const EROIC48* sourceImage,
 EROIC48* destinationImage
 )

void Copy(
 EBW1 constant,
 EROIBW1* destinationImage
 )

void Copy(
 EBW16 constant,
 EROIBW16* destinationImage
 )

void Copy(
 EBW32 constant,
 EROIBW32* destinationImage
 )

void Copy(
 EC24 constant,
 EROIC24* destinationImage
 )

void Copy(
 EC15 constant,
 EROIC15* destinationImage
 )
```

```
void Copy(
 EC16 constant,
 EROIC16* destinationImage
 )

void Copy(
 EBW8 constant,
 EROIBW8* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*constant*

Gray-level or color constant.

# EasyImage::CumulateHistogram

Cumulates histogram values in another histogram.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void CumulateHistogram(
 EBWHistogramVector* sourceVector,
 EBWHistogramVector* destinationVector
 )
```

### Parameters

*sourceVector*

Pointer to the source vector.

*destinationVector*

Pointer to the destination vector.

### Remarks

Calling this function after EasyImage::Histogram allows you to compute the cumulative histogram of an image, i.e. the count of pixels below a given threshold value (instead of the count of pixels with a given gray value, as computed by EasyImage::Histogram).

# EasyImage::DilateBox

Performs a dilation on an image (maximum of the gray values in a defined neighborhood) on a rectangular kernel.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DilateBox(
 EROIBW1* sourceImage,
 EROIBW1* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )

void DilateBox(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )

void DilateBox(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )

void DilateBox(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half of the box width minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

*halfOfKernelHeight*

Half of the box height minus one (by default, same as **halfOfKernelWidth**; **0** is allowed).

# EasyImage::DilateDisk

Performs a dilation on an image (maximum of the gray values in a defined neighborhood) on a quasi-circular kernel.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DilateDisk(
 EROIBW1* sourceImage,
 EROIBW1* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )

void DilateDisk(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )

void DilateDisk(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
```

```
void DilateDisk(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half width of the kernel minus one (by default, **halfOfKernelWidth =1**; **0** is allowed).

# EasyImage::Distance

Computes the morphological distance function on a binary image (0 for black, non 0 for white).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Distance(
 EROIBW8* sourceImage,
 EImageBW16* destinationImage,
 OEV_INT32 valueOutOfImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. Must not be the same as source image.

*valueOutOfImage*

Out-of-bounds image value. By default, this value is **0**.

### Remarks

So, each pixel of the destination image will contain, at the end of the processing, the morphological distance of the corresponding pixel in the source image. The distance function at a given pixel tells how many erosion passes are required to set it to black.

# EasyImage::DoubleThreshold

Converts an image by setting all pixels below the low threshold to a low value, all pixels above the high threshold to a high value, and the remaining pixels to an intermediate value.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DoubleThreshold(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 lowThreshold,
 OEV_UINT32 highThreshold,
 OEV_UINT8 lowValue,
 OEV_UINT8 middleValue,
 OEV_UINT8 highValue
 )

void DoubleThreshold(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 lowThreshold,
 OEV_UINT32 highThreshold,
 EBW16 lowValue,
 EBW16 middleValue,
 EBW16 highValue
 )

void DoubleThreshold(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 lowThreshold,
 OEV_UINT32 highThreshold
 )
```

```
void DoubleThreshold(
 const EROIBW8* sourceImage,
 const ERegion& region,
 EROIBW8* destinationImage,
 OEV_UINT32 lowThreshold,
 OEV_UINT32 highThreshold,
 OEV_UINT8 lowValue,
 OEV_UINT8 middleValue,
 OEV_UINT8 highValue
 )

void DoubleThreshold(
 const EROIBW16* sourceImage,
 const ERegion& region,
 EROIBW16* destinationImage,
 OEV_UINT32 lowThreshold,
 OEV_UINT32 highThreshold,
 EBW16 lowValue,
 EBW16 middleValue,
 EBW16 highValue
 )

void DoubleThreshold(
 const EROIBW16* sourceImage,
 const ERegion& region,
 EROIBW16* destinationImage,
 OEV_UINT32 lowThreshold,
 OEV_UINT32 highThreshold
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*lowThreshold*

Low threshold value.

*highThreshold*

High threshold value.

*lowValue*

Value for pixels strictly below the low threshold.

*middleValue*

Value for pixels that are above or equal to the low threshold, and below the high threshold.

*highValue*

Value for pixels above or equal to the high threshold.

*region*

Pointer to a region to apply the function only on a particular region in the image.

# EasyImage::Equalize

Equalizes an image histogram (the gray levels are re-mapped so that the histogram becomes as close to uniform as possible).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Equalize(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )

void Equalize(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

## Remarks

This strongly enhances the contrast in dark areas.

# EasyImage::ErodeBox

Performs an erosion on an image (minimum of the gray values in a defined neighborhood) on a rectangular kernel.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ErodeBox(
 EROIBW1* sourceImage,
 EROIBW1* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )

void ErodeBox(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )

void ErodeBox(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )

void ErodeBox(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half of the box width minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

*halfOfKernelHeight*

Half of the box height minus one (by default, same as **halfOfKernelWidth**; **0** is allowed).

# EasyImage::ErodeDisk

Performs an erosion on an image (minimum of the gray values in a defined neighborhood) on a quasi-circular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ErodeDisk(
 EROIBW1* sourceImage,
 EROIBW1* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )

void ErodeDisk(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )

void ErodeDisk(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )

void ErodeDisk(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half width of the kernel minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

# EasyImage::Focusing

Returns a measure of the focusing of an image by computing the total gradient energy.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float Focusing(
 EROIBW8* image
 )

float Focusing(
 EROIBW16* image
 )

float Focusing(
 EROIC24* image
 )
```

## Parameters

*image*

Pointer to the source image/ROI.

## Remarks

When this quantity is maximum for a given image, sharp focusing is achieved.
For more information, please refer to the section **Using Open eVision -> EasyImage - Computing Image Statistics -> Image Focus**in the documentation.

# EasyImage::Gain

Transforms an image, applying a gain and offset to all pixels.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Gain(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 EColor Gain
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*Gain*

Constant gain. By default (argument omitted) **1**, i.e. no change.

### Remarks

The gain should remain close to **1**, and allows contrast adjustment of the image.

The offset can be positive or negative, and allows to adjust the image intensity. The resulting values are always saturated to range **[0..255]**.

For color images, the separate gain and offset values are specified as triple of values stored in a EColor. The default values leave the image unchanged.

Internally, the computations are achieved through fixed-point arithmetic with 5 bits of precision for the fractional part. This can result in loss of precision with small gains.

# EasyImage::GainOffset

Transforms an image, applying a gain and offset to all pixels.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GainOffset(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 float gain,
 float offset
 )
void GainOffset(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 float gain,
 float offset
 )
void GainOffset(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 EColor gain,
 EColor offset
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*gain*

Constant gain. By default (argument omitted) **1**, i.e. no change.

*offset*

Constant offset. By default (argument omitted) **0**, i.e. no change.

### Remarks

The gain should remain close to **1**, and allows contrast adjustment of the image.

The offset can be positive or negative, and allows to adjust the image intensity. The resulting values are always saturated to range **[0..255]**.

For color images, the separate gain and offset values are specified as triple of values stored in a EColor. The default values leave the image unchanged.

Internally, the computations are achieved through fixed-point arithmetic with 5 bits of precision for the fractional part. This can result in loss of precision with small gains.

# EasyImage::GetFrame

Extracts the frame of given parity from an image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetFrame(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 BOOL odd
 )
void GetFrame(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 BOOL odd
 )
void GetFrame(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 BOOL odd
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. Must not be the same as the source image.

*odd*

Specifies which frame is extracted (the frame made up of all lines of the same parity as **odd**).

### Remarks

The size of the destination image is determined as follows:
**DstImage_Width = SrcImage_Width**
**DstImage_Height = (SrcImage_Height + 1 - odd ) / 2**

# EasyImage::GetProfilePeaks

Detects peaks in a gray-level profile. Maxima as well as minima are considered.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetProfilePeaks(
 EBW8Vector* profile,
 EPeakVector* peaks,
 OEV_UINT32 lowThreshold,
 OEV_UINT32 highThreshold,
 OEV_UINT32 minimumAmplitude,
 OEV_UINT32 minimumArea
 )
void GetProfilePeaks(
 EBW16Vector* profile,
 EPeakVector* peaks,
 OEV_UINT32 lowThreshold,
 OEV_UINT32 highThreshold,
 OEV_UINT32 minimumAmplitude,
 OEV_UINT32 minimumArea
 )
```

### Parameters

*profile*

Pointer to the source vector.

*peaks*

Pointer to the destination vector.

*lowThreshold*

Threshold used for the minimum peaks.

*highThreshold*

Threshold used for the maximum peaks.

*minimumAmplitude*

Minimum amplitude required for a peak to be kept (may be **0**).

*minimumArea*

Minimum area required for a peak to be kept (may be **0**).

### Remarks

To eliminate false peaks due to noise, two selection criteria are used.
A peak is the portion of the signal that is above [below] a given threshold. The peak amplitude is defined to be the difference between the threshold value and the maximum [minimum] signal value. The peak area is defined to be the surface comprised between the signal curve and the horizontal line at the given threshold.
The result is stored in a peaks vector.

# EasyImage::GradientScalar

Computes the (scalar) gradient image derived from a given source image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GradientScalar(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 EROIBW8* lookupTable
 )

void GradientScalar(
 EROIBW32* sourceImage,
 EROIBW8* destinationImage,
 EROIBW8* lookupTable
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*lookupTable*

Pointer to the image/ROI used as a preset lookup-table. This lookup table can be generated by one of EasyImage::ArgumentImage or EasyImage::ModulusImage, or be user-defined.

### Remarks

The scalar value derived from the gradient depends on the preset lookup-table image.
The gradient of a gray-scale image corresponds to a vector, the components of which are the partial derivatives of the gray-level signal in the horizontal and vertical direction. A vector can be characterized by a direction and a length, corresponding to the gradient orientation, here called *argument*, and the gradient magnitude, here called *magnitude*. Function EasyImage::GradientScalar generates a gradient direction or gradient magnitude map (gray-level image) from a given gray-level image. For efficiency, a pre-computed lookup-table is used to define the desired transformation. This lookup-table is stored as a standard EImageBW8/EImageBW16. Use one of EasyImage::ArgumentImage or EasyImage::ModulusImage once before calling EasyImage::GradientScalar.

# EasyImage::GravityCenter

Computes the coordinates of the gravity center of an image, i.e. the average coordinates of the pixels above (or on) the threshold.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void GravityCenter(
 const EROIBW8* sourceImage,
 OEV_UINT32 threshold,
 float& gravityX,
 float& gravityY
 )

void GravityCenter(
 const EROIBW16* sourceImage,
 OEV_UINT32 threshold,
 float& gravityX,
 float& gravityY
 )
```

```
void GravityCenter(
 const EROIBW8* sourceImage,
 const ERegion& region,
 OEV_UINT32 threshold,
 float& gravityX,
 float& gravityY
 )
void GravityCenter(
 const EROIBW16* sourceImage,
 const ERegion& region,
 OEV_UINT32 threshold,
 float& gravityX,
 float& gravityY
 )
void GravityCenter(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 OEV_UINT32 threshold,
 float& gravityX,
 float& gravityY
 )
void GravityCenter(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 OEV_UINT32 threshold,
 float& gravityX,
 float& gravityY
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*threshold*

Threshold.

*gravityX*

Reference to the gravity center abscissa.

*gravityY*

Reference to the gravity center ordinate.

*region*

Pointer to a region to apply the function only on a particular region in the image.

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

# EasyImage::HDRFusion

Fuses two images using HDR principles.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void HDRFusion(
 const EROIBW8* darkSrc,
 const EROIBW8* lightSrc,
 int shutterSpeedFactor,
 EROIBW16* dst
 )

void HDRFusion(
 const EROIBW16* darkSrc,
 const EROIBW16* lightSrc,
 int shutterSpeedFactor,
 EROIBW16* dst
 )

void HDRFusion(
 const EROIBW16* darkSrc,
 const EROIBW16* lightSrc,
 int shutterSpeedFactor,
 EROIBW32* dst
 )

void HDRFusion(
 const EROIC24* darkSrc,
 const EROIC24* lightSrc,
 int shutterSpeedFactor,
 EROIC48* dst
 )
```

```
void HDRFusion(
 const EROIC48* darkSrc,
 const EROIC48* lightSrc,
 int shutterSpeedFactor,
 EROIC48* dst
 )
```

### Parameters

*darkSrc*

Dark input image (high shutter speed).

*lightSrc*

Light input image (low shutter speed).

*shutterSpeedFactor*

Shutter speed factor between light and dark image.

*dst*

Destination image.

# EasyImage::Histogram

Computes the histogram of an image (count of each gray-level value).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Histogram(
 const EROIBW8* sourceImage,
 EBWHistogramVector* histogram
 )

void Histogram(
 const EROIBW8* sourceImage,
 const ERegion& region,
 EBWHistogramVector* histogram
 )
```

```
void Histogram(
 const EROIBW16* sourceImage,
 EBWHistogramVector* histogram,
 OEV_UINT32 mostSignificantBit,
 OEV_UINT32 numberOfSignificantBits,
 BOOL saturate
 )

void Histogram(
 const EROIBW16* sourceImage,
 const ERegion& region,
 EBWHistogramVector* histogram,
 OEV_UINT32 mostSignificantBit,
 OEV_UINT32 numberOfSignificantBits,
 BOOL saturate
 )

void Histogram(
 const EROIBW32* sourceImage,
 EBWHistogramVector* histogram,
 OEV_UINT32 mostSignificantBit,
 OEV_UINT32 numberOfSignificantBits,
 BOOL saturate
 )

void Histogram(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 EBWHistogramVector* histogram
 )

void Histogram(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 EBWHistogramVector* histogram,
 OEV_UINT32 mostSignificantBit,
 OEV_UINT32 numberOfSignificantBits,
 BOOL saturate
 )
```

```
void Histogram(
 const EROIBW32* sourceImage,
 const EROIBW8* mask,
 EBWHistogramVector* histogram,
 OEV_UINT32 mostSignificantBit,
 OEV_UINT32 numberOfSignificantBits,
 BOOL saturate
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*histogram*

Pointer to the destination vector.

*region*

Pointer to a region to apply the function only on a particular region in the image.

*mostSignificantBit*

Index of the most significant bit of the pixels (**0** has weight 1).

*numberOfSignificantBits*

Number of significant bits; the histogram will possess $2^{numberOfSignificantBits}$ entries.

*saturate*

Boolean indicating if values larger than $2^{mostSignificantBit}$-1 are saturated (default **TRUE**) or not (**FALSE**).

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

# EasyImage::HistogramThreshold

Determines an appropriate threshold level based on the histogram contents, using an automatic threshold mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void HistogramThreshold(
 EBWHistogramVector* histogram,
 OEV_UINT32& threshold,
 float& averageOfPixelsBelowThreshold,
 float& averageOfPixelsAboveThreshold,
 float relativeThreshold,
 OEV_UINT32 from,
 OEV_UINT32 to
 )
```

### Parameters

*histogram*

Pointer to a vector conaining an image histogram.

*threshold*

reference to the threshold value. Before calling the function, must be set to the appropriate thresholding mode, as defined by EThresholdMode.

*averageOfPixelsBelowThreshold*

Average gray level of the dark pixels (below threshold).

*averageOfPixelsAboveThreshold*

Average gray level of the light pixels (above threshold).

*relativeThreshold*

Relative threshold value, relevant only in the EThresholdMode_Relative mode.

*from*

Lower bound of the analyzed gray-level range.

*to*

Upper bound of the analyzed gray-level range.

### Remarks

Additionally, returns the average gray levels in the regions below and above the threshold. The threshold level can be computed by analyzing a range of gray levels in the histogram.

# EasyImage::HistogramThresholdBW16

Determines an appropriate threshold level based on the histogram contents, using an automatic threshold mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void HistogramThresholdBW16(
 EBWHistogramVector* histogram,
 OEV_UINT32& threshold,
 float& averageOfPixelsBelowThreshold,
 float& averageOfPixelsAboveThreshold,
 float relativeThreshold,
 OEV_UINT32 from,
 OEV_UINT32 to
 )
```

### Parameters

*histogram*

Pointer to a vector conaining an image histogram.

*threshold*

reference to the threshold value. Before calling the function, must be set to the appropriate thresholding mode, as defined by EThresholdMode.

*averageOfPixelsBelowThreshold*

Average gray level of the dark pixels (below threshold).

*averageOfPixelsAboveThreshold*

Average gray level of the light pixels (above threshold).

*relativeThreshold*

Relative threshold value, relevant only in the EThresholdMode_Relative mode.

*from*

Lower bound of the analyzed gray-level range.

*to*

Upper bound of the analyzed gray-level range.

### Remarks

Additionally, returns the average gray levels in the regions below and above the threshold. The threshold level can be computed by analyzing a range of gray levels in the histogram.

# EasyImage::HitAndMiss

Apply the morphological hit-and-miss transform to detect a particular pattern of foreground and background pixels in a BW8, BW16 or C24 image/ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void HitAndMiss(
 const EROIBW8* source,
 EROIBW8* destination,
 const EHitAndMissKernel& kernel
 )
void HitAndMiss(
 const EROIBW16* source,
 EROIBW16* destination,
 const EHitAndMissKernel& kernel
 )
void HitAndMiss(
 const EROIC24* source,
 EROIC24* destination,
 const EHitAndMissKernel& kernel
 )
```

### Parameters

*source*

The source image/ROI.

*destination*

The destination image/ROI.

*kernel*

The hit-and-miss kernel.

# EasyImage::HorizontalMirror

Mirrors an image horizontally (the columns are swapped).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void HorizontalMirror(
 EROIBW8* sourceImage
 )

void HorizontalMirror(
 EROIC24* sourceImage
 )

void HorizontalMirror(
 EROIBW16* sourceImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

# EasyImage::ImageToLineSegment

Copies the pixel values along a given line segment (arbitrarily oriented) to a vector.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ImageToLineSegment(
 const EROIBW8* sourceImage,
 EBW8Vector* path,
 OEV_INT32 X0,
 OEV_INT32 Y0,
 OEV_INT32 X1,
 OEV_INT32 Y1
 )
```

```
void ImageToLineSegment(
 const EROIBW16* sourceImage,
 EBW16Vector* path,
 OEV_INT32 X0,
 OEV_INT32 Y0,
 OEV_INT32 X1,
 OEV_INT32 Y1
 )

void ImageToLineSegment(
 const EROIC24* sourceImage,
 EC24Vector* path,
 OEV_INT32 X0,
 OEV_INT32 Y0,
 OEV_INT32 X1,
 OEV_INT32 Y1
 )

void ImageToLineSegment(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 EBW8 outOfMaskValue,
 EBW8Vector* path,
 OEV_INT32 X0,
 OEV_INT32 Y0,
 OEV_INT32 X1,
 OEV_INT32 Y1
 )

void ImageToLineSegment(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 EBW16 outOfMaskValue,
 EBW16Vector* path,
 OEV_INT32 X0,
 OEV_INT32 Y0,
 OEV_INT32 X1,
 OEV_INT32 Y1
 )
```

```
void ImageToLineSegment(
 const EROIC24* sourceImage,
 const EROIBW8* mask,
 EC24 outOfMaskValue,
 EC24Vector* path,
 OEV_INT32 X0,
 OEV_INT32 Y0,
 OEV_INT32 X1,
 OEV_INT32 Y1
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*path*

Pointer to the destination vector.

*X0*

Coordinates of the starting point of the segment.

*Y0*

Coordinates of the starting point of the segment.

*X1*

Coordinates of the ending point of the segment.

*Y1*

Coordinates of the ending point of the segment.

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

*outOfMaskValue*

The value to be given to the pixels that lie out of the mask.

### Remarks

The line segment must be wholly contained within the image. The vector length is adjusted automatically.

# EasyImage::ImageToPath

Given a path described by coordinates in a path vector, copies the corresponding pixel values into the same vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ImageToPath(
 const EROIBW8* sourceImage,
 EBW8PathVector* path
 )
void ImageToPath(
 const EROIBW16* sourceImage,
 EBW16PathVector* path
 )
void ImageToPath(
 const EROIC24* sourceImage,
 EC24PathVector* path
 )
void ImageToPath(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 EBW8 outOfMaskValue,
 EBW8PathVector* path
 )
void ImageToPath(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 EBW16 outOfMaskValue,
 EBW16PathVector* path
 )
void ImageToPath(
 const EROIC24* sourceImage,
 const EROIBW8* mask,
 EC24 outOfMaskValue,
 EC24PathVector* path
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*path*

Pointer to the destination vector.

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

*outOfMaskValue*

The value to be given to the pixels that lie out of the mask.

# EasyImage::IsodataThreshold

Computes a suitable threshold value for a histogram.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW8 IsodataThreshold(
 EBWHistogramVector* histogram,
 OEV_UINT32 from,
 OEV_UINT32 to
 )
```

## Parameters

*histogram*

Pointer to a vector containing an image histogram.

*from*

Lower bound of the useful gray-level interval (by default, **0**).

*to*

Upper bound of the useful gray-level interval (by default, **255**).

## Remarks

The "isodata" rule is used: if one computes the average gray level of pixels below the threshold and the average gray level of pixels above the threshold, the threshold lies exactly halfway between them. Optionally, the threshold selection can be restricted to a range of gray-level values.

It is possible that, in the automatic or relative thresholding modes, the computed threshold exceeds the dynamic range of the return type. In this case, the value is clipped to the maximum value that is representable in the return type.

# EasyImage::IsodataThresholdBW16

Computes a suitable threshold value for a histogram.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW16 IsodataThresholdBW16(
 EBWHistogramVector* histogram,
 OEV_UINT32 from,
 OEV_UINT32 to
 )
```

### Parameters

*histogram*

Pointer to a vector containing an image histogram.

*from*

Lower bound of the useful gray-level interval (by default, **0**).

*to*

Upper bound of the useful gray-level interval (by default, **65535**).

### Remarks

The "isodata" rule is used: if one computes the average gray level of pixels below the threshold and the average gray level of pixels above the threshold, the threshold lies exactly halfway between them. Optionally, the threshold selection can be restricted to a range of gray-level values.
Returns the threshold.

# EasyImage::LinearTransform

Applies a general affine transformation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void LinearTransform(
 EROIBW8* sourceImage,
 float Axx,
 float Axy,
 float Ax,
 float Ayx,
 float Ayy,
 float Ay,
 EROIBW8* destinationImage,
 OEV_INT32 interpolationBits
 )

void LinearTransform(
 EROIBW16* sourceImage,
 float Axx,
 float Axy,
 float Ax,
 float Ayx,
 float Ayy,
 float Ay,
 EROIBW16* destinationImage,
 OEV_INT32 interpolationBits
 )

void LinearTransform(
 EROIC24* sourceImage,
 float Axx,
 float Axy,
 float Ax,
 float Ayx,
 float Ayy,
 float Ay,
 EROIC24* destinationImage,
 OEV_INT32 interpolationBits
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*Axx*

See formula below.

*Axy*

See formula below.

*Ax*

See formula below.

*Ayx*

See formula below.

*Ayy*

See formula below.

*Ay*

See formula below.

*destinationImage*

Pointer to the destination image/ROI.

*interpolationBits*

Number of bits of accuracy for interpolation. Allowed values are **0** (no interpolation, nearest neighbor), **4 (linear interpolation)** or **8 (cubic interpolation)**.

### Remarks

An affine transformation is an important class of linear 2D geometric transformations which maps variables (e.g. pixel intensity values located at position $(X_{Src}, Y_{Src})$ in an input image) into new variables (e.g. $(X_{Dst}, Y_{Dst})$ in an output image) by applying a linear combination of translation, rotation, scaling and/or shearing (i.e. non-uniform scaling in some directions) operations.
The parameters of the EasyImage::LinearTransform function are the coefficients of the affine equations below:

$$X_{Dst} = A_{xx}X_{Src} + A_{xy}Y_{Src} + A_x$$

$$Y_{Dst} = A_{yx}X_{Src} + A_{yy}Y_{Src} + A_y$$

# EasyImage::LineSegmentToImage

Copies the pixel values from a vector or a constant to the pixels of a given line segment (arbitrarily oriented).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void LineSegmentToImage(
 EROIBW8* destinationImage,
 EBW8 pixel,
 OEV_INT32 X0,
 OEV_INT32 Y0,
 OEV_INT32 X1,
 OEV_INT32 Y1
 )

void LineSegmentToImage(
 EROIBW16* destinationImage,
 EBW16 pixel,
 OEV_INT32 X0,
 OEV_INT32 Y0,
 OEV_INT32 X1,
 OEV_INT32 Y1
 )

void LineSegmentToImage(
 EROIC24* destinationImage,
 EC24 pixel,
 OEV_INT32 X0,
 OEV_INT32 Y0,
 OEV_INT32 X1,
 OEV_INT32 Y1
 )

void LineSegmentToImage(
 EROIBW8* destinationImage,
 EBW8Vector* path,
 OEV_INT32 X0,
 OEV_INT32 Y0,
 OEV_INT32 X1,
 OEV_INT32 Y1
 )

void LineSegmentToImage(
 EROIBW16* destinationImage,
 EBW16Vector* path,
 OEV_INT32 X0,
 OEV_INT32 Y0,
 OEV_INT32 X1,
 OEV_INT32 Y1
 )
```

```
void LineSegmentToImage(
 EROIC24* destinationImage,
 EC24Vector* path,
 OEV_INT32 X0,
 OEV_INT32 Y0,
 OEV_INT32 X1,
 OEV_INT32 Y1
 )
```

### Parameters

*destinationImage*

Pointer to the destination image/ROI.

*pixel*

Constant color value.

*X0*

Coordinates of the starting point of the segment.

*Y0*

Coordinates of the starting point of the segment.

*X1*

Coordinates of the ending point of the segment.

*Y1*

Coordinates of the ending point of the segment.

*path*

Pointer to the source vector.

### Remarks

The line segment must be wholly contained within the image.

# EasyImage::LocalAverage

Computes the average in a rectangular window centered on every pixel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void LocalAverage(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfWidth,
 OEV_UINT32 halfHeight
 )
void LocalAverage(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 halfWidth,
 OEV_UINT32 halfHeight
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*halfWidth*

Half of the window width minus one.

*halfHeight*

Half of the window height minus one.

### Remarks

The window dimensions can be an arbitrary odd integer.
The running time of this function does not depend on the window size.

# EasyImage::LocalDeviation

Computes the standard deviation in a rectangular window centered on every pixel.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void LocalDeviation(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfWidth,
 OEV_UINT32 halfHeight
 )

void LocalDeviation(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 halfWidth,
 OEV_UINT32 halfHeight
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. Must not be the same as the source image.

*halfWidth*

Half of the window width minus one.

*halfHeight*

Half of the window height minus one.

### Remarks

The window dimensions can be an arbitrary odd integer.
The running time of this function does not depend on the window size.

# EasyImage::Lut

Transforms the gray levels of an image, using a lookup table stored in a vector (of unsigned values).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Lut(
 const EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 EBW16Vector* lookupTable
 )

void Lut(
 const EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 EBW8Vector* lookupTable
 )

void Lut(
 const EROIBW16* sourceImage,
 EROIBW8* destinationImage,
 EBW8Vector* lookupTable,
 OEV_UINT32 numberOfScalingBits
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*lookupTable*

Pointer to the lookup vector.

*numberOfScalingBits*

Number of scaling bits (or right padding bits).

## Remarks

A 16-bit image usually does not make use of its 16 bits. In most cases, only 10 or 12 bits are used. These bits are called *significant bits*. In the 16-bit information, significant bits can be left aligned, right aligned or not aligned at all. To indicate which are the significant bits, we have to tell how many bits are significant and the number of right padding bits (0 right padding bit means that significant bits are right aligned).
The number of significant bits is given by the number of Look Up table entries. For example a Lut of 1024 entries is used for an image of 10 significant bits (as $2^{10} = 1024$).
The number of right padding bits is given by means of the **numberOfScalingBits** parameter. Leaving this parameter undefined indicates that the significant bits are left aligned on the word.

# EasyImage::MatchFrames

Determines the optimal shift amplitude by comparing two successive lines of the image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void MatchFrames(
 EROIBW8* sourceImage,
 OEV_INT32 fixedRow,
 OEV_INT32 minimumOffset,
 OEV_INT32 maximumOffset,
 OEV_INT32& bestOffset
 )
void MatchFrames(
 EROIBW16* sourceImage,
 OEV_INT32 fixedRow,
 OEV_INT32 minimumOffset,
 OEV_INT32 maximumOffset,
 OEV_INT32& bestOffset
 )
void MatchFrames(
 EROIC24* sourceImage,
 OEV_INT32 fixedRow,
 OEV_INT32 minimumOffset,
 OEV_INT32 maximumOffset,
 OEV_INT32& bestOffset
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*fixedRow*

Index of the line used for comparison. Line **fixedRow** remains in place and is compared with line (**fixedRow + 1**), shifted by some amount.

*minimumOffset*

Minimum value of the allowed offset (positive to the right).

*maximumOffset*

Maximum value of the allowed offset (positive to the right).

*bestOffset*

Estimated shift amplitude.

### Remarks

These lines should be chosen such that they cross some edges or non-uniform areas.
When an image is interlaced, the two frames (even and odd lines) are not recorded at the same time. If there is movement in the scene, a visible artifact can result (the edges of objects exhibit a "comb" effect).
When the movement is uniform and horizontal (objects on a conveyor belt), one cure to this problem is to shift one of the frames horizontally with respect to the other frame (using EasyImage::RealignFrame). The amplitude of the shift can be estimated automatically.

# EasyImage::Median

Applies a median filter to an image (median of the gray values in a 3x3 neighborhood).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Median(
 EROIBW1* sourceImage,
 EROIBW1* destinationImage
 )

void Median(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )

void Median(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )

void Median(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. Must not be the same as source image.

# EasyImage::ModulusImage

Prepares a lookup-table image for use for gradient magnitude computation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ModulusImage(
 EImageBW8* destinationImage,
 float gain
 )
```

### Parameters

*destinationImage*

Pointer to the destination image.

*gain*

Gain value to be applied to the modulus. **1** saturates; **1/Sqrt(2)** does not.

### Remarks

The modulus of the gradient argument can be adjusted to avoid saturation. EasyImage::ModulusImage sets a lookup-table image for use with function EasyImage::GradientScalar, ready to compute the modulus of the gradient in the source image, i.e. its amplitude (as defined by the Euclidian norm). The argument will be returned as a value in range **0..255** suitable for storage in an EImageBW8 or as a value in range **0..65535** suitable for storage in an EImageBW16. A gain coefficient can be adjusted to avoid saturation (gain = 1 saturates gradient amplitudes larger than 255 in the EBW8 case and 65535 in the EBW16 case; gain = **1/Sqrt(2)** never saturates).

# EasyImage::MorphoGradientBox

Computes the morphological gradient of an image using a rectangular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void MorphoGradientBox(
 EROIBW1* sourceImage,
 EROIBW1* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
void MorphoGradientBox(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
void MorphoGradientBox(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
void MorphoGradientBox(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half of the box width minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

*halfOfKernelHeight*

Half of the box height minus one (by default, same as **halfOfKernelWidth**; **0** is allowed).

### Remarks

The morphological gradient is the difference between the dilation and the erosion of the image, using the same structuring element.
The kernel size is a pair of odd numbers; they must be halved before they are passed. For instance, a 3x5 kernel is passed as 1x2.

# EasyImage::MorphoGradientDisk

Computes the morphological gradient of an image using a quasi-circular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void MorphoGradientDisk(
 EROIBW1* sourceImage,
 EROIBW1* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
void MorphoGradientDisk(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
void MorphoGradientDisk(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
```

```
void MorphoGradientDisk(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half width of the kernel minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

### Remarks

The morphological gradient is the difference between the dilation and the erosion of the image, using the same structuring element.

# EasyImage::Normalize

Normalizes an image, i.e. applies a linear transform to the gray levels so that their average and standard deviation are imposed.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Normalize(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 float imposedAverage,
 float imposedStandardDeviation
 )
```

```
void Normalize(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 float imposedAverage,
 float imposedStandardDeviation
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*imposedAverage*

Imposed average.

*imposedStandardDeviation*

Imposed standard deviation.

# EasyImage::Offset

Transforms an image, applying a gain and offset to all pixels.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Offset(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 EColor Offset
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*Offset*

Constant offset. By default (argument omitted) **0**, i.e. no change.

### Remarks

The gain should remain close to **1**, and allows contrast adjustment of the image.

The offset can be positive or negative, and allows to adjust the image intensity. The resulting values are always saturated to range **[0..255]**.

For color images, the separate gain and offset values are specified as triple of values stored in a EColor. The default values leave the image unchanged.

Internally, the computations are achieved through fixed-point arithmetic with 5 bits of precision for the fractional part. This can result in loss of precision with small gains.

# EasyImage::OpenBox

Performs an opening on an image (erosion followed by dilation) on a rectangular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void OpenBox(
 EROIBW1* sourceImage,
 EROIBW1* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
void OpenBox(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
```

```
void OpenBox(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )

void OpenBox(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

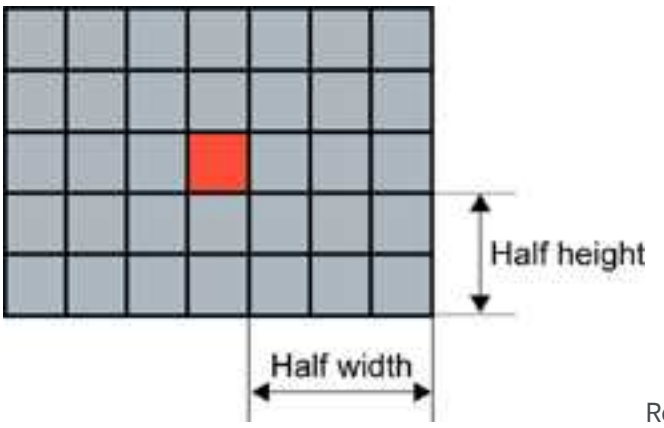
*halfOfKernelWidth*

Half of the box width minus one, as shown on the picture below (by default, **halfOfKernelWidth =1**; **0** is allowed).

*halfOfKernelHeight*

Half of the box height minus one, as shown on the picture below (by default, same as **halfOfKernelWidth**; **0** is allowed).

## Remarks




Rectangular kernel of half width = 3 and half height = 2

# EasyImage::OpenDisk

Performs an opening on an image (erosion followed by dilation) on a quasi-circular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void OpenDisk(
 EROIBW1* sourceImage,
 EROIBW1* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
void OpenDisk(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
void OpenDisk(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
void OpenDisk(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. If **NULL** (default), this operation is destructive (i.e. applied to the source image).

*halfOfKernelWidth*

Half width of the kernel minus one, as shown on the picture below (by default, **halfOfKernelWidth = 1**; **0** is allowed).

# EasyImage::Oper

Applies the desired arithmetic or logic pixel-wise operator between two images or constants.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 EBW1 constant,
 EROIBW1* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 const EROIBW1* sourceImage,
 EROIBW1* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 const EROIBW1* sourceImage0,
 const EROIBW1* sourceImage1,
 EROIBW1* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 EBW8 constant,
 EROIBW8* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 EBW16 constant,
 EROIBW16* destinationImage
 )
```

```
void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 EC24 constant,
 EROIC24* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 EBW8 constant,
 const EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 EBW16 constant,
 const EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 EC24 constant,
 const EROIC24* sourceImage,
 EROIC24* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 const EROIBW8* sourceImage,
 EBW8 constant,
 EROIBW8* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 const EROIBW16* sourceImage,
 EBW16 constant,
 EROIBW16* destinationImage
 )
```

```
void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 const EROIC24* sourceImage,
 EC24 constant,
 EROIC24* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 const EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 const EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 const EROIC24* sourceImage,
 EROIC24* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 const EROIBW8* sourceImage,
 EROIC24* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 const EROIBW8* sourceImage0,
 const EROIBW8* sourceImage1,
 EROIBW8* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 const EROIBW16* sourceImage0,
 const EROIBW16* sourceImage1,
 EROIBW16* destinationImage
 )
```

```
void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 const EROIC24* sourceImage0,
 const EROIC24* sourceImage1,
 EROIC24* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 const EROIBW8* sourceImage0,
 const EROIBW8* sourceImage1,
 EROIBW16* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 const EROIBW16* sourceImage0,
 const EROIBW8* sourceImage1,
 EROIBW16* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 const EROIBW8* sourceImage0,
 const EROIBW8* sourceImage1,
 EROIC24* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 const EROIBW8* sourceImage0,
 const EROIC24* sourceImage1,
 EROIC24* destinationImage
 )

void Oper(
 Euresys::Open_eVision_2_10::EArithmeticLogicOperation operation,
 const EROIC24* sourceImage0,
 const EROIBW8* sourceImage1,
 EROIC24* destinationImage
 )
```

## Parameters

*operation*

Arithmetic or logic operator, as defined by EArithmeticLogicOperation.

*constant*

Gray-level or color constant.

*destinationImage*

Pointer to the destination image/ROI.

*sourceImage*

Pointer to the second source image/ROI (right operand).

*sourceImage0*

Pointer to the first source image/ROI (left operand).

*sourceImage1*

Pointer to the second source image/ROI (right operand).

### Remarks

The source and destination images may be the same.
When the source operands are two color images/constants, the components are combined pair-wise. The result is a color image.
When the source operands are a color image and a gray-level image, each color component is combined with the gray-level component. The result is a color image.

# EasyImage::Overlay

Overlays an image on the top of a color image, at a given position.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Overlay(
 const EROIC24* sourceImage,
 EROIC16* destinationImage,
 OEV_INT32 panX,
 OEV_INT32 panY,
 EC24 referenceValue
 )
```

```
void Overlay(
 const EROIC24* sourceImage,
 EROIC15* destinationImage,
 OEV_INT32 panX,
 OEV_INT32 panY,
 EC24 referenceValue
 )

void Overlay(
 const EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_INT32 panX,
 OEV_INT32 panY,
 EC24 referenceValue
 )

void Overlay(
 const EROIC24* sourceImage,
 const EROIBW8* mask,
 EROIC15* destinationImage,
 OEV_INT32 panX,
 OEV_INT32 panY
 )

void Overlay(
 const EROIC24* sourceImage,
 const EROIBW8* mask,
 EROIC16* destinationImage,
 OEV_INT32 panX,
 OEV_INT32 panY
 )

void Overlay(
 const EROIC24* sourceImage,
 const EROIBW8* mask,
 EROIC24* destinationImage,
 OEV_INT32 panX,
 OEV_INT32 panY
 )
```

```
void Overlay(
 const EROIBW8* sourceImage,
 EROIC24* overlay,
 EROIC24* destinationImage,
 OEV_INT32 panX,
 OEV_INT32 panY,
 EC24 referenceValue
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*panX*

Translation for panning in the horizontal direction, in pixels.

*panY*

Translation for panning in the vertical direction, in pixels.

*referenceValue*

Reference color.

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

*overlay*

When a **BW8** source image is specified, pointer to the overlay image/ROI.

## Remarks

If a color image is provided as the source image, all the pixels of this image are copied to the destination image, but the ones that equal the reference color.
If a **BW8** image is provided as the source image, all the pixels of the overlay image are copied to the destination image, but the ones that equal the reference color, the latter being replaced by the content of the source image.

# EasyImage::GetOverlayColor

# EasyImage::SetOverlayColor

Gets/Sets the color of the overlay in the destination image when a **BW8** Image is used as overlay source image in functions.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
static EC24 GetOverlayColor()
static void SetOverlayColor(EC24 color)
```

## Remarks

**Note.** When a **C24** Image is used as overlay source image, the color of the overlay in destination image is the same as the one in the overlay source image, thus allowing multi colored overlays.

# EasyImage::PathToImage

Given a path described by coordinates in a path vector, copies the pixel values from the path vector to the corresponding image pixels.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void PathToImage(
 EROIBW8* sourceImage,
 EBW8PathVector* path
 )
```

```
void PathToImage(
 EROIBW16* sourceImage,
 EBW16PathVector* path
 )

void PathToImage(
 EROIC24* sourceImage,
 EC24PathVector* path
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*path*

Pointer to the destination vector.

# EasyImage::PixelAverage

Computes the average pixel value in a gray-level or color image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void PixelAverage(
 const EROIBW8* sourceImage,
 float& average
 )

void PixelAverage(
 const EROIBW16* sourceImage,
 float& average
 )
```

```
void PixelAverage(
 const EROIC24* sourceImage,
 float& average0,
 float& average1,
 float& average2
 )

void PixelAverage(
 const EROIBW8* sourceImage,
 const ERegion& region,
 float& average
 )

void PixelAverage(
 const EROIBW16* sourceImage,
 const ERegion& region,
 float& average
 )

void PixelAverage(
 const EROIC24* sourceImage,
 const ERegion& region,
 float& average0,
 float& average1,
 float& average2
 )

void PixelAverage(
 const EROIBW8* sourceImage,
 const EROIBW8* inputMask,
 float& average
 )

void PixelAverage(
 const EROIBW16* sourceImage,
 const EROIBW8* inputMask,
 float& average
 )

void PixelAverage(
 const EROIC24* sourceImage,
 const EROIBW8* inputMask,
 float& average0,
 float& average1,
 float& average2
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*average*

Reference to the average gray-level value.

*average0*

Reference to the average values for the first color channel.

*average1*

Reference to the average values for the second color channel.

*average2*

Reference to the average values for the third color channel.

*region*

Pointer to a region to apply the function only on a particular region in the image.

*inputMask*

Pointer to the mask, which allows functions to be applied on a particular region in the image.

# EasyImage::PixelCompare

Counts the number of pixels differing between two images.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 PixelCompare(
 const EROIBW1* sourceImage0,
 const EROIBW1* sourceImage1
 )

OEV_UINT32 PixelCompare(
 const EROIBW8* sourceImage0,
 const EROIBW8* sourceImage1
 )

OEV_UINT32 PixelCompare(
 const EROIBW16* sourceImage0,
 const EROIBW16* sourceImage1
 )
```

```
OEV_UINT32 PixelCompare(
 const EROIC24* sourceImage0,
 const EROIC24* sourceImage1
 )

OEV_UINT32 PixelCompare(
 const EROIBW8* sourceImage0,
 const ERegion& region,
 const EROIBW8* sourceImage1
 )

OEV_UINT32 PixelCompare(
 const EROIBW16* sourceImage0,
 const ERegion& region,
 const EROIBW16* sourceImage1
 )

OEV_UINT32 PixelCompare(
 const EROIC24* sourceImage0,
 const ERegion& region,
 const EROIC24* sourceImage1
 )

OEV_UINT32 PixelCompare(
 const EROIBW8* sourceImage0,
 const EROIBW8* sourceImage1,
 const EROIBW8* mask
 )

OEV_UINT32 PixelCompare(
 const EROIBW16* sourceImage0,
 const EROIBW16* sourceImage1,
 const EROIBW8* mask
 )

OEV_UINT32 PixelCompare(
 const EROIC24* sourceImage0,
 const EROIC24* sourceImage1,
 const EROIBW8* mask
 )
```

## Parameters

*sourceImage0*

Pointer to the first source image/ROI.

*sourceImage1*

Pointer to the second source image/ROI.

*region*

Pointer to a region to apply the function only on a particular region in the image.

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

# EasyImage::PixelCount

Counts the pixels in the three value classes separated by two thresholds.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void PixelCount(
 const EROIBW8* sourceImage,
 EBW8 lowThreshold,
 EBW8 highThreshold,
 OEV_INT32& numberOfPixelsBelowThreshold,
 OEV_INT32& numberOfPixelsBetweenThresholds,
 OEV_INT32& numberOfPixelsAboveThreshold
 )

void PixelCount(
 const EROIBW16* sourceImage,
 EBW16 lowThreshold,
 EBW16 highThreshold,
 OEV_INT32& numberOfPixelsBelowThreshold,
 OEV_INT32& numberOfPixelsBetweenThresholds,
 OEV_INT32& numberOfPixelsAboveThreshold
 )

void PixelCount(
 const EROIBW8* sourceImage,
 const ERegion& region,
 EBW8 lowThreshold,
 EBW8 highThreshold,
 OEV_INT32& numberOfPixelsBelowThreshold,
 OEV_INT32& numberOfPixelsBetweenThresholds,
 OEV_INT32& numberOfPixelsAboveThreshold
 )
```

```
void PixelCount(
 const EROIBW16* sourceImage,
 const ERegion& region,
 EBW16 lowThreshold,
 EBW16 highThreshold,
 OEV_INT32& numberOfPixelsBelowThreshold,
 OEV_INT32& numberOfPixelsBetweenThresholds,
 OEV_INT32& numberOfPixelsAboveThreshold
 )

void PixelCount(
 const EROIBW8* sourceImage,
 const ERegion& region,
 EBW8 lowThreshold,
 EBW8 highThreshold,
 OEV_UINT64& numberOfPixelsBelowThreshold,
 OEV_UINT64& numberOfPixelsBetweenThresholds,
 OEV_UINT64& numberOfPixelsAboveThreshold
 )

void PixelCount(
 const EROIBW16* sourceImage,
 const ERegion& region,
 EBW16 lowThreshold,
 EBW16 highThreshold,
 OEV_UINT64& numberOfPixelsBelowThreshold,
 OEV_UINT64& numberOfPixelsBetweenThresholds,
 OEV_UINT64& numberOfPixelsAboveThreshold
 )

void PixelCount(
 const EROIBW8* sourceImage,
 EBW8 lowThreshold,
 EBW8 highThreshold,
 OEV_UINT64& numberOfPixelsBelowThreshold,
 OEV_UINT64& numberOfPixelsBetweenThresholds,
 OEV_UINT64& numberOfPixelsAboveThreshold
 )
```

```
void PixelCount(
 const EROIBW16* sourceImage,
 EBW16 lowThreshold,
 EBW16 highThreshold,
 OEV_UINT64& numberOfPixelsBelowThreshold,
 OEV_UINT64& numberOfPixelsBetweenThresholds,
 OEV_UINT64& numberOfPixelsAboveThreshold
 )

void PixelCount(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 EBW8 lowThreshold,
 EBW8 highThreshold,
 OEV_INT32& numberOfPixelsBelowThreshold,
 OEV_INT32& numberOfPixelsBetweenThresholds,
 OEV_INT32& numberOfPixelsAboveThreshold
 )

void PixelCount(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 EBW16 lowThreshold,
 EBW16 highThreshold,
 OEV_INT32& numberOfPixelsBelowThreshold,
 OEV_INT32& numberOfPixelsBetweenThresholds,
 OEV_INT32& numberOfPixelsAboveThreshold
 )

void PixelCount(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 EBW8 lowThreshold,
 EBW8 highThreshold,
 OEV_UINT64& numberOfPixelsBelowThreshold,
 OEV_UINT64& numberOfPixelsBetweenThresholds,
 OEV_UINT64& numberOfPixelsAboveThreshold
 )
```

```
void PixelCount(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 EBW16 lowThreshold,
 EBW16 highThreshold,
 OEV_UINT64& numberOfPixelsBelowThreshold,
 OEV_UINT64& numberOfPixelsBetweenThresholds,
 OEV_UINT64& numberOfPixelsAboveThreshold
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*lowThreshold*

Inferior threshold.

*highThreshold*

Superior threshold.

*numberOfPixelsBelowThreshold*

Reference to the count of pixels strictly below the inferior threshold.

*numberOfPixelsBetweenThresholds*

Reference to the count of pixels above or equal to the inferior threshold, and strictly below the superior threshold.

*numberOfPixelsAboveThreshold*

Reference to the count of pixels above or equal to the superior threshold.

*region*

Pointer to a region to apply the function only on a particular region in the image.

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

# EasyImage::PixelMax

Computes the maximum gray-level value in an image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void PixelMax(
 const EROIBW8* sourceImage,
 EBW8& maximumValue
 )

void PixelMax(
 const EROIBW16* sourceImage,
 EBW16& maximumValue
 )

void PixelMax(
 const EROIBW8* sourceImage,
 const ERegion& region,
 EBW8& maximumValue
 )

void PixelMax(
 const EROIBW16* sourceImage,
 const ERegion& region,
 EBW16& maximumValue
 )

void PixelMax(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 EBW8& maximumValue
 )

void PixelMax(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 EBW16& maximumValue
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*maximumValue*

Reference to the maximum value.

*region*

Pointer to a region to apply the function only on a particular region in the image.

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

# EasyImage::PixelMaxBW16

Computes the maximum gray-level value in an image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void PixelMaxBW16(
 const EROIBW16* sourceImage,
 EBW16& maximumValue
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*maximumValue*

Reference to the maximum value.

# EasyImage::PixelMaxBW8

Computes the maximum gray-level value in an image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void PixelMaxBW8(
 const EROIBW8* sourceImage,
 EBW8& maximumValue
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*maximumValue*

Reference to the maximum value.

# EasyImage::PixelMin

Computes the minimum gray-level value in an image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void PixelMin(
 const EROIBW8* sourceImage,
 EBW8& minimumValue
 )

void PixelMin(
 const EROIBW16* sourceImage,
 EBW16& minimumValue
 )

void PixelMin(
 const EROIBW8* sourceImage,
 const ERegion& region,
 EBW8& minimumValue
 )
```

```
void PixelMin(
 const EROIBW16* sourceImage,
 const ERegion& region,
 EBW16& minimumValue
 )
void PixelMin(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 EBW8& minimumValue
 )
void PixelMin(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 EBW16& minimumValue
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*minimumValue*

Reference to the minimum value.

*region*

Pointer to a region to apply the function only on a particular region in the image.

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

# EasyImage::PixelMinBW16

Computes the minimum gray-level value in an image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void PixelMinBW16(
 const EROIBW16* sourceImage,
 EBW16& minimumValue
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*minimumValue*

Reference to the minimum value.

# EasyImage::PixelMinBW8

Computes the minimum gray-level value in an image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void PixelMinBW8(
 const EROIBW8* sourceImage,
 EBW8& minimumValue
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*minimumValue*

Reference to the minimum value.

# EasyImage::PixelStat

Computes the minimum, maximum and average gray-level values in an image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void PixelStat(
 const EROIBW8* sourceImage,
 EBW8& minimumValue,
 EBW8& maximumValue,
 float& average
 )

void PixelStat(
 const EROIBW16* sourceImage,
 EBW16& minimumValue,
 EBW16& maximumValue,
 float& average
 )

void PixelStat(
 const EROIBW8* sourceImage,
 const ERegion& region,
 EBW8& minimumValue,
 EBW8& maximumValue,
 float& average
 )

void PixelStat(
 const EROIBW16* sourceImage,
 const ERegion& region,
 EBW16& minimumValue,
 EBW16& maximumValue,
 float& average
 )
```

```
void PixelStat(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 EBW8& minimumValue,
 EBW8& maximumValue,
 float& average
 )
void PixelStat(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 EBW16& minimumValue,
 EBW16& maximumValue,
 float& average
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*minimumValue*

Reference to the minimum value.

*maximumValue*

Reference to the maximum value.

*average*

Reference to the average value.

*region*

Pointer to a region to apply the function only on a particular region in the image.

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

# EasyImage::PixelStatBW16

Computes the minimum, maximum and average gray-level values in an image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void PixelStatBW16(
 const EROIBW16* sourceImage,
 EBW16& minimumValue,
 EBW16& maximumValue,
 float& average
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*minimumValue*

Reference to the minimum value.

*maximumValue*

Reference to the maximum value.

*average*

Reference to the average value.

# EasyImage::PixelStatBW8

Computes the minimum, maximum and average gray-level values in an image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void PixelStatBW8(
 const EROIBW8* sourceImage,
 EBW8& minimumValue,
 EBW8& maximumValue,
 float& average
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*minimumValue*

Reference to the minimum value.

*maximumValue*

Reference to the maximum value.

*average*

Reference to the average value.

# EasyImage::PixelStdDev

Computes the average gray-level or color value in an image, the standard deviation of the color components, and the correlation between the color components (in the case of color images).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void PixelStdDev(
 const EROIBW8* sourceImage,
 float& standardDeviation,
 float& mean
 )

void PixelStdDev(
 const EROIBW16* sourceImage,
 float& standardDeviation,
 float& mean
 )
```

```
void PixelStdDev(
 const EROIC24* sourceImage,
 float& standardDeviation0,
 float& standardDeviation1,
 float& standardDeviation2,
 float& correlation01,
 float& correlation12,
 float& correlation20,
 float& mean0,
 float& mean1,
 float& mean2
 )

void PixelStdDev(
 const EROIBW8* sourceImage,
 const ERegion& region,
 float& standardDeviation,
 float& mean
 )

void PixelStdDev(
 const EROIBW16* sourceImage,
 const ERegion& region,
 float& standardDeviation,
 float& mean
 )

void PixelStdDev(
 const EROIC24* sourceImage,
 const ERegion& region,
 float& standardDeviation0,
 float& standardDeviation1,
 float& standardDeviation2,
 float& correlation01,
 float& correlation12,
 float& correlation20,
 float& mean0,
 float& mean1,
 float& mean2
 )
```

```
void PixelStdDev(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 float& standardDeviation,
 float& mean
 )

void PixelStdDev(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 float& standardDeviation,
 float& mean
 )

void PixelStdDev(
 const EROIC24* sourceImage,
 const EROIBW8* mask,
 float& standardDeviation0,
 float& standardDeviation1,
 float& standardDeviation2,
 float& correlation01,
 float& correlation12,
 float& correlation20,
 float& mean0,
 float& mean1,
 float& mean2
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*standardDeviation*

Reference to a variable in which the standard deviation of the pixel values is to be stored (for gray-level images).

*mean*

Reference to a variable in which the average value of the pixels is to be stored (for gray-level images).

*standardDeviation0*

Reference to a variable in which the standard deviation of the values of the first color component is to be stored (for color images).

*standardDeviation1*

Reference to a variable in which the standard deviation of the values of the second color component is to be stored (for color images).

*standardDeviation2*

Reference to a variable in which the standard deviation of the values of the third color component is to be stored (for color images).

*correlation01*

Reference to a variable in which the correlation between the values of the first color component and the second color component is to be stored (for color images).

*correlation12*

Reference to a variable in which the correlation between the values of the second color component and the third color component is to be stored (for color images).

*correlation20*

-

*mean0*

Reference to a variable in which the average value of the first color component is to be stored (for color images).

*mean1*

Reference to a variable in which the average value of the second color component is to be stored (for color images).

*mean2*

Reference to a variable in which the average value of the third color component is to be stored (for color images).

*region*

Pointer to a region to apply the function only on a particular region in the image.

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

### Remarks

The variance can be obtained from the standard deviation by squaring it.

# EasyImage::PixelVariance

For a gray-level image, computes the mean and variance of the pixel values.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void PixelVariance(
 const EROIBW8* sourceImage,
 float& variance,
 float& mean
 )

void PixelVariance(
 const EROIBW16* sourceImage,
 float& variance,
 float& mean
 )

void PixelVariance(
 const EROIC24* sourceImage,
 float& variance0,
 float& variance1,
 float& variance2,
 float& covariance01,
 float& covariance12,
 float& covariance20,
 float& mean0,
 float& mean1,
 float& mean2
 )

void PixelVariance(
 const EROIBW8* sourceImage,
 const ERegion& region,
 float& variance,
 float& mean
 )

void PixelVariance(
 const EROIBW16* sourceImage,
 const ERegion& region,
 float& variance,
 float& mean
 )
```

```
void PixelVariance(
 const EROIC24* sourceImage,
 const ERegion& region,
 float& variance0,
 float& variance1,
 float& variance2,
 float& covariance01,
 float& covariance12,
 float& covariance20,
 float& mean0,
 float& mean1,
 float& mean2
 )

void PixelVariance(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 float& variance,
 float& mean
 )

void PixelVariance(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 float& variance,
 float& mean
 )

void PixelVariance(
 const EROIC24* sourceImage,
 const EROIBW8* mask,
 float& variance0,
 float& variance1,
 float& variance2,
 float& covariance01,
 float& covariance12,
 float& covariance20,
 float& mean0,
 float& mean1,
 float& mean2
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*variance*

Reference to the covariances of the pairs of pixel component values.

*mean*

Reference to the mean pixel component values.

*variance0*

Reference to the covariances of the pairs of pixel component values.

*variance1*

Reference to the covariances of the pairs of pixel component values.

*variance2*

Reference to the covariances of the pairs of pixel component values.

*covariance01*

Reference to the covariances of the pairs of pixel component values.

*covariance12*

Reference to the covariances of the pairs of pixel component values.

*covariance20*

Reference to the covariances of the pairs of pixel component values.

*mean0*

Reference to the mean pixel component values.

*mean1*

Reference to the mean pixel component values.

*mean2*

Reference to the mean pixel component values.

*region*

Pointer to a region to apply the function only on a particular region in the image.

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

### Remarks

For a color image, computes the means of the three pixel color components, the variances of the components and the covariances between pairs of components.

# EasyImage::ProfileDerivative

Computes the first derivative of a profile extracted from a gray-level image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ProfileDerivative(
 EBW8Vector* sourceVector,
 EBW8Vector* destinationVector
 )

void ProfileDerivative(
 EBW16Vector* sourceVector,
 EBW16Vector* destinationVector
 )
```

### Parameters

*sourceVector*

Pointer to the source vector.

*destinationVector*

Pointer to the destination vector.

### Remarks

Taking the derivative transforms transitions (edges) into peaks.

**Note.** Since the EBW8 data type only handles unsigned values, the derivative is shifted up by 128. Values under [above] 128 correspond to negative [positive] derivative (decreasing [increasing] slope).

# EasyImage::ProjectOnAColumn

Projects an image horizontally onto a column.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ProjectOnAColumn(
 const EROIBW8* sourceImage,
 EBW32Vector* destinationVector
 )
```

```
void ProjectOnAColumn(
 const EROIBW16* sourceImage,
 EBW32Vector* destinationVector
 )

void ProjectOnAColumn(
 const EROIBW8* sourceImage,
 EBW8Vector* destinationVector
 )

void ProjectOnAColumn(
 const EROIBW16* sourceImage,
 EBW16Vector* destinationVector
 )

void ProjectOnAColumn(
 const EROIC24* sourceImage,
 EC24Vector* destinationVector
 )

void ProjectOnAColumn(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 EBW32Vector* destinationVector
 )

void ProjectOnAColumn(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 EBW32Vector* destinationVector
 )

void ProjectOnAColumn(
 const EROIBW8* sourceImage,
 EROIBW8* mask,
 EBW8Vector* destinationVector
 )

void ProjectOnAColumn(
 const EROIBW16* sourceImage,
 EROIBW8* mask,
 EBW16Vector* destinationVector
 )
```

```
void ProjectOnAColumn(
 const EROIC24* sourceImage,
 EROIBW8* mask,
 EC24Vector* destinationVector
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationVector*

Pointer to the destination vector.

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

### Remarks

Pixel gray/color levels are added when projecting into an EBW32Vector. When projecting into an EBW8Vector/EBW16Vector/EC24Vector, pixel values are averaged, instead.

# EasyImage::ProjectOnARow

Projects an image vertically onto a row.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ProjectOnARow(
 const EROIBW8* sourceImage,
 EBW32Vector* destinationVector
 )

void ProjectOnARow(
 const EROIBW16* sourceImage,
 EBW32Vector* destinationVector
 )
```

```
void ProjectOnARow(
 const EROIBW8* sourceImage,
 EBW8Vector* destinationVector
 )

void ProjectOnARow(
 const EROIBW16* sourceImage,
 EBW16Vector* destinationVector
 )

void ProjectOnARow(
 const EROIC24* sourceImage,
 EC24Vector* destinationVector
 )

void ProjectOnARow(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 EBW32Vector* destinationVector
 )

void ProjectOnARow(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 EBW32Vector* destinationVector
 )

void ProjectOnARow(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 EBW8Vector* destinationVector
 )

void ProjectOnARow(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 EBW16Vector* destinationVector
 )

void ProjectOnARow(
 const EROIC24* sourceImage,
 const EROIBW8* mask,
 EC24Vector* destinationVector
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationVector*

Pointer to the destination vector.

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

### Remarks

Pixel gray/color levels are added when projecting into an EBW32Vector. When projecting into an EBW8Vector/EBW16Vector/EC24Vector, pixel values are averaged, instead.

# EasyImage::RealignFrame

Shifts one frame of the image horizontally.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RealignFrame(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_INT32 offset,
 OEV_UINT32 fixedRow
 )

void RealignFrame(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_INT32 offset,
 OEV_UINT32 fixedRow
 )

void RealignFrame(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_INT32 offset,
 OEV_UINT32 fixedRow
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*offset*

Indicates the number of pixels by which to shift (positive to the right).

*fixedRow*

Specifies which frame remains unchanged (the frame made up of all lines of the same parity as **fixedRow**; by default, **fixedRow = 0**).

### Remarks

The same image should be used as source and destination. If the destination image differs from the source image, only the shifted rows are copied. To use a different destination image, the source image must be copied first in the destination image object.
When an image is interlaced, the two frames (even and odd lines) are not recorded at the same time. If there is movement in the scene, a visible artifact can result (the edges of objects exhibit a "comb" effect).
When the movement is uniform and horizontal (objects on a conveyor belt), one cure to this problem is to shift one of the frames horizontally with respect to the other frame. The amplitude of the shift can be estimated automatically (using EasyImage::MatchFrames).

# EasyImage::RebuildFrame

Rebuilds one frame of the image by interpolation between the lines of the other frame.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RebuildFrame(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 fixedRow
 )
```

```
void RebuildFrame(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 fixedRow
 )
void RebuildFrame(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_UINT32 fixedRow
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*fixedRow*

Specifies which frame remains unchanged (the frame made up of all lines of the same parity as **fixedRow**; by default, **fixedRow = 0**).

### Remarks

The same image should be used as source and destination. If the destination image differs from the source image, only the shifted rows are copied. To use a different destination image, the source image must be copied first in the destination image object. When an image is interlaced, the two frames (even and odd lines) are not recorded at the same time. If there is movement in the scene, a visible artifact can result (the edges of objects exhibit a "comb" effect). One cure to this problem is to replace one of the frames by linearly interpolating between the lines of the other frame.

# EasyImage::RecursiveAverage

Applies stronger noise reduction to small variations and conversely.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void RecursiveAverage(
 EROIBW8* sourceImage,
 EROIBW16* store,
 EROIBW8* destinationImage,
 EBW16Vector* lookupTable
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*store*

Pointer to a 16-bit work image.

*destinationImage*

Pointer to the destination image/ROI.

*lookupTable*

Pointer to the LUT vector generated by a call to EasyImage::SetRecursiveAverageLUT.

### Remarks

Recursive averaging is a well known process for noise reduction by temporal integration. The principle is to continuously update a noise-free image by blending it, using a linear combination, with the raw, noisy, live image stream.
Algorithmically, this amounts to apply the following recurrence: where **a** is a mixture coefficient. The value of this coefficient can be adjusted so that a prescribed noise reduction ratio is achieved. This procedure is effective when applied to still images, but generates a trailing effect on moving objects because of the transient behavior of the filter. The larger the noise reduction ratio, the heavier the trailing effect. To work around this, a non-linearity can be introduced in the blending process: small gray-level values variations between successive images are usually caused by noise, while large variations correspond to changes in the signal itself (camera displacement or object movements). Function EasyImage::RecursiveAverage uses this observation and applies stronger noise reduction to small variations and conversely. This way, noise is better reduced in still areas and trailing is avoided in moving areas.
For optimal performance, the non-linearity must be pre-computed once for all using function EasyImage::SetRecursiveAverageLUT.

**Note.** Before the first call to the EasyImage::RecursiveAverage method, the 16-bit work image *must* be cleared (all pixel values set to zero).

# EasyImage::Register

Registers an image by realigning one, two or three pivot points to reference positions.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Register(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 float sourceImagePivot0X,
 float sourceImagePivot0Y,
 float destinationImagePivot0X,
 float destinationImagePivot0Y,
 OEV_INT32 interpolationBits
 )

void Register(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 float sourceImagePivot0X,
 float sourceImagePivot0Y,
 float destinationImagePivot0X,
 float destinationImagePivot0Y,
 OEV_INT32 interpolationBits
 )

void Register(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 float sourceImagePivot0X,
 float sourceImagePivot0Y,
 float destinationImagePivot0X,
 float destinationImagePivot0Y,
 OEV_INT32 interpolationBits
 )
```

```
void Register(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 float sourceImagePivot0X,
 float sourceImagePivot0Y,
 float sourceImagePivot1X,
 float sourceImagePivot1Y,
 float destinationImagePivot0X,
 float destinationImagePivot0Y,
 float destinationImagePivot1X,
 float destinationImagePivot1Y,
 OEV_INT32 interpolationBits,
 BOOL resize
 )

void Register(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 float sourceImagePivot0X,
 float sourceImagePivot0Y,
 float sourceImagePivot1X,
 float sourceImagePivot1Y,
 float destinationImagePivot0X,
 float destinationImagePivot0Y,
 float destinationImagePivot1X,
 float destinationImagePivot1Y,
 OEV_INT32 interpolationBits,
 BOOL resize
 )

void Register(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 float sourceImagePivot0X,
 float sourceImagePivot0Y,
 float sourceImagePivot1X,
 float sourceImagePivot1Y,
 float destinationImagePivot0X,
 float destinationImagePivot0Y,
 float destinationImagePivot1X,
 float destinationImagePivot1Y,
 OEV_INT32 interpolationBits,
 BOOL resize
 )
```

```
void Register(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 float sourceImagePivot0X,
 float sourceImagePivot0Y,
 float sourceImagePivot1X,
 float sourceImagePivot1Y,
 float sourceImagePivot2X,
 float sourceImagePivot2Y,
 float destinationImagePivot0X,
 float destinationImagePivot0Y,
 float destinationImagePivot1X,
 float destinationImagePivot1Y,
 float destinationImagePivot2X,
 float destinationImagePivot2Y,
 OEV_INT32 interpolationBits
 )

void Register(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 float sourceImagePivot0X,
 float sourceImagePivot0Y,
 float sourceImagePivot1X,
 float sourceImagePivot1Y,
 float sourceImagePivot2X,
 float sourceImagePivot2Y,
 float destinationImagePivot0X,
 float destinationImagePivot0Y,
 float destinationImagePivot1X,
 float destinationImagePivot1Y,
 float destinationImagePivot2X,
 float destinationImagePivot2Y,
 OEV_INT32 interpolationBits
 )
```

```
void Register(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 float sourceImagePivot0X,
 float sourceImagePivot0Y,
 float sourceImagePivot1X,
 float sourceImagePivot1Y,
 float sourceImagePivot2X,
 float sourceImagePivot2Y,
 float destinationImagePivot0X,
 float destinationImagePivot0Y,
 float destinationImagePivot1X,
 float destinationImagePivot1Y,
 float destinationImagePivot2X,
 float destinationImagePivot2Y,
 OEV_INT32 interpolationBits
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. May not be the same as the source image.

*sourceImagePivot0X*

First pivot point abscissa in the source image.

*sourceImagePivot0Y*

First pivot point ordinate in the source image.

*destinationImagePivot0X*

First pivot point abscissa in the destination image.

*destinationImagePivot0Y*

First pivot point ordinate in the destination image.

*interpolationBits*

Number of bits of accuracy for interpolation. Allowed values are **0** (no interpolation, nearest neighbor), **4 (linear interpolation)** or **8 (cubic interpolation)**.

*sourceImagePivot1X*

Second pivot point abscissa in the source image.

*sourceImagePivot1Y*

Second pivot point ordinate in the source image.

*destinationImagePivot1X*

Second pivot point abscissa in the destination image.

*destinationImagePivot1Y*

Second pivot point ordinate in the destination image.

*resize*

**TRUE** if scaling is allowed.

*sourceImagePivot2X*

Third pivot point abscissa in the source image.

*sourceImagePivot2Y*

Third pivot point ordinate in the source image.

*destinationImagePivot2X*

Third pivot point abscissa in the destination image.

*destinationImagePivot2Y*

Third pivot point ordinate in the destination image.

### Remarks

Out-of-image-bounds pixels are black.
*Registration* is the process of realigning two misaligned images so that point-to-point comparisons are possible. The simplest way to achieve this is to accurately locate features in both images (landmarks or pivots), using pattern matching, point measurement or whatever other technique, and realign one of the images so that the landmarks are superimposed.
* When a single pivot point is used, the registration transform is a simple translation. If interpolation bits are used, sub-pixel translation is achieved.
* When two pivot points are used, the registration is a combination of translation, rotation and optionally scaling. If scaling is not allowed, the second pivot point will not be matched exactly in general. Anyway, for most applications scaling should not be used unless it corresponds to a change of lens magnification or viewing distance.
* When three pivot points are used, the registration is a combination of translation, rotation, shearing correction and optionally scaling. The so-called shear effect can arise when acquiring images with a misaligned line-scan camera.
To achieve good accuracy, the pivot points should be chosen as far apart as possible.

# EasyImage::RmsNoise

Computes the root-mean-square amplitude of noise, by comparing a given image to a reference image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float RmsNoise(
 const EROIBW8* sourceImage,
 const EROIBW8* referenceImage,
 Euresys::Open_eVision_2_10::EReferenceNoise referenceNoise
 )

float RmsNoise(
 const EROIBW16* sourceImage,
 const EROIBW16* referenceImage,
 Euresys::Open_eVision_2_10::EReferenceNoise referenceNoise
 )

float RmsNoise(
 const EROIC24* sourceImage,
 const EROIC24* referenceImage,
 Euresys::Open_eVision_2_10::EReferenceNoise referenceNoise
 )

float RmsNoise(
 const EROIBW8* sourceImage,
 const EROIBW8* referenceImage,
 const EROIBW8* mask,
 Euresys::Open_eVision_2_10::EReferenceNoise referenceNoise
 )

float RmsNoise(
 const EROIBW16* sourceImage,
 const EROIBW16* referenceImage,
 const EROIBW8* mask,
 Euresys::Open_eVision_2_10::EReferenceNoise referenceNoise
 )

float RmsNoise(
 const EROIC24* sourceImage,
 const EROIC24* referenceImage,
 const EROIBW8* mask,
 Euresys::Open_eVision_2_10::EReferenceNoise referenceNoise
 )

float RmsNoise(
 const EROIBW8* sourceImage,
 const EROIBW16* referenceImage,
 OEV_UINT32 count,
 Euresys::Open_eVision_2_10::EReferenceNoise referenceNoise
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*referenceImage*

Pointer to the reference image/ROI.

*referenceNoise*

Specifies how the reference image is affected by noise, as defined by EReferenceNoise.

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

*count*

-

### Remarks

The reference image can be noiseless (obtained by suppressing the source of noise), or affected by a noise of the same distribution as the given image.

# EasyImage::ScaleRotate

Re-scales an image by an arbitrary factor and/or rotates it by an arbitrary angle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ScaleRotate(
 EROIBW8* sourceImage,
 float sourceImagePivotX,
 float sourceImagePivotY,
 float destinationImagePivotX,
 float destinationImagePivotY,
 float scaleX,
 float scaleY,
 float rotation,
 EROIBW8* destinationImage,
 OEV_INT32 interpolationBits
 )
```

```
void ScaleRotate(
 EROIC24* sourceImage,
 float sourceImagePivotX,
 float sourceImagePivotY,
 float destinationImagePivotX,
 float destinationImagePivotY,
 float scaleX,
 float scaleY,
 float rotation,
 EROIC24* destinationImage,
 OEV_INT32 interpolationBits
 )
void ScaleRotate(
 EROIBW16* sourceImage,
 float sourceImagePivotX,
 float sourceImagePivotY,
 float destinationImagePivotX,
 float destinationImagePivotY,
 float scaleX,
 float scaleY,
 float rotation,
 EROIBW16* destinationImage,
 OEV_INT32 interpolationBits
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*sourceImagePivotX*

Pivot point abscissa in the source image.

*sourceImagePivotY*

Pivot point ordinate in the source image.

*destinationImagePivotX*

Pivot point abscissa in the destination image.

*destinationImagePivotY*

Pivot point ordinate in the destination image.

*scaleX*

Scale factor for the abscissas. Its value must be different than **0.0**.

*scaleY*

Scale factor for the ordinates. Its value must be different than **0.0**.

*rotation*

Rotation angle, using the current angle unit.

*destinationImage*

Pointer to the destination image/ROI. May not be the same as the source image.

*interpolationBits*

Number of bits of accuracy for interpolation. Allowed values are **0** (no interpolation, nearest neighbor), **4 (linear interpolation)** or **8 (cubic interpolation)**.

### Remarks

For resampling, the nearest neighbor rule or bilinear interpolation with 4 or 8 bits of accuracy is used.
The pivot point is a given point in the source image which is mapped to a given point in the destination image. Rotation and scaling are done around the pivot point. The pivot point reference coordinates are based on the 'Pixel Coordinate System', meaning that the origin (0, 0) is the center of the top left pixel of the image.
Out-of-image-bounds pixels are black.

# EasyImage::SetCircleWarp

Prepares suitable warp images for use with function EasyImage::Warp to unwarp a circular ring-wedge shape into a straight rectangle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetCircleWarp(
 float centerX,
 float centerY,
 OEV_INT32 numberOfRadialSampledPoints,
 float minimumRadius,
 float maximumRadius,
 OEV_INT32 numberOfTangentSampledPoints,
 float minimumAngle,
 float maximumAngle,
 EImageBW16* warpImageX,
 EImageBW16* warpImageY
 )
```

### Parameters

*centerX*

Abscissa of the ring-wedge center.

*centerY*

Ordinate of the ring-wedge center.

*numberOfRadialSampledPoints*

Number of points to be sampled in the radial direction.

*minimumRadius*

Starting radius of the ring-wedge shape.

*maximumRadius*

Ending radius of the ring-wedge shape.

*numberOfTangentSampledPoints*

Number of points to be sampled in the tangent direction.

*minimumAngle*

Starting angle of the ring-wedge shape.

*maximumAngle*

Ending angle of the ring-wedge shape.

*warpImageX*

Destination warp image for the abscissas.

*warpImageY*

Destination warp image for the ordinates.

### Remarks

Typical use is unwarping of a text printed around a circle.

**Note.** A ring-wedge is delimited by two concentric circles and two straight lines passing through the center.

# EasyImage::SetFrame

Replaces the frame of given parity in an image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void SetFrame(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 BOOL odd
 )
void SetFrame(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 BOOL odd
 )
void SetFrame(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 BOOL odd
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. Must not be the same as the source image.

*odd*

Specifies which frame is replaced (the frame made up of all lines of the same parity as **odd**).

### Remarks

The size of the destination image is determined as follows: **DstImage_Width = SrcImage_WidthDstImage_Height = (SrcImage_Height + 1 - odd ) / 2**

# EasyImage::SetRecursiveAverageLUT

Pre-compute the required non-linear transfer function for noise reduction by recursive temporal averaging.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetRecursiveAverageLUT(
 EBW16Vector* lookupTable,
 float reductionNoiseFactor,
 float reductionNoiseWidth
 )
```

### Parameters

*lookupTable*

Pointer to the LUT vector holding the non-linear transfer function.

*reductionNoiseFactor*

Noise reduction factor. The larger the value, the more effectively noise will be reduced.

*reductionNoiseWidth*

Indicates the extent to which noise reduction applies to large variations in gray-level values. For variations small with respect to this parameter, noise will be reduced by a factor close to the **reductionNoiseFactor** value; for variations much larger than **reductionNoiseWidth**, no noise reduction will take place.

### Remarks

This function is a companion to EasyImage::RecursiveAverage.

# EasyImage::SetupEqualize

Prepares a lookup-table for image equalization, using an image histogram.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetupEqualize(
 EBWHistogramVector* histogram,
 EBW8Vector* lookupTable
 )
```

### Parameters

*histogram*

Pointer to the source histogram vector.

*lookupTable*

Pointer to the destination lookup-table vector.

### Remarks

This function, along with EasyImage::Histogram and EasyImage::Lut, is an alternative to using EasyImage::Equalize.

# EasyImage::Shrink

Resizes an image to a smaller size. Pre-filtering is applied to avoid aliasing.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Shrink(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )
void Shrink(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
void Shrink(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

# EasyImage::SignalNoiseRatio

Computes the signal to noise ratio, in dB, by comparing a given image to a reference image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float SignalNoiseRatio(
 const EROIBW8* sourceImage,
 const EROIBW8* referenceImage,
 Euresys::Open_eVision_2_10::EReferenceNoise referenceNoise
 )

float SignalNoiseRatio(
 const EROIBW16* sourceImage,
 const EROIBW16* referenceImage,
 Euresys::Open_eVision_2_10::EReferenceNoise referenceNoise
 )

float SignalNoiseRatio(
 const EROIC24* sourceImage,
 const EROIC24* referenceImage,
 Euresys::Open_eVision_2_10::EReferenceNoise referenceNoise
 )

float SignalNoiseRatio(
 const EROIBW8* sourceImage,
 const EROIBW8* referenceImage,
 const EROIBW8* mask,
 Euresys::Open_eVision_2_10::EReferenceNoise referenceNoise
 )

float SignalNoiseRatio(
 const EROIBW16* sourceImage,
 const EROIBW16* referenceImage,
 const EROIBW8* mask,
 Euresys::Open_eVision_2_10::EReferenceNoise referenceNoise
 )
```

```
float SignalNoiseRatio(
 const EROIC24* sourceImage,
 const EROIC24* referenceImage,
 const EROIBW8* mask,
 Euresys::Open_eVision_2_10::EReferenceNoise referenceNoise
 )
float SignalNoiseRatio(
 EROIBW8* pSrcImage,
 EROIBW16* pRefImage,
 OEV_UINT32 un32Count,
 Euresys::Open_eVision_2_10::EReferenceNoise eReferenceNoise
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*referenceImage*

Pointer to the reference image/ROI.

*referenceNoise*

Specifies how the reference image is affected by noise, as defined by EReferenceNoise.

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

*pSrcImage*

-

*pRefImage*

-

*un32Count*

-

*eReferenceNoise*

-

## Remarks

The reference image can be noiseless (obtained by suppressing the source of noise) or be affected by a noise of the same distribution as the given image.
The signal amplitude is defined as the sum of the squared pixel gray-level values while the noise amplitude is defined as the sum of the squared difference between the pixel gray-level values of the given image and the reference.

# EasyImage::SwapFrames

Interchanges the even and odd rows of an image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SwapFrames(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )
void SwapFrames(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )
void SwapFrames(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

### Remarks

This is helpful when acquisition of an interleaved image has confused even and odd frames.

# EasyImage::Thick

Applies a thickening operation on an image, using a 3x3 kernel.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Thick(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 EKernel* thickeningKernel,
 Euresys::Open_eVision_2_10::EKernelRotation rotationMode,
 OEV_INT32& numberOfIterations
 )

void Thick(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 EKernel* thickeningKernel,
 Euresys::Open_eVision_2_10::EKernelRotation rotationMode,
 OEV_INT32& numberOfIterations
 )

void Thick(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 EKernel* thickeningKernel,
 Euresys::Open_eVision_2_10::EKernelRotation rotationMode,
 OEV_INT32& numberOfIterations
 )

OEV_INT32 Thick(
 EROIBW1* sourceImage,
 EROIBW1* destinationImage,
 EKernel* thickeningKernel,
 Euresys::Open_eVision_2_10::EKernelRotation rotationMode,
 OEV_INT32& numberOfIterations
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. Must not be the same as source image.

*thickeningKernel*

Pointer to the thickening kernel.

*rotationMode*

Rotation mode, as defined by EKernelRotation.

*numberOfIterations*

Number of iterations to apply. **0** indicates stop when convergence is reached. Upon return, gives the number of passes actually performed. If the rotation mode is set to either EKernelRotation_Clockwise or EKernelRotation_Anticlockwise, a pass comprises eight kernel rotations.

### Remarks

The thickening kernel coefficients must be **0** (matching black pixel, value 0), **1** (matching non black pixel, value > 0) or **-1** (don't care). When a match is found between the kernel coefficients and the neighborhood of a pixel, the pixel value is set to **255**.

# EasyImage::Thin

Applies a thinning operation on an image, using a 3x3 kernel.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Thin(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 EKernel* thinningKernel,
 Euresys::Open_eVision_2_10::EKernelRotation rotationMode,
 OEV_INT32& numberOfIterations
 )

void Thin(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 EKernel* thinningKernel,
 Euresys::Open_eVision_2_10::EKernelRotation rotationMode,
 OEV_INT32& numberOfIterations
 )

void Thin(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 EKernel* thinningKernel,
 Euresys::Open_eVision_2_10::EKernelRotation rotationMode,
 OEV_INT32& numberOfIterations
 )
```

```
OEV_INT32 Thin(
 EROIBW1* sourceImage,
 EROIBW1* destinationImage,
 EKernel* thinningKernel,
 Euresys::Open_eVision_2_10::EKernelRotation rotationMode,
 OEV_INT32& numberOfIterations
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. Must not be the same as source image.

*thinningKernel*

Pointer to the thinning kernel.

*rotationMode*

Rotation mode, as defined by EKernelRotation.

*numberOfIterations*

Number of iterations to apply. **0** indicates stop when convergence is reached. Upon return, gives the number of passes actually performed. If the rotation mode is set to either EKernelRotation_Clockwise or EKernelRotation_Anticlockwise, a pass comprises eight kernel rotations.

### Remarks

The thinning kernel coefficients must be **0** (matching black pixel, value 0), **1** (matching non black pixel, value > 0) or **-1** (don't care). When a match is found between the kernel coefficients and the neighborhood of a pixel, the pixel value is set to 0.

# EasyImage::ThreeLevelsMinResidueThreshold

Computes the two threshold values used to separate the pixels of an image in three classes.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
float ThreeLevelsMinResidueThreshold(
 EBWHistogramVector* histogram,
 EBW8& firstGrayPixelValue,
 EBW8& firstWhitePixelValue,
 float& averageBlack,
 float& averageGray,
 float& averageWhite
 )
```

### Parameters

*histogram*

Histogram of the image.

*firstGrayPixelValue*

Low threshold.

*firstWhitePixelValue*

High threshold.

*averageBlack*

Average value of the black pixels (pixels under the low threshold).

*averageGray*

Average value of the gray pixels (pixels between the low threshold and the high threshold).

*averageWhite*

Average value of the white pixels (pixels over the high threshold).

### Remarks

These values are computed using the minimum residue criterion from the histogram of the image. The function returns the minimum residue as per the method. The residue is the Euclidian distance between the source image and the thresholded image.

# EasyImage::Threshold

Binarize an image by setting pixels to two different possible values in a destination image, according to their value in a source image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Threshold(
 EROIBW8* sourceImage,
 EROIBW1* destinationImage,
 OEV_UINT32 threshold,
 float relativeThreshold
 )

void Threshold(
 EROIBW16* sourceImage,
 EROIBW1* destinationImage,
 OEV_UINT32 threshold,
 float relativeThreshold
 )

void Threshold(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 threshold,
 OEV_UINT8 lowValue,
 OEV_UINT8 highValue,
 float relativeThreshold
 )

void Threshold(
 const EROIBW8* sourceImage,
 const ERegion& region,
 EROIBW8* destinationImage,
 OEV_UINT32 threshold,
 OEV_UINT8 lowValue,
 OEV_UINT8 highValue,
 float relativeThreshold
 )

void Threshold(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage
 )

void Threshold(
 const EROIBW16* sourceImage,
 const ERegion& region,
 EROIBW16* destinationImage
 )
```

```
void Threshold(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 threshold
 )

void Threshold(
 const EROIBW16* sourceImage,
 const ERegion& region,
 EROIBW16* destinationImage,
 OEV_UINT32 threshold
 )

void Threshold(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 threshold,
 EBW16 lowValue,
 EBW16 highValue
 )

void Threshold(
 const EROIBW16* sourceImage,
 const ERegion& region,
 EROIBW16* destinationImage,
 OEV_UINT32 threshold,
 EBW16 lowValue,
 EBW16 highValue
 )

void Threshold(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 float relativeThreshold
 )

void Threshold(
 const EROIBW16* sourceImage,
 const ERegion& region,
 EROIBW16* destinationImage,
 float relativeThreshold
 )
```

```
void Threshold(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 float relativeThreshold,
 EBW16 lowValue,
 EBW16 highValue
 )

void Threshold(
 const EROIBW16* sourceImage,
 const ERegion& region,
 EROIBW16* destinationImage,
 float relativeThreshold,
 EBW16 lowValue,
 EBW16 highValue
 )

void Threshold(
 EROIC24* sourceImage,
 EROIBW8* destinationImage,
 EC24 minimum,
 EC24 maximum
 )

void Threshold(
 const EROIC24* sourceImage,
 const ERegion& region,
 EROIBW8* destinationImage,
 EC24 minimum,
 EC24 maximum
 )

void Threshold(
 EROIC24* sourceImage,
 EROIBW8* destinationImage,
 EC24 minimum,
 EC24 maximum,
 EColorLookup* colorLookupTable,
 EBW8 rejectValue,
 EBW8 acceptValue
 )
```

```
void Threshold(
 const EROIC24* sourceImage,
 const ERegion& region,
 EROIBW8* destinationImage,
 EC24 minimum,
 EC24 maximum,
 EColorLookup* colorLookupTable,
 EBW8 rejectValue,
 EBW8 acceptValue
 )
void Threshold(
 EROIC24* sourceImage,
 EROIBW8* destinationImage,
 EC24 minimum,
 EC24 maximum,
 EColorLookup* colorLookupTable
 )
void Threshold(
 const EROIC24* sourceImage,
 const ERegion& region,
 EROIBW8* destinationImage,
 EC24 minimum,
 EC24 maximum,
 EColorLookup* colorLookupTable
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*threshold*

The value to compare each pixel to

*relativeThreshold*

Fraction of the image pixels that will be set below the threshold. Only used when the threshold value is EThresholdMode_Relative (by default, **0.5**). This value must be greater than (or equal to) 0 and strictly less than 1.

*lowValue*

Value for pixels below the threshold (by default, **0**).

*highValue*

Value for pixels above the threshold (by default, it is set to **255** for BW8 destination images and **65535** for BW16 destination images).

*region*

Pointer to a region to apply the function only on a particular region in the image.

*minimum*

Three lower thresholds combined in a single color value.

*maximum*

Three upper thresholds combined in a single color value.

*colorLookupTable*

Pointer to the color lookup table to be applied before thresholding, if any.

*rejectValue*

Value for pixels falling outside the range (by default, **0**).

*acceptValue*

Value for pixels falling inside the range (by default, **255**).

### Remarks

When the source image is gray-level, the pixel values are measured against a threshold. All pixels below this threshold will yield a low value in the destination image, and all pixels above or on the threshold will yield a high value.
When the destination image is binary (BW1 pixel type), then the values are set to **0** or **1**, according to the criterion.
When the destination image is gray-level (BW8 or BW16), then the values are set to **0** or to the maximum pixel value for the image type (**255** for BW8 and **65535** for BW16). In some overloads, these minimum and maximum destination values can be specified.
When the source image is gray-level, several modes are available: absolute (the threshold value is given), relative (the threshold value is computed to obtain a desired fraction of the image pixels), or automatic (using three different criteria). In the function overloads where this mode cannot be specified, it is assumed to be absolute.
If the source image is color, all pixels whose components are comprised in a range of values (minimum to maximum) will be set to a constant value (white by default), while other pixels will be set to another constant value (black by default). In this case, if a color lookup is specified, it is applied on the fly to the color image before thresholding.
The simpler color image overload does not support the use of an on-the-fly color lookup table nor **rejectValue**/**acceptValue** arguments. On the other hand, it has been MMX optimized, and will run significantly faster when the acceptance region is large.

# EasyImage::TwoLevelsMinResidueThreshold

Computes the threshold value used to separate the pixels of an image in two classes.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float TwoLevelsMinResidueThreshold(
 EBWHistogramVector* histogram,
 EBW8& firstWhitePixelValue,
 float& averageBlack,
 float& averageWhite
 )
```

### Parameters

*histogram*

Histogram of the image.

*firstWhitePixelValue*

Threshold.

*averageBlack*

Average value of the black pixels (pixels under the threshold).

*averageWhite*

Average value of the white pixels (pixels over the threshold).

### Remarks

This value is computed using the minimum residue criterion from the histogram of the image. The function returns the minimum residue as per the method. The residue is the Euclidian distance between the source image and the thresholded image.

# EasyImage::Uniformize

Shading correction is the process of transforming the gray or color component values of an image, using one or two reference images or vectors.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Uniformize(
 EROIBW8* sourceImage,
 EBW8 pixelReference,
 EROIBW8* imageReference,
 EROIBW8* destinationImage,
 BOOL multiplicative
 )

void Uniformize(
 EROIBW16* sourceImage,
 EBW16 pixelReference,
 EROIBW16* imageReference,
 EROIBW16* destinationImage,
 BOOL multiplicative
 )

void Uniformize(
 EROIC24* sourceImage,
 EC24 pixelReference,
 EROIC24* imageReference,
 EROIC24* destinationImage,
 BOOL multiplicative
 )

void Uniformize(
 EROIBW8* sourceImage,
 EBW8 pixelReference,
 EBW8Vector* vectorOfPixelReference,
 EROIBW8* destinationImage,
 BOOL multiplicative
 )

void Uniformize(
 EROIBW16* sourceImage,
 EBW16 pixelReference,
 EBW16Vector* vectorOfPixelReference,
 EROIBW16* destinationImage,
 BOOL multiplicative
 )
```

```
void Uniformize(
 EROIC24* sourceImage,
 EC24 pixelReference,
 EC24Vector* vectorOfPixelReference,
 EROIC24* destinationImage,
 BOOL multiplicative
 )

void Uniformize(
 EROIBW8* sourceImage,
 EBW8 pixelLightReference,
 EROIBW8* imageLightReference,
 EBW8 pixelDarkReference,
 EROIBW8* imageDarkReference,
 EROIBW8* destinationImage
 )

void Uniformize(
 EROIBW16* sourceImage,
 EBW16 pixelLightReference,
 EROIBW16* imageLightReference,
 EBW16 pixelDarkReference,
 EROIBW16* imageDarkReference,
 EROIBW16* destinationImage
 )

void Uniformize(
 EROIC24* sourceImage,
 EC24 pixelLightReference,
 EROIC24* imageLightReference,
 EC24 pixelDarkReference,
 EROIC24* imageDarkReference,
 EROIC24* destinationImage
 )

void Uniformize(
 EROIBW8* sourceImage,
 EBW8 pixelLightReference,
 EBW8Vector* vectorOfPixelLightReference,
 EBW8 pixelDarkReference,
 EBW8Vector* vectorOfPixelDarkReference,
 EROIBW8* destinationImage
 )
```

```
void Uniformize(
 EROIBW16* sourceImage,
 EBW16 pixelLightReference,
 EBW16Vector* vectorOfPixelLightReference,
 EBW16 pixelDarkReference,
 EBW16Vector* vectorOfPixelDarkReference,
 EROIBW16* destinationImage
 )

void Uniformize(
 EROIC24* sourceImage,
 EC24 pixelLightReference,
 EC24Vector* vectorOfPixelLightReference,
 EC24 pixelDarkReference,
 EC24Vector* vectorOfPixelDarkReference,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*pixelReference*

Constant value to transform the reference image or vector into.

*imageReference*

Pointer to the reference source image/ROI or vector.

*destinationImage*

Pointer to the destination image/ROI.

*multiplicative*

**TRUE**, if the transform is multiplicative (gain); **FALSE**, if the transform is additive (offset) (by default, **TRUE**).

*vectorOfPixelReference*

Constant value to transform the reference image or vector into.

*pixelLightReference*

Constant value to transform the light reference image or vector into.

*imageLightReference*

Pointer to the light reference source image/ROI or vector.

*pixelDarkReference*

Constant value to transform the dark reference image/ROI or vector into.

*imageDarkReference*

Pointer to the dark reference source image/ROI or vector.

*vectorOfPixelLightReference*

Constant value to transform the light reference image or vector into.

*vectorOfPixelDarkReference*

Constant value to transform the dark reference image/ROI or vector into.

### Remarks

The intent is to compensate for non-uniform lighting or sensor response non-uniformity by providing images of the background with no foreground object present.
In the case of area-scan cameras, the illumination can change anywhere in the field of view, requiring 2D compensation. In the case of line-scan cameras imaging moving parts, illumination remains constant across image rows. Only 1D compensation is required. In this case, the reference illumination is specified as a vector, which is replicated across all image rows.
* When a single reference image is used, the transform is analog to an adaptive (space-variant) gain *or* offset (Gain * Intensity or Intensity + Offset); the transform lets the reference image(s) become a specified constant value, i.e. flat field illumination.
* When two reference images are used, the transform is analog to adaptive gain *and* offset (Gain * Intensity + Offset); the transform let both reference images become specified constants, i.e. flat field illumination with a correct black reference.

**Note.** The reference image(s) should be chosen such that they contain no saturated pixel values (remain in the linear domain) and little (filtered out) noise.

# EasyImage::VerticalMirror

Mirrors an image vertically (the rows are swapped).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void VerticalMirror(
 EROIBW8* sourceImage
 )
void VerticalMirror(
 EROIBW16* sourceImage
 )
```

```
void VerticalMirror(
 EROIC24* sourceImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

# EasyImage::Warp

Transforms an image so that each pixels are moved to the locations specified in the "warp" images used as look-up tables.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Warp(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 EImageBW16* warpImageX,
 EImageBW16* warpImageY,
 OEV_INT32 shiftX,
 OEV_INT32 shiftY
 )
void Warp(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 EImageBW16* warpImageX,
 EImageBW16* warpImageY,
 OEV_INT32 shiftX,
 OEV_INT32 shiftY
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI.

*warpImageX*

Pointer to the X lookup image.

*warpImageY*

Pointer to the Y lookup image.

*shiftX*

Horizontal translation.

*shiftY*

Vertical translation.

### Remarks

For example, pixel **[10,20]** moves to location **[WarpXImage[10,20], WarpYImage[10,20]]**.

# EasyImage::WeightedMoments

Computes the zero-th, first, second, third or fourth order weighted moments on the gray-level image. The weight of a pixel is its gray-level value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void WeightedMoments(
 const EROIBW8* sourceImage,
 float& M,
 float& Mx,
 float& My
 )
void WeightedMoments(
 const EROIBW16* sourceImage,
 float& M,
 float& Mx,
 float& My
 )
```

```
void WeightedMoments(
 const EROIBW8* sourceImage,
 const ERegion& region,
 float& M,
 float& Mx,
 float& My
 )

void WeightedMoments(
 const EROIBW16* sourceImage,
 const ERegion& region,
 float& M,
 float& Mx,
 float& My
 )

void WeightedMoments(
 const EROIBW8* sourceImage,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy
 )

void WeightedMoments(
 const EROIBW16* sourceImage,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy
 )
```

```
void WeightedMoments(
 const EROIBW8* sourceImage,
 const ERegion& region,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy
 )

void WeightedMoments(
 const EROIBW16* sourceImage,
 const ERegion& region,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy
 )

void WeightedMoments(
 const EROIBW8* sourceImage,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy,
 float& Mxxx,
 float& Mxxy,
 float& Mxyy,
 float& Myyy
 )
```

```
void WeightedMoments(
 const EROIBW16* sourceImage,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy,
 float& Mxxx,
 float& Mxxy,
 float& Mxyy,
 float& Myyy
 )

void WeightedMoments(
 const EROIBW8* sourceImage,
 const ERegion& region,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy,
 float& Mxxx,
 float& Mxxy,
 float& Mxyy,
 float& Myyy
 )

void WeightedMoments(
 const EROIBW16* sourceImage,
 const ERegion& region,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy,
 float& Mxxx,
 float& Mxxy,
 float& Mxyy,
 float& Myyy
 )
```

```
void WeightedMoments(
 const EROIBW8* sourceImage,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy,
 float& Mxxx,
 float& Mxxy,
 float& Mxyy,
 float& Myyy,
 float& Mxxxx,
 float& Mxxxy,
 float& Mxxyy,
 float& Mxyyy,
 float& Myyyy
 )

void WeightedMoments(
 const EROIBW16* sourceImage,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy,
 float& Mxxx,
 float& Mxxy,
 float& Mxyy,
 float& Myyy,
 float& Mxxxx,
 float& Mxxxy,
 float& Mxxyy,
 float& Mxyyy,
 float& Myyyy
 )
```

```
void WeightedMoments(
 const EROIBW8* sourceImage,
 const ERegion& region,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy,
 float& Mxxx,
 float& Mxxy,
 float& Mxyy,
 float& Myyy,
 float& Mxxxx,
 float& Mxxxy,
 float& Mxxyy,
 float& Mxyyy,
 float& Myyyy
 )
void WeightedMoments(
 const EROIBW16* sourceImage,
 const ERegion& region,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy,
 float& Mxxx,
 float& Mxxy,
 float& Mxyy,
 float& Myyy,
 float& Mxxxx,
 float& Mxxxy,
 float& Mxxyy,
 float& Mxyyy,
 float& Myyyy
 )
```

```
void WeightedMoments(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 float& M,
 float& Mx,
 float& My
 )

void WeightedMoments(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 float& M,
 float& Mx,
 float& My
 )

void WeightedMoments(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy
 )

void WeightedMoments(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy
 )
```

```
void WeightedMoments(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy,
 float& Mxxx,
 float& Mxxy,
 float& Mxyy,
 float& Myyy
 )
void WeightedMoments(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy,
 float& Mxxx,
 float& Mxxy,
 float& Mxyy,
 float& Myyy
 )
```

```
void WeightedMoments(
 const EROIBW8* sourceImage,
 const EROIBW8* mask,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy,
 float& Mxxx,
 float& Mxxy,
 float& Mxyy,
 float& Myyy,
 float& Mxxxx,
 float& Mxxxy,
 float& Mxxyy,
 float& Mxyyy,
 float& Myyyy
 )
void WeightedMoments(
 const EROIBW16* sourceImage,
 const EROIBW8* mask,
 float& M,
 float& Mx,
 float& My,
 float& Mxx,
 float& Mxy,
 float& Myy,
 float& Mxxx,
 float& Mxxy,
 float& Mxyy,
 float& Myyy,
 float& Mxxxx,
 float& Mxxxy,
 float& Mxxyy,
 float& Mxyyy,
 float& Myyyy
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*M*

Reference to the zero-th order weighted moment (total gray value).

*Mx*

Reference to the first order moments (weighted sums of abscissas and ordinates).

*My*

Reference to the first order moments (weighted sums of abscissas and ordinates).

*region*

Pointer to a region to apply the function only on a particular region in the image.

*Mxx*

Reference to the second order uncentered moments (weighted sums of squared abscissas, cross-product and squared ordinates).

*Mxy*

Reference to the second order uncentered moments (weighted sums of squared abscissas, cross-product and squared ordinates).

*Myy*

Reference to the second order uncentered moments (weighted sums of squared abscissas, cross-product and squared ordinates).

*Mxxx*

Reference to the third order uncentered moments (weighted sums of third order products).

*Mxxy*

Reference to the third order uncentered moments (weighted sums of third order products).

*Mxyy*

Reference to the third order uncentered moments (weighted sums of third order products).

*Myyy*

Reference to the third order uncentered moments (weighted sums of third order products).

*Mxxxx*

Reference to the fourth order uncentered moments (weighted sums of fourth order products).

*Mxxxy*

Reference to the fourth order uncentered moments (weighted sums of fourth order products).

*Mxxyy*

Reference to the fourth order uncentered moments (weighted sums of fourth order products).

*Mxyyy*

Reference to the fourth order uncentered moments (weighted sums of fourth order products).

*Myyyy*

Reference to the fourth order uncentered moments (weighted sums of fourth order products).

*mask*

Pointer to a mask to apply the function only on a particular region in the image. Note: the mask must have the same size as the source image.

# EasyImage::WhiteTopHatBox

Performs a top-hat filtering on an image (source image minus opened image) on a rectangular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void WhiteTopHatBox(
 EROIBW1* sourceImage,
 EROIBW1* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
void WhiteTopHatBox(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
void WhiteTopHatBox(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
void WhiteTopHatBox(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_UINT32 halfOfKernelWidth,
 OEV_UINT32 halfOfKernelHeight
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. Must not be the same as the source image.

*halfOfKernelWidth*

Half of the box width minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

*halfOfKernelHeight*

Half of the box height minus one (by default, same as **halfOfKernelWidth**; **0** is allowed).

### Remarks

This filter enhances the thin white features.

# EasyImage::WhiteTopHatDisk

Performs a top-hat filtering on an image (source image minus opened image) on a quasi-circular kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void WhiteTopHatDisk(
 EROIBW1* sourceImage,
 EROIBW1* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
void WhiteTopHatDisk(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
void WhiteTopHatDisk(
 EROIBW16* sourceImage,
 EROIBW16* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
void WhiteTopHatDisk(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 OEV_UINT32 halfOfKernelWidth
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination image/ROI. Must not be the same as the source image.

*halfOfKernelWidth*

Half width of the kernel minus one (by default, **halfOfKernelWidth = 1**; **0** is allowed).

### Remarks

This filter enhances the thin white features.

# 4.15. EasyObject Class

This class contains static properties and methods specific to the EasyObject library.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| ContourArea | Computes the area of an object from its contour. |
| ContourGravityCenter | Computes the area and gravity center of an object from its contour. |
| ContourInertia | Computes the inertia parameters of an object from its contour. |
| IsFloatFeature | Tests whether a given feature is associated with floating-point values. |
| IsIntegerFeature | Tests whether a given feature is associated with integer values. |

| | |
|---|---|
| IsUnsignedIntegerFeature | Tests whether a given feature is associated with unsigned integer values. |

a

# syObject::ContourArea

Computes the area of an object from its contour.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ContourArea(
 EPathVector* pPathVector,
 OEV_INT32& n32Area
 )
```

### Parameters

*pPathVector*

Pointer to the source vector.

*n32Area*

Reference to the area to compute.

# EasyObject::ContourGravityCenter

Computes the area and gravity center of an object from its contour.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void ContourGravityCenter(
 EPathVector* pPathVector,
 OEV_INT32& n32Area,
 float& f32GravityCenterX,
 float& f32GravityCenterY
 )
```

### Parameters

*pPathVector*

Pointer to the source vector.

*n32Area*

Reference to the area to compute.

*f32GravityCenterX*

Reference to the abscissa of the gravity center to compute.

*f32GravityCenterY*

Reference to the ordinate of the gravity center to compute.

# EasyObject::ContourInertia

Computes the inertia parameters of an object from its contour.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ContourInertia(
 EPathVector* pPathVector,
 OEV_INT32& n32Area,
 float& f32GravityCenterX,
 float& f32GravityCenterY,
 float& f32SigmaX,
 float& f32SigmaY,
 float& f32SigmaXY
 )
```

### Parameters

*pPathVector*

Pointer to the source vector.

*n32Area*

Reference to the area to compute.

*f32GravityCenterX*

Reference to the abscissa of the gravity center to compute.

*f32GravityCenterY*

Reference to the ordinate of the gravity center to compute.

*f32SigmaX*

Centered cross moment of inertia.

*f32SigmaY*

Centered moment of inertia around Y.

*f32SigmaXY*

Centered cross moment of inertia.

# EasyObject::IsFloatFeature

Tests whether a given feature is associated with floating-point values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool IsFloatFeature(
 Euresys::Open_eVision_2_10::EFeature feature
 )
```

### Parameters

*feature*

The feature.

### Remarks

Most features are floating-point. The exceptions are listed in these functions: EasyObject::IsUnsignedIntegerFeature and EasyObject::IsIntegerFeature.

# EasyObject::IsIntegerFeature

Tests whether a given feature is associated with integer values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool IsIntegerFeature(
 Euresys::Open_eVision_2_10::EFeature feature
 )
```

### Parameters

*feature*

The feature.

### Remarks

The features associated with integer values are: EFeature_ContourX, EFeature_ContourY, EFeature_LeftLimit, EFeature_RightLimit, EFeature_TopLimit, and EFeature_BottomLimit.

# EasyObject::IsUnsignedIntegerFeature

Tests whether a given feature is associated with unsigned integer values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool IsUnsignedIntegerFeature(
 Euresys::Open_eVision_2_10::EFeature feature
 )
```

### Parameters

*feature*

The feature.

### Remarks

The features associated with unsigned integer values are: EFeature_ElementIndex, EFeature_LayerIndex, EFeature_RunCount, EFeature_Area and EFeature_LargestRun.

# 4.16. EBarCode Class

Manages a complete context for the reading or verification of bar codes in EasyBarCode.

**Base Class:** ERectangleShape

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Decode | Provides the decoded information (or a reading error code) corresponding to the specified symbology. |
| Detect | Processes the image, reads the possible contents of the bar code corresponding to all enabled symbologies and rates their likeliness. |
| Drag | Moves a handle to a new position and updates the position parameters of the symbol bounding box. |
| Draw | Draws the symbol bounding box. |
| DrawWithCurrentPen | Draws the symbol bounding box. |
| EBarCode | Creates an EBarCode object. |

| | |
|---|---|
| GetAdditionalSymbologies | Enabled symbologies belonging to the group of additional symbologies. |
| GetDecodedAngle | Allows retrieving the bar code reading angle, pertaining to the specified symbology, or, at least, to the most likely symbology. |
| GetDecodedDirection | Returns the encoding direction of the bar code, pertaining to the specified symbology, or, at least, to the most likely symbology. |
| GetDecodedRectangle | Allows retrieving the bar code reading rectangle, pertaining to the specified symbology, or, at least, to the most likely symbology. |
| GetDecodedSymbology | Returns the identifier of one of the symbologies that were successfully decoded. |
| GetKnownLocation | Flag indicating whether the symbol location is known or not. |
| GetKnownModule | Flag indicating whether the symbol module is known or not. |
| GetModule | Module value. |
| GetNumDecodedSymbologies | Number of symbologies (among the enabled ones) for which the decoding process was successful. |
| GetNumEnabledSymbologies | Number of enabled symbologies. |

| | |
|---|---|
| GetRelativeReadingSizeX | Reading area width, relative to the symbol extent. |
| GetRelativeReadingSizeY | Reading area height, relative to the symbol extent. |
| GetRelativeReadingX | Reading area abscissa, relative to the symbol position. |
| GetRelativeReadingY | Reading area ordinate, relative to the symbol position. |
| GetStandardSymbologies | Enabled symbologies belonging to the group of standard symbologies. |
| GetSymbologyName | Retrieves the name of given symbology as a string. |
| GetThicknessRatio | Bars thickness ratio. |
| GetVerifyChecksum | "VerifyChecksum" mode. |
| HitTest | Checks whether the cursor is positioned over a handle (**TRUE**) or not (**FALSE**). |
| Read | Processes the image, reads the possible contents of the bar code and returns the single possible decoding (mono-symbology) or the most likely one (multi-symbology) or a reading error code. |

| | |
|---|---|
| SetAdditionalSymbologies | Enabled symbologies belonging to the group of additional symbologies. |
| SetKnownLocation | Flag indicating whether the symbol location is known or not. |
| SetKnownModule | Flag indicating whether the symbol module is known or not. |
| SetModule | Module value. |
| SetReadingCenter | Sets the reading area center coordinates, relative to the symbol position and extent. |
| SetReadingSize | Sets the reading area size, relative to the symbol extent. |
| SetRectangle | Sets the nominal position (center coordinates), size and rotation angle of the symbol bounding box according to a known rectangle. |
| SetStandardSymbologies | Enabled symbologies belonging to the group of standard symbologies. |
| SetThicknessRatio | Bars thickness ratio. |
| SetVerifyChecksum | "VerifyChecksum" mode. |

# EBarCode::GetAdditionalSymbologies

# EBarCode::SetAdditionalSymbologies

Enabled symbologies belonging to the group of additional symbologies.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetAdditionalSymbologies()
void SetAdditionalSymbologies(OEV_UINT32 un32AdditionalSymbologies)
```

## Remarks

Due to the large number of supported symbologies, they have been gathered in two groups. For more information about these groups, see ESymbologies.

# EBarCode::Decode

Provides the decoded information (or a reading error code) corresponding to the specified symbology.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::string Decode(
 Euresys::Open_eVision_2_10::ESymbologies symbology
 )
```

## Parameters

*symbology*

Specified symbology, as defined by ESymbologies this symbology must have been enabled).

### Remarks

Before calling EBarCode::Decode, an EBarCode::Detect operation must have been performed. In case of the mono-symbology mode, or if only the most likely decoding matters, the EBarCode::Read method should be used.

# EBarCode::Detect

Processes the image, reads the possible contents of the bar code corresponding to all enabled symbologies and rates their likeliness.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Detect(
 EROIBW8* sourceImage
 )
```

### Parameters

*sourceImage*

Pointer to the image containing the bar code.

### Remarks

Processing the image means finding the symbol (in case of automatic location mode) and retrieving its descriptive parameters. The decoded information corresponding to a specific symbology is provided by the decode function. The symbologies that were successfully decoded are ranked by decreasing likeliness (range **0** to **NumDecodedSymbologies-1**). In case of the mono-symbology mode or if only the most likely decoding matters, the **Read** function should be used.

# EBarCode::Drag

Moves a handle to a new position and updates the position parameters of the symbol bounding box.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 OEV_INT32 cursorX,
 OEV_INT32 cursorY
 )
```

### Parameters

*cursorX*

Cursor current coordinates.

*cursorY*

Cursor current coordinates.

# EBarCode::Draw

Draws the symbol bounding box.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
void Draw(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the symbol bounding box must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the shapes attached to the symbol bounding box are to be displayed as well.

*color*

The color in which to draw the overlay.

### Remarks

The bounding box corresponds to the nominal position of the bar code (EDrawingMode_Nominal), in case this information has been explicitly provided, and to the actual position (EDrawingMode_Actual) if it has been determined by image analysis.

# EBarCode::DrawWithCurrentPen

Draws the symbol bounding box.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the symbol bounding box must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the shapes attached to the symbol bounding box are to be displayed as well.

### Remarks

The bounding box corresponds to the nominal position of the bar code (EDrawingMode_Nominal), in case this information has been explicitly provided, and to the actual position (EDrawingMode_Actual) if it has been determined by image analysis.

# EBarCode::EBarCode

Creates an EBarCode object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EBarCode(
 )
void EBarCode(
 const EBarCode& other
 )
```

### Parameters

*other*

-

# EBarCode::GetDecodedAngle

Allows retrieving the bar code reading angle, pertaining to the specified symbology, or, at least, to the most likely symbology.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void GetDecodedAngle(
 float& decodedAngle
 )

void GetDecodedAngle(
 float& decodedAngle,
 float cutAngle
 )

void GetDecodedAngle(
 Euresys::Open_eVision_2_10::ESymbologies symbology,
 float& decodedAngle
 )

void GetDecodedAngle(
 Euresys::Open_eVision_2_10::ESymbologies symbology,
 float& decodedAngle,
 float cutAngle
 )
```

### Parameters

*decodedAngle*

Returned bar code reading angle value.

*cutAngle*

Cut angle value (°) defining the allowed range for the bar code reading angle (**[cutAngle, cutAngle + 360]**). By default, the cut angle equals **-45**.

*symbology*

Specified symbology, as defined by ESymbologies (this symbology must have been enabled).

# EBarCode::GetDecodedDirection

Returns the encoding direction of the bar code, pertaining to the specified symbology, or, at least, to the most likely symbology.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetDecodedDirection(
 BOOL& directEncoding
 )

void GetDecodedDirection(
 Euresys::Open_eVision_2_10::ESymbologies symbology,
 BOOL& directEncoding
 )
```

### Parameters

*directEncoding*

Boolean holding the encoding direction status.

*symbology*

Specified symbology, as defined by ESymbologies (this symbology must have been enabled).

### Remarks

The encoding direction of the bar code is "Direct" or "Inverse". The encoding direction is said to be "Direct" when the bar code longitudinal axis falls in the range [-45°, 135]. Conversely, when the bar code longitudinal axis doesn't fall in the previous range, the encoding direction is said to be "Inverse".

# EBarCode::GetDecodedRectangle

Allows retrieving the bar code reading rectangle, pertaining to the specified symbology, or, at least, to the most likely symbology.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetDecodedRectangle(
 ERectangle& rect
 )
```

```
void GetDecodedRectangle(
 Euresys::Open_eVision_2_10::ESymbologies symbology,
 ERectangle& rect
 )
```

### Parameters

*rect*

Returned bar code reading rectangle.

*symbology*

Specified symbology, as defined by ESymbologies (this symbology must have been enabled).

# EBarCode::GetDecodedSymbology

Returns the identifier of one of the symbologies that were successfully decoded.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::ESymbologies GetDecodedSymbology(
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index of the specified symbology (range **0** to **NumDecodedSymbologies-1**).

### Remarks

The desired symbology is specified by its ranking index (range **0** to **NumDecodedSymbologies-1**). The symbologies that were successfully decoded are ranked by decreasing likeliness.

# EBarCode::GetSymbologyName

Retrieves the name of given symbology as a string.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::string GetSymbologyName(
 Euresys::Open_eVision_2_10::ESymbologies symbology
 )
```

## Parameters

*symbology*

Symbology.

# EBarCode::HitTest

Checks whether the cursor is positioned over a handle (**TRUE**) or not (**FALSE**).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL HitTest(
 BOOL daughters
 )
```

## Parameters

*daughters*

**TRUE** if the handles of the shapes attached to the symbol bounding box have to be considered as well.

# EBarCode::GetKnownLocation

# EBarCode::SetKnownLocation

Flag indicating whether the symbol location is known or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetKnownLocation()
void SetKnownLocation(BOOL bKnownLocation)
```

## Remarks

In case of known location, use EBarCode::Rectangle to adjust a rectangle around the symbol.

# EBarCode::GetKnownModule

# EBarCode::SetKnownModule

Flag indicating whether the symbol module is known or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetKnownModule()
void SetKnownModule(BOOL bKnownModule)
```

### Remarks

If **TRUE**, it is also necessary to get the EBarCode::Module and EBarCode::ThicknessRatio properties in order to specify the requested module and thickness ratio.

# EBarCode::GetModule

# EBarCode::SetModule

Module value.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetModule()
void SetModule(float f32Module)
```

### Remarks

The module value is a descriptive parameter participating in the encoding; it corresponds to the thinner bar width. Symbols whose bars thickness can take two values, the module and **V** times the module (where **V** runs from **1.5** to **3**), are called *binary* bar codes. When the bars thickness are small integer multiples (1 to 4 or 5) of a module, the symbols are called *modular* bar codes.

# EBarCode::GetNumDecodedSymbologies

Number of symbologies (among the enabled ones) for which the decoding process was successful.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumDecodedSymbologies()
```

# EBarCode::GetNumEnabledSymbologies

Number of enabled symbologies.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumEnabledSymbologies()
```

# EBarCode::Read

Processes the image, reads the possible contents of the bar code and returns the single possible decoding (mono-symbology) or the most likely one (multi-symbology) or a reading error code.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::string Read(
 EROIBW8* sourceImage
 )
```

## Parameters

*sourceImage*

The image containing the bar code.

### Remarks

Processing the image means finding the symbol (in case of automatic location mode) and retrieving its descriptive parameters. When decoding other than the most likely one are also required, a call to the EBarCode::Detect function followed by a EBarCode::Decode should be used.

# EBarCode::SetRectangle

Sets the nominal position (center coordinates), size and rotation angle of the symbol bounding box according to a known rectangle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetRectangle(const ERectangle& rectangle)
```

### Remarks

An ERectangle object is characterized by its center coordinates, its size and its rotation angle.

# EBarCode::GetRelativeReadingSizeX

Reading area width, relative to the symbol extent.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetRelativeReadingSizeX()
```

# EBarCode::GetRelativeReadingSizeY

Reading area height, relative to the symbol extent.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetRelativeReadingSizeY()
```

# EBarCode::GetRelativeReadingX

Reading area abscissa, relative to the symbol position.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetRelativeReadingX()
```

# EBarCode::GetRelativeReadingY

Reading area ordinate, relative to the symbol position.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetRelativeReadingY()
```

# EBarCode::SetReadingCenter

Sets the reading area center coordinates, relative to the symbol position and extent.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetReadingCenter(
 float relativeX,
 float relativeY
 )
```

### Parameters

*relativeX*

Reading area center abscissa, relative to the symbol position and extent.

*relativeY*

Reading area center ordinate, relative to the symbol position and extent.

# EBarCode::SetReadingSize

Sets the reading area size, relative to the symbol extent.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void SetReadingSize(
 float relativeSizeX,
 float relativeSizeY
 )
```

### Parameters

*relativeSizeX*

Reading area width, relative to the symbol extent.

*relativeSizeY*

Reading area height, relative to the symbol extent.

# EBarCode::GetStandardSymbologies

# EBarCode::SetStandardSymbologies

Enabled symbologies belonging to the group of standard symbologies.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetStandardSymbologies()
void SetStandardSymbologies(OEV_UINT32 un32StandardSymbologies)
```

### Remarks

Due to the large number of supported symbologies, they have been gathered in two groups. For more information about these groups, see ESymbologies.

# EBarCode::GetThicknessRatio

# EBarCode::SetThicknessRatio

Bars thickness ratio.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetThicknessRatio()
void SetThicknessRatio(float f32ThicknessRatio)
```

## Remarks

This property is relevant in case of binary codes only. It corresponds to the ratio of a thin bar width over a thick bar width, and should range from **1.5** to **3**.

# EBarCode::GetVerifyChecksum

# EBarCode::SetVerifyChecksum

"VerifyChecksum" mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetVerifyChecksum()
void SetVerifyChecksum(BOOL bVerifyChecksum)
```

### Remarks

The "VerifyChecksum" mode enables or disables verification of the checksum character. That verification mode is set in the same way for all enabled symbologies. It is worth noting that checksum may be present or not in the bar code, and the user may verify or not checksum validity. These two circumstances are independent, and give rise to four modes of operation. The verification process will return an "invalid checksum" error in case of bad checksum character. This error can also be generated if there is no checksum in the bar code. In the other case, when checksum are not verified, no error will occur, and the process will continue silently. When the "VerifyChecksum" mode is enabled, the returned decoded string does not contain the checksum character(s). Conversely, when the verification process is disabled, the checksum character(s) are concatenated to the encoded information.

# 4.17. EBaseROI Class

This represents the abstract base class for all ROI and image classes.

**Derived Class(es):** EROIC24 EROIBW8 EROIBW16 EROIBW32 EROIC15 EROIC16 EROIC24A EROIBW1 EROIC48

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Attach | This method attaches the ROI to another ROI or image.<br>Only ROIs can be attached. An image can never be attached to another image or ROI.<br>Optionally, the ROI can be moved and resized after attachment by supplying additional parameters. |
| CopyTo | Copies all the data of the current EBaseROI object into another EBaseROI object and returns it. |

| | |
|---|---|
| CropToImage | This method reduces the ROI size so that it is completely contained within its topmost parent image bounds (if such an image exists at the top of the hierarchy). This means that, after calling this method, either the ROI has a zero size, or all of its pixels map to valid pixels of the underlying image buffer. If the ROI is already contained within its parent image bounds, then this method does nothing. |
| Drag | Moves the specified handle to a new position and updates all placement parameters of the ROI. |
| Draw | Draws an ROI/image in a device context. |
| DrawFrame | Draws a rectangular frame around an image or ROI. |
| DrawFrameWithCurrentPen | Draws a rectangular frame around an image or ROI. |
| GetAuthor | Gets or sets the Author attribute of the ROI/image. This is a convenience property that allows storing some information in the ROI/image. Please note that not all file formats preserve this information. |
| GetBaseTopParent | Returns the image at the top of the hierarchy, or NULL if there is no image on top. |
| GetBitsPerPixel | Gets the number of storage bits per pixel. |
| GetColorSystem | Gets or sets the color system used by this image, as defined by the EColorSystem enumeration. |

| | |
|---|---|
| GetColPitch | Gets the pitch of a column (number of bytes between two horizontally adjacent pixels). For BW1 (one bit per pixel) images, this value is undefined. |
| GetComment | Gets or sets the Comment attribute of the ROI/image. This is a convenience property that allows storing some information in the ROI/image. Please note that not all file formats preserve this information. |
| GetDate | Gets or sets the Date attribute of the ROI/image. This is a convenience property that allows storing some information in the ROI/image. Please note that not all file formats preserve this information. |
| GetHeight | Gets or sets the height of the ROI. |
| GetImagePtr | Returns a pointer to the pixel at given coordinates within the image/ROI. |
| GetOrgX | Gets or sets the x-coordinate of the upper left corner of the ROI, relative to its parent ROI/image. |
| GetOrgY | Gets or sets the y-coordinate of the upper left corner of the ROI, relative to its parent ROI/image. |
| GetParent | Returns the hierarchical parent of this object. |

| | |
|---|---|
| GetPlanesPerPixel | Gets the number of color components in each pixel of the ROI/image. |
| GetRowPitch | Gets the pitch of a row (the number of bytes between two vertically adjacent pixels). This is also valid for BW1 images (one bit per pixel). |
| GetSubBaseROIs | Returns all the children, and possibly recursively all their children, too. |
| GetTitle | Gets or sets the Title attribute of the ROI/image. This is a convenience property that allows storing some information in the ROI/image. Please note that not all file formats preserve this information. |
| GetTotalHeight | Gets the height, in pixels, of the ROI topmost parent. |
| GetTotalOrgX | Gets the x-coordinate of the upper left pixel of the ROI, relative to its topmost parent upper left pixel. |
| GetTotalOrgY | Gets the y-coordinate of the upper left pixel of the ROI, relative to its topmost parent upper left pixel. |
| GetTotalWidth | Gets the width, in pixels, of the ROI topmost parent. |

| | |
|---|---|
| GetType | Gets the ROI/image pixel type, as defined by the EImageType enumeration. |
| GetWidth | Gets or sets the width of the ROI. |
| HasSubROI | Tests whether this object has a given attached ROI. |
| HitTest | Detects if the cursor is placed over one of the dragging handles. |
| IsAnROI | Tests whether this object is an ROI or an image. |
| IsVoid | Tests whether if the topmost parent image of this hierarchy has a zero size. |
| Load | Restores an image stored in the given file. |
| Save | Saves the EBaseROI object to the given file. |
| SaveJpeg | Saves the EBaseROI object to the given file, in JPEG format. |
| SaveJpeg2K | Saves the EBaseROI object to the given file, in JPEG 2000 format. |
| SavePng | Saves the EBaseROI object to the given file, in PNG format. |

| | |
|---|---|
| Serialize | Serializes the EBaseROI. |
| SetAuthor | Gets or sets the Author attribute of the ROI/image. This is a convenience property that allows storing some information in the ROI/image. Please note that not all file formats preserve this information. |
| SetColorSystem | Gets or sets the color system used by this image, as defined by the EColorSystem enumeration. |
| SetComment | Gets or sets the Comment attribute of the ROI/image. This is a convenience property that allows storing some information in the ROI/image. Please note that not all file formats preserve this information. |
| SetDate | Gets or sets the Date attribute of the ROI/image. This is a convenience property that allows storing some information in the ROI/image. Please note that not all file formats preserve this information. |
| SetHeight | Gets or sets the height of the ROI. |
| SetImagePtr | Sets the pointer to an externally allocated image buffer. |
| SetOrgX | Gets or sets the x-coordinate of the upper left corner of the ROI, relative to its parent ROI/image. |

| | |
|---|---|
| SetOrgY | Gets or sets the y-coordinate of the upper left corner of the ROI, relative to its parent ROI/image. |
| SetPlacement | Sets the placement of an ROI, relative to its parent ROI/image. |
| SetSize | Sets the width and height of an ROI/image. |
| SetTitle | Gets or sets the Title attribute of the ROI/image. This is a convenience property that allows storing some information in the ROI/image. Please note that not all file formats preserve this information. |
| SetWidth | Gets or sets the width of the ROI. |

E

B

# aseROI::Attach

This method attaches the ROI to another ROI or image.
Only ROIs can be attached. An image can never be attached to another image or ROI.
Optionally, the ROI can be moved and resized after attachment by supplying additional parameters.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Attach(
 EBaseROI* parent
 )
```

```
void Attach(
 EBaseROI* parent,
 int orgX,
 int orgY,
 int width,
 int height
 )
```

### Parameters

*parent*

ROI or image on which to attache the ROI.

*orgX*

When specified, sets the new x-coordinate of the ROI top-left corner.

*orgY*

When specified, sets the new y-coordinate of the ROI top-left corner.

*width*

When specified, sets the new width of the ROI.

*height*

When specified, sets the new height of the ROI.

# EBaseROI::GetAuthor

# EBaseROI::SetAuthor

Gets or sets the Author attribute of the ROI/image. This is a convenience property that allows storing some information in the ROI/image. Please note that not all file formats preserve this information.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::string GetAuthor() const

void SetAuthor(const std::string& author)
```

# EBaseROI::GetBaseTopParent

Returns the image at the top of the hierarchy, or NULL if there is no image on top.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBaseROI* GetBaseTopParent()
```

# EBaseROI::GetBitsPerPixel

Gets the number of storage bits per pixel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetBitsPerPixel() const
```

# EBaseROI::GetColorSystem

# EBaseROI::SetColorSystem

Gets or sets the color system used by this image, as defined by the EColorSystem enumeration.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EColorSystem GetColorSystem() const
void SetColorSystem(Euresys::Open_eVision_2_10::EColorSystem colorSystem)
```

### Remarks

Upon object creation, a default color system is set, compatible with the ROI/image type (EColorSystem_GrayLevel for gray-level types and EColorSystem_Rgb for color types).
The color system associated to an image is mainly relevant when working on color images. See EasyColor (FG) for more information.

# EBaseROI::GetColPitch

Gets the pitch of a column (number of bytes between two horizontally adjacent pixels). For BW1 (one bit per pixel) images, this value is undefined.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetColPitch() const
```

# EBaseROI::GetComment

# EBaseROI::SetComment

Gets or sets the Comment attribute of the ROI/image. This is a convenience property that allows storing some information in the ROI/image. Please note that not all file formats preserve this information.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::string GetComment() const
void SetComment(const std::string& comment)
```

# EBaseROI::CopyTo

Copies all the data of the current EBaseROI object into another EBaseROI object and returns it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void CopyTo(
 EBaseROI* dest
 )
```

## Parameters

*dest*

Pointer to the EBaseROI object in which the current EBaseROI object data have to be copied.

## Remarks

This method copies all the object date to the destination object. The attached ROIs are copied recursively and attached to the destination object. They will be deleted automatically when the the destination object is deleted.
When the buffer of the source image has been provided by a call to **SetImagePtr**, the pointer will be copied into the destination image. Both images will thus refer the same external buffer.

# EBaseROI::CropToImage

This method reduces the ROI size so that it is completely contained within its topmost parent image bounds (if such an image exists at the top of the hierarchy). This means that, after calling this method, either the ROI has a zero size, or all of its pixels map to valid pixels of the underlying image buffer. If the ROI is already contained within its parent image bounds, then this method does nothing.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void CropToImage(
 )
```

# EBaseROI::GetDate

# EBaseROI::SetDate

Gets or sets the Date attribute of the ROI/image. This is a convenience property that allows storing some information in the ROI/image. Please note that not all file formats preserve this information.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::string GetDate() const
void SetDate(const std::string& date)
```

# EBaseROI::Drag

Moves the specified handle to a new position and updates all placement parameters of the ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 Euresys::Open_eVision_2_10::EDragHandle dragHandle,
 OEV_INT32 x,
 OEV_INT32 y,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*dragHandle*

Handle identifier, as defined by EDragHandle. The value returned by EBaseROI::HitTest should be used.

*x*

x-coordinate of the mouse cursor.

*y*

y-coordinate of the mouse cursor.

*zoomX*

Horizontal zoom factor. By default, true scale is used.

*zoomY*

Vertical zoom factor. By default, true scale is used.

*panX*

Horizontal pan offset. By default, no pan is added.

*panY*

Vertical pan offset. By default, no pan is added.

## Remarks

If zooming and/or panning were used when drawing the ROI, the same values must be used with EBaseROI::HitTest and EBaseROI::Drag.

# EBaseROI::Draw

Draws an ROI/image in a device context.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void Draw(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
void Draw(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the image is displayed in **TRUE** scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Pan offset (in pixels) in the horizontal direction. By default, no panning is applied.

*panY*

Pan offset (in pixels) in the vertical direction. By default, no panning is applied.

*c24Vector*

When supplied, this parameter allows using a LUT that maps from BW8 to C24 when drawing (false colors).

*bw8Vector*

When supplied, this parameter allows using a LUT that maps from BW8 to BW8 when drawing.

## Remarks

An ROI/image can be drawn (its pixels rendered) using a device context.
The horizontal and vertical zooming factors can be different and must be contained in the **1/16..16** range.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EBaseROI::DrawFrame

Draws a rectangular frame around an image or ROI.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawFrame(
 HDC graphicContext,
 BOOL handles,
 float zoomX,
 float zoomY,
 float x,
 float y
 )

void DrawFrame(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EFramePosition framePosition,
 BOOL handles,
 float zoomX,
 float zoomY,
 float x,
 float y
 )

void DrawFrame(
 HDC graphicContext,
 const ERGBColor& color,
 BOOL handles,
 float zoomX,
 float zoomY,
 float x,
 float y
 )
```

```
void DrawFrame(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EFramePosition framePosition,
 BOOL handles,
 float zoomX,
 float zoomY,
 float x,
 float y
 )
void DrawFrame(
 EDrawAdapter* graphicContext,
 BOOL handles,
 float zoomX,
 float zoomY,
 float x,
 float y
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination window.

*handles*

**TRUE** if handles are to be drawn.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the image is displayed in **TRUE** scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*x*

Translation factor for panning in the horizontal direction.

*y*

Translation factor for panning in the vertical direction.

*framePosition*

Positioning of the frame relative to the ROI.

*color*

Coçlor in which to draw the frame.

## Remarks

A frame can be drawn (using a device context) around an image or ROI, possibly with 9 sizing handles.
A suitable default pen is used (see EBaseROI::DrawFrameWithCurrentPen if you wish to use the pen currently selected into the device context).
Zooming and panning are possible. Please note that panning is applied *before* zooming.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EBaseROI::DrawFrameWithCurrentPen

Draws a rectangular frame around an image or ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawFrameWithCurrentPen(
 HDC graphicContext,
 BOOL handles,
 float zoomX,
 float zoomY,
 float x,
 float y
 )
void DrawFrameWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EFramePosition framePosition,
 BOOL handles,
 float zoomX,
 float zoomY,
 float x,
 float y
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination window.

*handles*

**TRUE** if handles are to be drawn.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the image is displayed in **TRUE** scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*x*

Translation factor for panning in the horizontal direction.

*y*

Translation factor for panning in the vertical direction.

*framePosition*

Positioning of the frame relative to the ROI.

### Remarks

A frame can be drawn (using a device context) around an image or ROI, possibly with 9 sizing handles.
The current device context pen is used. Zooming and panning are possible. Please note that
panning is applied *before* zooming.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EBaseROI::GetImagePtr

Returns a pointer to the pixel at given coordinates within the image/ROI.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void* GetImagePtr(
 OEV_INT32 x,
 OEV_INT32 y
 )

const void* GetImagePtr(
 OEV_INT32 x,
 OEV_INT32 y
 )

void* GetImagePtr(
 )
```

```
const void* GetImagePtr(
 )
```

### Parameters

*x*

The pixel x-coordinate.

*y*

The pixel y-coordinate.

### Remarks

This methods returns the memory address of the byte that contains the pixel (or address that contains the first byte of the pixel if it is bigger than one byte).
If the pixel coordinates are not specified, the method returns the address of the top-left pixel of the ROI/image.

# EBaseROI::GetSubBaseROIs

Returns all the children, and possibly recursively all their children, too.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::vector<Euresys::Open_eVision_2_10::EBaseROI*> GetSubBaseROIs(
 bool recursive
 )

std::vector<const Euresys::Open_eVision_2_10::EBaseROI*> GetSubBaseROIs(
 bool recursive
 )
```

### Parameters

*recursive*

TRUE to retrieve all sub-ROIs recursively. FALSE otherwise.

# EBaseROI::HasSubROI

Tests whether this object has a given attached ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool HasSubROI(
 const EBaseROI* subROI
 )
```

## Parameters

*subROI*

Sub ROI to find.

# EBaseROI::GetHeight

# EBaseROI::SetHeight

Gets or sets the height of the ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetHeight() const
void SetHeight(OEV_INT32 h)
```

## Remarks

The *placement of an ROI* is given by the coordinates its upper left corner (origin point), relative to its parent ROI/image and by its size (width and height).

# EBaseROI::HitTest

Detects if the cursor is placed over one of the dragging handles.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EDragHandle HitTest(
 OEV_INT32 x,
 OEV_INT32 y,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*x*

x-coordinate of the mouse cursor.

*y*

y-coordinate of the mouse cursor.

*zoomX*

Horizontal zoom factor. By default, true scale is used.

*zoomY*

Vertical zoom factor. By default, true scale is used.

*panX*

Horizontal pan offset. By default, no pan is added.

*panY*

Vertical pan offset. By default, no pan is added.

## Remarks

Returns a handle identifier, as defined by EDragHandle.
If zooming and/or panning were used when drawing the ROI, the same values must be used with EBaseROI::HitTest and EBaseROI::Drag.

# EBaseROI::IsAnROI

Tests whether this object is an ROI or an image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
bool IsAnROI(
 )
```

# EBaseROI::GetIsVoid

Tests whether if the topmost parent image of this hierarchy has a zero size.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
bool IsVoid() const
```

## Remarks

For an image, this method returns **TRUE** if the image size is zero.
For an ROI, this method returns **TRUE** if the topmost parent image size is zero or if there is no topmost image.

# EBaseROI::Load

Restores an image stored in the given file.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Load(
 const std::string& path
 )

void Load(
 ESerializer* serializer
 )
```

### Parameters

*path*

Full path of the file.

*serializer*

The ESerializer file-like object that is read from.

### Remarks

When loading, an image is resized if need be. On the opposite, an ROI cannot be resized, and the sizes *must* match. The image contents around the ROI remains unchanged.
If a serializer is used, then the Euresys proprietary file format is expected. This format preserves attributes and sub-ROIs.
See Supported Image File Types for details about supported files.
See Image File Access - Save, Load - for details about file format and compatibility.

# EBaseROI::GetOrgX

# EBaseROI::SetOrgX

Gets or sets the x-coordinate of the upper left corner of the ROI, relative to its parent ROI/image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 GetOrgX() const
```

```
void SetOrgX(OEV_INT32 x)
```

### Remarks

The *placement of an ROI* is given by the coordinates its upper left corner (origin point), relative to its parent ROI/image and by its size (width and height).

# EBaseROI::GetOrgY

# EBaseROI::SetOrgY

Gets or sets the y-coordinate of the upper left corner of the ROI, relative to its parent ROI/image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetOrgY() const
void SetOrgY(OEV_INT32 y)
```

### Remarks

The *placement of an ROI* is given by the coordinates its upper left corner (origin point), relative to its parent ROI/image and by its size (width and height).

# EBaseROI::GetParent

Returns the hierarchical parent of this object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EBaseROI* GetParent()
```

# EBaseROI::GetPlanesPerPixel

Gets the number of color components in each pixel of the ROI/image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetPlanesPerPixel() const
```

# EBaseROI::GetRowPitch

Gets the pitch of a row (the number of bytes between two vertically adjacent pixels). This is also valid for BW1 images (one bit per pixel).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 GetRowPitch() const
```

# EBaseROI::Save

Saves the EBaseROI object to the given file.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Save(
 const std::string& path,
 Euresys::Open_eVision_2_10::EImageFileType type
 )
void Save(
 ESerializer* serializer
 )
```

### Parameters

*path*

The full path of the destination file.

*type*

File format, as defined by EImageFileType. If not specified, the file format is determined from the file extension.

*serializer*

The ESerializer file-like object that is written to.

### Remarks

By default (if no format is specified), the file format is determined from the file extension.
If a serializer is used, then the Euresys proprietary file format is used. This format preserves attributes and sub-ROIs.
See Supported Image File Types for details about supported files.
See Image File Access - Save, Load - for details about file format and compatibility.

# EBaseROI::SaveJpeg

Saves the EBaseROI object to the given file, in JPEG format.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void SaveJpeg(
 const std::string& path,
 int quality
 )
```

### Parameters

*path*

The full path of the destination file.

*quality*

JPEG quality, between 0 and 100 (100 is best quality). The default value is 75.

### Remarks

See Image File Access - Save, Load - for details about file format and quality.

# EBaseROI::SaveJpeg2K

Saves the EBaseROI object to the given file, in JPEG 2000 format.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SaveJpeg2K(
 const std::string& path,
 int quality
 )
```

### Parameters

*path*

The full path of the destination file.

*quality*

JPEG 2000 quality, between 1 and 512. The default value is 16.

### Remarks

See Image File Access - Save, Load - for details about file format and quality.

# EBaseROI::SavePng

Saves the EBaseROI object to the given file, in PNG format.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SavePng(
 const std::string& path,
 int compression
 )
```

### Parameters

*path*

The full path of the destination file.

*compression*

PNG compression, between 1 and 9 (1 is the fastest and 9 is the best compression). The default value is 1.

### Remarks

See Image File Access - Save, Load - for details about file format and quality.

# EBaseROI::Serialize

Serializes the EBaseROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

Serializer used for the serialization.

# EBaseROI::SetImagePtr

Sets the pointer to an externally allocated image buffer.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetImagePtr(
 OEV_INT32 width,
 OEV_INT32 height,
 void* imagePointer,
 OEV_INT32 bitsPerRow
 )
```

### Parameters

*width*

The width of the supplied buffer, in pixels.

*height*

The height of the supplied buffer, in pixels.

*imagePointer*

The pointer (aligned on 4 bytes) to the buffer, which must be large enough to hold the image.

*bitsPerRow*

The total number of bits contained in a row, padding included. Using the value **0** (default) means that this size is computed from the buffer width and the pixel size plus a padding with the smallest possible value that leads to a multiple of 4 bytes (32 bits), which is the minimum padding accepted by EBaseROI::SetImagePtr.

### Remarks

This call is only valid on an image. An ROI gets its buffer from its parent while an image normally allocates a pixel buffer automatically. The pointer to this buffer refers to the top left pixel of the image. The next pixels are stored contiguously, row by row, from top to bottom and from left to right.

Padding at the end of a row may be used, but it must lead to rows that are multiple of 4 bytes. This method overrides the internally allocated image buffer of the EBaseROI.

As long as the image accesses this buffer, it must not be deleted.

# EBaseROI::SetPlacement

Sets the placement of an ROI, relative to its parent ROI/image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetPlacement(
 OEV_INT32 originX,
 OEV_INT32 originY,
 OEV_INT32 width,
 OEV_INT32 height
 )
```

## Parameters

*originX*

New x-coordinate of the top-left pixel of this ROI.

*originY*

New y-coordinate of the top-left pixel of this ROI.

*width*

New ROI width.

*height*

New ROI height.

## Remarks

This method can only be called on ROIs.
The *placement of an ROI* is given by the coordinates its upper left corner (origin point), relative to its parent ROI/image and by its size (width and height).

# EBaseROI::SetSize

Sets the width and height of an ROI/image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetSize(
 OEV_INT32 width,
 OEV_INT32 height
 )
void SetSize(
 const EBaseROI* other
 )
```

### Parameters

*width*

The new requested ROI/image width.

*height*

The new requested ROI/image height.

*other*

The other ROI/image whose dimensions have to be used for the current object.

### Remarks

Open eVision will allocate a new image buffer (deallocate the old image buffer) if the supplied width and height are different from the existing ones.
If an external buffer has been specified by means of **SetImagePtr**, it will be kept only if the size does not change.
Creating a new Open eVision image buffer and setting its size creates a 4-byte aligned buffer, by default. The *size of an image* is specified as a number of columns (width) and rows (height).
The maximum image dimensions are 32767 by 32767. Furthermore, it must fit into the available memory, that depends upon the physical memory, the operating system and the memory already allocated by the process in other modules or libraries.
The *placement of an ROI* is given by the x and y coordinates of its upper left pixel relative to its parent image, and also by its width and its height.

# EBaseROI::GetTitle

# EBaseROI::SetTitle

Gets or sets the Title attribute of the ROI/image. This is a convenience property that allows storing some information in the ROI/image. Please note that not all file formats preserve this information.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::string GetTitle() const
void SetTitle(const std::string& title)
```

# EBaseROI::GetTotalHeight

Gets the height, in pixels, of the ROI topmost parent.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetTotalHeight() const
```

## Remarks

The *total size* of an ROI is the size of its *topmost* parent.

# EBaseROI::GetTotalOrgX

Gets the x-coordinate of the upper left pixel of the ROI, relative to its topmost parent upper left pixel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetTotalOrgX() const
```

## Remarks

The *total origin coordinates* of an ROI indicate the position of its upper left pixel relative to the *topmost* parent image. The total origin coordinates (top-left pixel) of a topmost parent are always **(0,0)**.

# EBaseROI::GetTotalOrgY

Gets the y-coordinate of the upper left pixel of the ROI, relative to its topmost parent upper left pixel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetTotalOrgY() const
```

## Remarks

The *total origin coordinates* of an ROI indicate the position of its upper left pixel relative to the *topmost* parent. The total origin coordinates (top-left pixel) of a topmost parent are always **(0,0)**.

# EBaseROI::GetTotalWidth

Gets the width, in pixels, of the ROI topmost parent.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetTotalWidth() const
```

## Remarks

The *total size* of an ROI is the size of its *topmost* parent.

# EBaseROI::GetType

Gets the ROI/image pixel type, as defined by the EImageType enumeration.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EImageType GetType() const
```

## EBaseROI::GetWidth

## EBaseROI::SetWidth

Gets or sets the width of the ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetWidth() const
void SetWidth(OEV_INT32 w)
```

### Remarks

The *placement of an ROI* is given by the coordinates its upper left corner (origin point), relative to its parent ROI/image and by its size (width and height).

# 4.18. EBinaryImageSegmenter Class

Segments a binary image.

### Remarks

This segmenter is applicable to EROIBW1 grayscale images.
It produces coded images with two layers: The Black layer (usually, with index 0) contains the unmasked pixels having a binary value equal to zero; and the White layer (usually, with index 1) contains the remaining unmasked pixels, i.e. unmasked pixels having a binary value equal to one.

**Base Class:** ETwoLayersImageSegmenter

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

# 4.19. EBW16PathVector Class

Vector objects are used to store 1-dimensional data.

### Remarks

Using vectors is very similar to using 1-dimensional arrays, except that the size can vary at runtime. Memory allocation is handled internally. * To create a vector, use its constructor. * To fill a vector with values, first empty it, using the EBW16PathVector member, and then add elements one at time at the tail by calling the EBW16PathVector::AddElement member. * To access a vector element, either for reading or writing, use the [] operator. * To inquire for the current number of elements, use member EBW16PathVector.

**Base Class:** EVector

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddElement | Appends (adds at the tail) an element to the vector. |
| Draw | Draws a plot of the vector element values. |
| DrawWithCurrentPen | Draws a plot of the vector element values. |
| EBW16PathVector | Constructs a vector. |

| | |
|---|---|
| GetClosed | - |
| GetElement | Returns the vector element at the given index. |
| GetRawDataPtr | Pointer to the vector data. |
| operator[] | Gives access to the vector element at the given index. |
| operator= | Copies all the data from another EBW16PathVector object into the current EBW16PathVector object |
| SetClosed | - |
| SetElement | Modifies the vector element at the given index by the given value. |

E

B

# W16PathVector::AddElement

Appends (adds at the tail) an element to the vector.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void AddElement(
 EBW16Path element
 )
```

### Parameters

*element*

The element to be added.

# EBW16PathVector::GetClosed

# EBW16PathVector::SetClosed

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetClosed()
void SetClosed(BOOL bClosed)
```

# EBW16PathVector::Draw

Draws a plot of the vector element values.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )

void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )

void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*zoomX*

Zooming factor along the X axis (**1.0f** means no zoom).

*zoomY*

Zooming factor along the Y axis (**1.0f** means no zoom).

*originX*

Abscissa of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*originY*

Ordinate of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*color*

The color in which to draw the overlay.

## Remarks

The vector draws line segment between the element coordinates. The drawing appears on the image itself. Drawing is done in the device context associated to the desired window. To draw the curves, the current pen is used.

# EBW16PathVector::DrawWithCurrentPen

Draws a plot of the vector element values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*zoomX*

Zooming factor along the X axis (**1.0f** means no zoom).

*zoomY*

Zooming factor along the Y axis (**1.0f** means no zoom).

*originX*

Abscissa of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*originY*

Ordinate of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

## Remarks

The vector draws line segment between the element coordinates. The drawing appears on the image itself. Drawing is done in the device context associated to the desired window. To draw the curves, the current pen is used.

# EBW16PathVector::EBW16PathVector

Constructs a vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EBW16PathVector(
 )
void EBW16PathVector(
 OEV_UINT32 maxNumberOfElements
 )
void EBW16PathVector(
 const EBW16PathVector& other
 )
```

### Parameters

*maxNumberOfElements*

Optionally, memory can be pre-allocated to accommodate a given number of elements.

*other*

EBW16PathVector object to be copied

# EBW16PathVector::GetElement

Returns the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
EBW16Path GetElement(
 OEV_INT32 index
 )
```

### Parameters

*index*

Index, between **0** and EBW16PathVector (excluded) of the element to be accessed.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# EBW16PathVector::operator[]

Gives access to the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW16Path& operator[](
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index, between **0** and EBW16PathVector (excluded) of the element to be accessed.

# EBW16PathVector::operator=

Copies all the data from another EBW16PathVector object into the current EBW16PathVector object

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EBW16PathVector& operator=(
 const EBW16PathVector& other
 )
```

### Parameters

*other*

EBW16PathVector object to be copied

# EBW16PathVector::GetRawDataPtr

Pointer to the vector data.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void* GetRawDataPtr() const
```

# EBW16PathVector::SetElement

Modifies the vector element at the given index by the given value.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetElement(
 OEV_INT32 index,
 EBW16Path value
 )
```

### Parameters

*index*

Index, between **0** and EBW16PathVector (excluded), of the element to be modified.

*value*

The new value for the element.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# 4.20. EBW16PixelAccessor Class

Manages a BW16 pixel accessor context.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EBW16PixelAccessor | Constructor. |
| GetPixel | Gets the Pixel at the given coordinates. |
| SetPixel | Sets the Pixel at the given coordinates. |

# EBW16PixelAccessor::EBW16PixelAccessor

Constructor.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EBW16PixelAccessor(
 EROIBW16& roi
 )
void EBW16PixelAccessor(
 const EBW16PixelAccessor&
 )
```

### Parameters

*roi*

Pixel source.

-

# EBW16PixelAccessor::GetPixel

Gets the Pixel at the given coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT16 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Pixel X coordinate.

*y*

Pixel Y coordinate.

## EBW16PixelAccessor::SetPixel

Sets the Pixel at the given coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetPixel(
 OEV_UINT16 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

Pixel value.

*x*

Pixel X coordinate.

*y*

Pixel Y coordinate.

# 4.21. EBW16Vector Class

Vector objects are used to store 1-dimensional data.

### Remarks

Using vectors is very similar to using 1-dimensional arrays, except that the size can vary at runtime. Memory allocation is handled internally. * To create a vector, use its constructor. * To fill a vector with values, first empty it, using the EBW16Vector member, and then add elements one at time at the tail by calling the EBW16Vector::AddElement member. * To access a vector element, either for reading or writing, use the [] operator. * To inquire for the current number of elements, use member EBW16Vector.

**Base Class:** EVector

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddElement | Appends (adds at the tail) an element to the vector. |
| Draw | Draws a plot of the vector element values. |
| DrawWithCurrentPen | Draws a plot of the vector element values. |
| EBW16Vector | Constructs a vector. |
| GetElement | Returns the vector element at the given index. |
| GetRawDataPtr | Pointer to the vector data. |
| operator[] | Gives access to the vector element at the given index. |
| operator= | Copies all the data from another EBW16Vector object into the current EBW16Vector object |

| | |
|---|---|
| SetElement | Modifies the vector element at the given index by the given value. |
| WeightedMoment | Returns the first order geometric moment (weighted gravity center). |

# W16Vector::AddElement

Appends (adds at the tail) an element to the vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddElement(
 EBW16 element
 )
```

### Parameters

*element*

The element to be added.

# EBW16Vector::Draw

Draws a plot of the vector element values.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Draw(
 EDrawAdapter* graphicContext,
 float width,
 float height,
 float originX,
 float originY
 )

void Draw(
 HDC graphicContext,
 float width,
 float height,
 float originX,
 float originY
 )

void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 float width,
 float height,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*width*

Outermost horizontal size, in pixels.

*height*

Outermost vertical size, in pixels.

*originX*

Abscissa of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*originY*

Ordinate of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*color*

The color in which to draw the overlay.

## Remarks

A vector is able to draw itself in a window. The vector plots the element values as a function of the element indices. The drawing appears on a neutral background. Drawing is done in the device context associated to the desired window. To draw the curves, the current pen is used. To draw the annotations, a black pen is used instead.

# EBW16Vector::DrawWithCurrentPen

Draws a plot of the vector element values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 float width,
 float height,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*width*

Outermost horizontal size, in pixels.

*height*

Outermost vertical size, in pixels.

*originX*

Abscissa of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*originY*

Ordinate of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

### Remarks

A vector is able to draw itself in a window. The vector plots the element values as a function of the element indices. The drawing appears on a neutral background. Drawing is done in the device context associated to the desired window. To draw the curves, the current pen is used. To draw the annotations, a black pen is used instead.

# EBW16Vector::EBW16Vector

Constructs a vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EBW16Vector(
 )
void EBW16Vector(
 OEV_UINT32 maxNumberOfElements
 )
void EBW16Vector(
 const EBW16Vector& other
 )
```

### Parameters

*maxNumberOfElements*

Optionally, memory can be pre-allocated to accommodate a given number of elements.

*other*

EBW16Vector object to be copied

# EBW16Vector::GetElement

Returns the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW16 GetElement(
 OEV_INT32 index
 )
```

### Parameters

*index*

Index, between **0** and EBW16Vector (excluded) of the element to be accessed.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# EBW16Vector::operator[]

Gives access to the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW16& operator[](
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index, between **0** and EBW16Vector (excluded) of the element to be accessed.

# EBW16Vector::operator=

Copies all the data from another EBW16Vector object into the current EBW16Vector object

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EBW16Vector& operator=(
 const EBW16Vector& other
 )
```

### Parameters

*other*

EBW16Vector object to be copied

# EBW16Vector::GetRawDataPtr

Pointer to the vector data.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void* GetRawDataPtr() const
```

# EBW16Vector::SetElement

Modifies the vector element at the given index by the given value.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetElement(
 OEV_INT32 index,
 EBW16 value
 )
```

### Parameters

*index*

Index, between **0** and EBW16Vector (excluded), of the element to be modified.

*value*

The new value for the element.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# EBW16Vector::WeightedMoment

Returns the first order geometric moment (weighted gravity center).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float WeightedMoment(
 OEV_UINT32 from,
 OEV_UINT32 to
 )
```

### Parameters

*from*

First element of the vector portion for which the weighted moment will be calculated. By default, this is the first element of the vector.

*to*

Last element of the vector portion for which the weighted moment will be calculated. By default, this is the last element of the vector.

# 4.22. EBW32PixelAccessor Class

Manages a BW32 pixel accessor context.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EBW32PixelAccessor | Constructor. |
| GetPixel | Gets the Pixel at the given coordinates. |
| SetPixel | Sets the Pixel at the given coordinates. |

# EBW32PixelAccessor::EBW32PixelAccessor

Constructor.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EBW32PixelAccessor(
 EROIBW32& roi
 )

void EBW32PixelAccessor(
 const EBW32PixelAccessor&
 )
```

### Parameters

*roi*

Pixel source.

-

# EBW32PixelAccessor::GetPixel

Gets the Pixel at the given coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Pixel X coordinate.

*y*

Pixel Y coordinate.

# EBW32PixelAccessor::SetPixel

Sets the Pixel at the given coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void SetPixel(
 OEV_UINT32 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

Pixel value.

*x*

Pixel X coordinate.

*y*

Pixel Y coordinate.

# 4.23. EBW32Vector Class

Vector objects are used to store 1-dimensional data.

### Remarks

Using vectors is very similar to using 1-dimensional arrays, except that the size can vary at runtime. Memory allocation is handled internally. * To create a vector, use its constructor. * To fill a vector with values, first empty it, using the EBW32Vector member, and then add elements one at time at the tail by calling the EBW32Vector::AddElement member. * To access a vector element, either for reading or writing, use the [] operator. * To inquire for the current number of elements, use member EBW32Vector.

**Base Class:** EVector

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddElement | Appends (adds at the tail) an element to the vector. |
| Draw | Draws a plot of the vector element values. |

| | |
|---|---|
| DrawWithCurrentPen | Draws a plot of the vector element values. |
| EBW32Vector | Constructs a vector. |
| GetElement | Returns the vector element at the given index. |
| GetRawDataPtr | Pointer to the vector data. |
| operator[] | Gives access to the vector element at the given index. |
| operator= | Copies all the data from another EBW32Vector object into the current EBW32Vector object |
| SetElement | Modifies the vector element at the given index by the given value. |
| WeightedMoment | Returns the first order geometric moment (weighted gravity center). |

E

B

# W32Vector::AddElement

Appends (adds at the tail) an element to the vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddElement(
 EBW32 element
 )
```

### Parameters

*element*

The element to be added.

# EBW32Vector::Draw

Draws a plot of the vector element values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 EDrawAdapter* graphicContext,
 float width,
 float height,
 float originX,
 float originY
 )
void Draw(
 HDC graphicContext,
 float width,
 float height,
 float originX,
 float originY
 )
```

```
void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 float width,
 float height,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*width*

Outermost horizontal size, in pixels.

*height*

Outermost vertical size, in pixels.

*originX*

Abscissa of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*originY*

Ordinate of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*color*

The color in which to draw the overlay.

### Remarks

A vector is able to draw itself in a window. The vector plots the element values as a function of the element indices. The drawing appears on a neutral background. Drawing is done in the device context associated to the desired window. To draw the curves, the current pen is used. To draw the annotations, a black pen is used instead.

# EBW32Vector::DrawWithCurrentPen

Draws a plot of the vector element values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 float width,
 float height,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*width*

Outermost horizontal size, in pixels.

*height*

Outermost vertical size, in pixels.

*originX*

Abscissa of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*originY*

Ordinate of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

### Remarks

A vector is able to draw itself in a window. The vector plots the element values as a function of the element indices. The drawing appears on a neutral background. Drawing is done in the device context associated to the desired window. To draw the curves, the current pen is used. To draw the annotations, a black pen is used instead.

# EBW32Vector::EBW32Vector

Constructs a vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EBW32Vector(
 )
void EBW32Vector(
 OEV_UINT32 maxNumberOfElements
 )
void EBW32Vector(
 const EBW32Vector& other
 )
```

### Parameters

*maxNumberOfElements*

Optionally, memory can be pre-allocated to accommodate a given number of elements.

*other*

EBW32Vector object to be copied

# EBW32Vector::GetElement

Returns the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW32 GetElement(
 OEV_INT32 index
 )
```

### Parameters

*index*

Index, between **0** and EBW32Vector (excluded) of the element to be accessed.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# EBW32Vector::operator[]

Gives access to the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW32& operator[](
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index, between **0** and EBW32Vector (excluded) of the element to be accessed.

# EBW32Vector::operator=

Copies all the data from another EBW32Vector object into the current EBW32Vector object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW32Vector& operator=(
 const EBW32Vector& other
 )
```

### Parameters

*other*

EBW32Vector object to be copied

# EBW32Vector::GetRawDataPtr

Pointer to the vector data.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void* GetRawDataPtr() const
```

# EBW32Vector::SetElement

Modifies the vector element at the given index by the given value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetElement(
 OEV_INT32 index,
 EBW32 value
 )
```

## Parameters

*index*

Index, between **0** and EBW32Vector (excluded), of the element to be modified.

*value*

The new value for the element.

## Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# EBW32Vector::WeightedMoment

Returns the first order geometric moment (weighted gravity center).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float WeightedMoment(
 OEV_UINT32 from,
 OEV_UINT32 to
 )
```

### Parameters

*from*

First element of the vector portion for which the weighted moment will be calculated. By default, this is the first element of the vector.

*to*

Last element of the vector portion for which the weighted moment will be calculated. By default, this is the last element of the vector.

# 4.24. EBW8PathVector Class

Vector objects are used to store 1-dimensional data.

### Remarks

Using vectors is very similar to using 1-dimensional arrays, except that the size can vary at runtime. Memory allocation is handled internally. * To create a vector, use its constructor. * To fill a vector with values, first empty it, using the EBW8PathVector member, and then add elements one at time at the tail by calling the EBW8PathVector::AddElement member. * To access a vector element, either for reading or writing, use the [] operator. * To inquire for the current number of elements, use member EBW8PathVector.

**Base Class:** EVector

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddElement | Appends (adds at the tail) an element to the vector. |
| Draw | Draws a plot of the vector element values. |
| DrawWithCurrentPen | Draws a plot of the vector element values. |
| EBW8PathVector | Constructs a vector. |
| GetClosed | Flag indicating whether the shape built with EasyImage::Contour must be closed or not. |
| GetElement | Returns the vector element at the given index. |
| GetRawDataPtr | Pointer to the vector data. |
| operator[] | Gives access to the vector element at the given index. |
| operator= | Copies all the data from another EBW8PathVector object into the current EBW8PathVector object |
| SetClosed | Flag indicating whether the shape built with EasyImage::Contour must be closed or not. |

| | |
|---|---|
| SetElement | Modifies the vector element at the given index by the given value. |

B

# W8PathVector::AddElement

Appends (adds at the tail) an element to the vector.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void AddElement(
 EBW8Path element
 )
```

## Parameters

*element*

The element to be added.

# EBW8PathVector::GetClosed

# EBW8PathVector::SetClosed

Flag indicating whether the shape built with EasyImage::Contour must be closed or not.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetClosed()
void SetClosed(BOOL bClosed)
```

# EBW8PathVector::Draw

Draws a plot of the vector element values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*zoomX*

Zooming factor along the X axis (**1.0f** means no zoom).

*zoomY*

Zooming factor along the Y axis (**1.0f** means no zoom).

*originX*

Abscissa of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*originY*

Ordinate of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*color*

The color in which to draw the overlay.

### Remarks

The vector draws line segment between the element coordinates. The drawing appears on the image itself. Drawing is done in the device context associated to the desired window. To draw the curves, the current pen is used.

# EBW8PathVector::DrawWithCurrentPen

Draws a plot of the vector element values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*zoomX*

Zooming factor along the X axis (**1.0f** means no zoom).

*zoomY*

Zooming factor along the Y axis (**1.0f** means no zoom).

*originX*

Abscissa of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*originY*

Ordinate of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

### Remarks

The vector draws line segment between the element coordinates. The drawing appears on the image itself. Drawing is done in the device context associated to the desired window. To draw the curves, the current pen is used.

# EBW8PathVector::EBW8PathVector

Constructs a vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EBW8PathVector(
 )
void EBW8PathVector(
 OEV_UINT32 maxNumberOfElements
 )
void EBW8PathVector(
 const EBW8PathVector& other
 )
```

### Parameters

*maxNumberOfElements*

Optionally, memory can be pre-allocated to accommodate a given number of elements.

*other*

EBW8PathVector object to be copied

# EBW8PathVector::GetElement

Returns the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW8Path GetElement(
 OEV_INT32 index
 )
```

### Parameters

*index*

Index, between **0** and EBW8PathVector (excluded) of the element to be accessed.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# EBW8PathVector::operator[]

Gives access to the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW8Path& operator[](
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index, between **0** and EBW8PathVector (excluded) of the element to be accessed.

# EBW8PathVector::operator=

Copies all the data from another EBW8PathVector object into the current EBW8PathVector object

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EBW8PathVector& operator=(
 const EBW8PathVector& other
 )
```

## Parameters

*other*

EBW8PathVector object to be copied

# EBW8PathVector::GetRawDataPtr

Pointer to the vector data.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void* GetRawDataPtr() const
```

# EBW8PathVector::SetElement

Modifies the vector element at the given index by the given value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetElement(
 OEV_INT32 index,
 EBW8Path value
 )
```

### Parameters

*index*

Index, between **0** and EBW8PathVector (excluded), of the element to be modified.

*value*

The new value for the element.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# 4.25. EBW8PixelAccessor Class

Manages a BW8 pixel accessor context.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EBW8PixelAccessor | Constructor. |
| GetPixel | Gets the Pixel at the given coordinates. |
| SetPixel | Sets the Pixel at the given coordinates. |

# EBW8PixelAccessor::EBW8PixelAccessor

Constructor.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EBW8PixelAccessor(
 EROIBW8& roi
 )
void EBW8PixelAccessor(
 const EBW8PixelAccessor&
 )
```

### Parameters

*roi*

Pixel source.

-

# EBW8PixelAccessor::GetPixel

Gets the Pixel at the given coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT8 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Pixel X coordinate.

*y*

Pixel Y coordinate.

## EBW8PixelAccessor::SetPixel

Sets the Pixel at the given coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetPixel(
 OEV_UINT8 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

Pixel value.

*x*

Pixel X coordinate.

*y*

Pixel Y coordinate.

# 4.26. EBW8Vector Class

Vector objects are used to store 1-dimensional data.

### Remarks

Using vectors is very similar to using 1-dimensional arrays, except that the size can vary at runtime. Memory allocation is handled internally. * To create a vector, use its constructor. * To fill a vector with values, first empty it, using the EBW8Vector member, and then add elements one at time at the tail by calling the EBW8Vector::AddElement member. * To access a vector element, either for reading or writing, use the [] operator. * To inquire for the current number of elements, use member EBW8Vector.

**Base Class:** EVector

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddElement | Appends (adds at the tail) an element to the vector. |
| Draw | Draws a plot of the vector element values. |
| DrawWithCurrentPen | Draws a plot of the vector element values. |
| EBW8Vector | Constructs a vector. |
| GetElement | Returns the vector element at the given index. |
| GetRawDataPtr | Pointer to the vector data. |
| operator[] | Gives access to the vector element at the given index. |
| operator= | Copies all the data from another EBW8Vector object into the current EBW8Vector object |

| | |
|---|---|
| SetElement | Modifies the vector element at the given index by the given value. |
| WeightedMoment | Returns the first order geometric moment (weighted gravity center). |

# W8Vector::AddElement

Appends (adds at the tail) an element to the vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddElement(
 EBW8 element
 )
```

### Parameters

*element*

The element to be added.

# EBW8Vector::Draw

Draws a plot of the vector element values.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Draw(
 HDC graphicContext,
 float width,
 float height,
 float originX,
 float originY
 )

void Draw(
 EDrawAdapter* graphicContext,
 float width,
 float height,
 float originX,
 float originY
 )

void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 float width,
 float height,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*width*

Outermost horizontal size, in pixels.

*height*

Outermost vertical size, in pixels.

*originX*

Abscissa of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*originY*

Ordinate of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*color*

The color in which to draw the overlay.

### Remarks

A vector is able to draw itself in a window. The vector plots the element values as a function of the element indices. The drawing appears on a neutral background. Drawing is done in the device context associated to the desired window. To draw the curves, the current pen is used. To draw the annotations, a black pen is used instead.

# EBW8Vector::DrawWithCurrentPen

Draws a plot of the vector element values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 float width,
 float height,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*width*

Outermost horizontal size, in pixels.

*height*

Outermost vertical size, in pixels.

*originX*

Abscissa of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*originY*

Ordinate of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

### Remarks

A vector is able to draw itself in a window. The vector plots the element values as a function of the element indices. The drawing appears on a neutral background. Drawing is done in the device context associated to the desired window. To draw the curves, the current pen is used. To draw the annotations, a black pen is used instead.

# EBW8Vector::EBW8Vector

Constructs a vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EBW8Vector(
 )
void EBW8Vector(
 OEV_UINT32 maxNumberOfElements
 )
void EBW8Vector(
 const EBW8Vector& other
 )
```

### Parameters

*maxNumberOfElements*

Optionally, memory can be pre-allocated to accommodate a given number of elements.

*other*

EBW8Vector object to be copied

# EBW8Vector::GetElement

Returns the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW8 GetElement(
 OEV_INT32 index
 )
```

### Parameters

*index*

Index, between **0** and EBW8Vector (excluded) of the element to be accessed.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# EBW8Vector::operator[]

Gives access to the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW8& operator[](
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index, between **0** and EBW8Vector (excluded) of the element to be accessed.

# EBW8Vector::operator=

Copies all the data from another EBW8Vector object into the current EBW8Vector object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW8Vector& operator=(
 const EBW8Vector& other
 )
```

### Parameters

*other*

EBW8Vector object to be copied

# EBW8Vector::GetRawDataPtr

Pointer to the vector data.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void* GetRawDataPtr() const
```

# EBW8Vector::SetElement

Modifies the vector element at the given index by the given value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void SetElement(
 OEV_INT32 index,
 EBW8 value
 )
```

### Parameters

*index*

Index, between **0** and EBW8Vector (excluded), of the element to be modified.

*value*

The new value for the element.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# EBW8Vector::WeightedMoment

Returns the first order geometric moment (weighted gravity center).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float WeightedMoment(
 OEV_UINT32 from,
 OEV_UINT32 to
 )
```

### Parameters

*from*

First element of the vector portion for which the weighted moment will be calculated. By default, this is the first element of the vector.

*to*

Last element of the vector portion for which the weighted moment will be calculated. By default, this is the last element of the vector.

# 4.27. EBWHistogramVector Class

Vector objects are used to store 1-dimensional data.

### Remarks

Using vectors is very similar to using 1-dimensional arrays, except that the size can vary at runtime. Memory allocation is handled internally. * To create a vector, use its constructor. * To fill a vector with values, first empty it, using the EBWHistogramVector member, and then add elements one at time at the tail by calling the EBWHistogramVector::AddElement member. * To access a vector element, either for reading or writing, use the [] operator. * To inquire for the current number of elements, use member EBWHistogramVector.

**Base Class:** EVector

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddElement | Appends (adds at the tail) an element to the vector. |
| Draw | Draws a plot of the vector element values. |
| DrawWithCurrentPen | Draws a plot of the vector element values. |
| EBWHistogramVector | Constructs a vector. |
| GetElement | Returns the vector element at the given index. |
| GetRawDataPtr | Pointer to the vector data. |

| | |
|---|---|
| operator[] | Gives access to the vector element at the given index. |
| operator= | Copies all the data from another EBWHistogramVector object into the current EBWHistogramVector object |
| SetElement | Modifies the vector element at the given index by the given value. |

E

B

# WHistogramVector::AddElement

Appends (adds at the tail) an element to the vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddElement(
 OEV_UINT32 element
 )
```

### Parameters

*element*

The element to be added.

# EBWHistogramVector::Draw

Draws a plot of the vector element values.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Draw(
 EDrawAdapter* graphicContext,
 float width,
 float height,
 float originX,
 float originY
 )

void Draw(
 HDC graphicContext,
 float width,
 float height,
 float originX,
 float originY
 )

void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 float width,
 float height,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*width*

Outermost horizontal size, in pixels.

*height*

Outermost vertical size, in pixels.

*originX*

Abscissa of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*originY*

Ordinate of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*color*

The color in which to draw the overlay.

### Remarks

A vector is able to draw itself in a window. The vector plots the element values as a function of the element indices. The drawing appears on a neutral background. Drawing is done in the device context associated to the desired window. To draw the curves, the current pen is used. To draw the annotations, a black pen is used instead.

# EBWHistogramVector::DrawWithCurrentPen

Draws a plot of the vector element values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 float width,
 float height,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*width*

Outermost horizontal size, in pixels.

*height*

Outermost vertical size, in pixels.

*originX*

Abscissa of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*originY*

Ordinate of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

### Remarks

A vector is able to draw itself in a window. The vector plots the element values as a function of the element indices. The drawing appears on a neutral background. Drawing is done in the device context associated to the desired window. To draw the curves, the current pen is used. To draw the annotations, a black pen is used instead.

# EBWHistogramVector::EBWHistogramVector

Constructs a vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EBWHistogramVector(
 )
void EBWHistogramVector(
 const EBWHistogramVector& other
 )
void EBWHistogramVector(
 OEV_UINT32 maxNumberOfElements
 )
```

### Parameters

*other*

EBWHistogramVector object to be copied

*maxNumberOfElements*

Optionally, memory can be pre-allocated to accommodate a given number of elements.

# EBWHistogramVector::GetElement

Returns the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetElement(
 OEV_INT32 index
 )
```

### Parameters

*index*

Index, between **0** and EBWHistogramVector (excluded) of the element to be accessed.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# EBWHistogramVector::operator[]

Gives access to the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32& operator[](
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index, between **0** and EBWHistogramVector (excluded) of the element to be accessed.

# EBWHistogramVector::operator=

Copies all the data from another EBWHistogramVector object into the current EBWHistogramVector object

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EBWHistogramVector& operator=(
 const EBWHistogramVector& other
 )
```

### Parameters

*other*

EBWHistogramVector object to be copied

# EBWHistogramVector::GetRawDataPtr

Pointer to the vector data.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void* GetRawDataPtr() const
```

# EBWHistogramVector::SetElement

Modifies the vector element at the given index by the given value.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetElement(
 OEV_INT32 index,
 OEV_UINT32 value
 )
```

### Parameters

*index*

Index, between **0** and EBWHistogramVector (excluded), of the element to be modified.

*value*

The new value for the element.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# 4.28. EC15PixelAccessor Class

Manages a C15 pixel accessor context.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EC15PixelAccessor | Constructor. |
| GetPixel | Gets the Pixel at the given coordinates. |
| SetPixel | Sets the Pixel at the given coordinates. |

# EC15PixelAccessor::EC15PixelAccessor

Constructor.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EC15PixelAccessor(
 EROIC15& roi
 )
void EC15PixelAccessor(
 const EC15PixelAccessor&
 )
```

### Parameters

*roi*

Pixel source.

-

# EC15PixelAccessor::GetPixel

Gets the Pixel at the given coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EC15 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Pixel X coordinate.

*y*

Pixel Y coordinate.

## EC15PixelAccessor::SetPixel

Sets the Pixel at the given coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetPixel(
 EC15 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

Pixel value.

*x*

Pixel X coordinate.

*y*

Pixel Y coordinate.

# 4.29. EC16PixelAccessor Class

Manages a C16 pixel accessor context.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EC16PixelAccessor | Constructor. |
| GetPixel | Gets the Pixel at the given coordinates. |
| SetPixel | Sets the Pixel at the given coordinates. |

# C16PixelAccessor::EC16PixelAccessor

Constructor.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EC16PixelAccessor(
 EROIC16& roi
 )
void EC16PixelAccessor(
 const EC16PixelAccessor&
 )
```

### Parameters

*roi*

Pixel source.

-

# EC16PixelAccessor::GetPixel

Gets the Pixel at the given coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EC16 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Pixel X coordinate.

*y*

Pixel Y coordinate.

# EC16PixelAccessor::SetPixel

Sets the Pixel at the given coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetPixel(
 EC16 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

Pixel value.

*x*

Pixel X coordinate.

*y*

Pixel Y coordinate.

# 4.30. EC24APixelAccessor Class

Manages a C24A pixel accessor context.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EC24APixelAccessor | Constructor. |
| GetPixel | Gets the Pixel at the given coordinates. |
| SetPixel | Sets the Pixel at the given coordinates. |

# EC24APixelAccessor::EC24APixelAccessor

Constructor.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EC24APixelAccessor(
 EROIC24A& roi
 )
void EC24APixelAccessor(
 const EC24APixelAccessor&
 )
```

### Parameters

*roi*

Pixel source.

-

# EC24APixelAccessor::GetPixel

Gets the Pixel at the given coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EC24A GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Pixel X coordinate.

*y*

Pixel Y coordinate.

# EC24APixelAccessor::SetPixel

Sets the Pixel at the given coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetPixel(
 EC24A value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

Pixel value.

*x*

Pixel X coordinate.

*y*

Pixel Y coordinate.

# 4.31. EC24PathVector Class

Vector objects are used to store 1-dimensional data.

### Remarks

Using vectors is very similar to using 1-dimensional arrays, except that the size can vary at runtime. Memory allocation is handled internally. * To create a vector, use its constructor. * To fill a vector with values, first empty it, using the EC24PathVector member, and then add elements one at time at the tail by calling the EC24PathVector::AddElement member. * To access a vector element, either for reading or writing, use the [] operator. * To inquire for the current number of elements, use member EC24PathVector.

**Base Class:** EVector

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddElement | Appends (adds at the tail) an element to the vector. |
| Draw | Draws a plot of the vector element values. |
| DrawWithCurrentPen | Draws a plot of the vector element values. |
| EC24PathVector | Constructs a vector. |
| GetClosed | - |
| GetElement | Returns the vector element at the given index. |
| GetRawDataPtr | Pointer to the vector data. |
| operator[] | Gives access to the vector element at the given index. |
| operator= | Copies all the data from another EC24PathVector object into the current EC24PathVector object |
| SetClosed | - |
| SetElement | Modifies the vector element at the given index by the given value. |

# EC24PathVector::AddElement

Appends (adds at the tail) an element to the vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddElement(
 EC24Path element
 )
```

### Parameters

*element*

The element to be added.

# EC24PathVector::GetClosed

# EC24PathVector::SetClosed

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetClosed()
void SetClosed(BOOL bClosed)
```

# EC24PathVector::Draw

Draws a plot of the vector element values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*zoomX*

Zooming factor along the X axis (**1.0f** means no zoom).

*zoomY*

Zooming factor along the Y axis (**1.0f** means no zoom).

*originX*

Abscissa of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*originY*

Ordinate of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*color*

The color in which to draw the overlay.

### Remarks

The vector draws line segment between the element coordinates. The drawing appears on the image itself. Drawing is done in the device context associated to the desired window. To draw the curves, the current pen is used.

# EC24PathVector::DrawWithCurrentPen

Draws a plot of the vector element values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*zoomX*

Zooming factor along the X axis (**1.0f** means no zoom).

*zoomY*

Zooming factor along the Y axis (**1.0f** means no zoom).

*originX*

Abscissa of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*originY*

Ordinate of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

### Remarks

The vector draws line segment between the element coordinates. The drawing appears on the image itself. Drawing is done in the device context associated to the desired window. To draw the curves, the current pen is used.

# EC24PathVector::EC24PathVector

Constructs a vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EC24PathVector(
 )
void EC24PathVector(
 const EC24PathVector& other
 )
void EC24PathVector(
 OEV_UINT32 maxNumberOfElements
 )
```

### Parameters

*other*

EC24PathVector object to be copied

*maxNumberOfElements*

Optionally, memory can be pre-allocated to accommodate a given number of elements.

# EC24PathVector::GetElement

Returns the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EC24Path GetElement(
 OEV_INT32 index
 )
```

### Parameters

*index*

Index, between **0** and EC24PathVector (excluded) of the element to be accessed.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# EC24PathVector::operator[]

Gives access to the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EC24Path& operator[](
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index, between **0** and EC24PathVector (excluded) of the element to be accessed.

# EC24PathVector::operator=

Copies all the data from another EC24PathVector object into the current EC24PathVector object

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EC24PathVector& operator=(
 const EC24PathVector& other
 )
```

### Parameters

*other*

EC24PathVector object to be copied

# EC24PathVector::GetRawDataPtr

Pointer to the vector data.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void* GetRawDataPtr() const
```

# EC24PathVector::SetElement

Modifies the vector element at the given index by the given value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetElement(
 OEV_INT32 index,
 EC24Path value
 )
```

### Parameters

*index*

Index, between **0** and EC24PathVector (excluded), of the element to be modified.

*value*

The new value for the element.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# 4.32. EC24PixelAccessor Class

Manages a C24 pixel accessor context.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EC24PixelAccessor | Constructor. |
| GetPixel | Gets the Pixel at the given coordinates. |
| SetPixel | Sets the Pixel at the given coordinates. |

# EC24PixelAccessor::EC24PixelAccessor

Constructor.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EC24PixelAccessor(
 EROIC24& roi
 )
void EC24PixelAccessor(
 const EC24PixelAccessor&
 )
```

### Parameters

*roi*

Pixel source.

-

# EC24PixelAccessor::GetPixel

Gets the Pixel at the given coordinates.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EC24 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Pixel X coordinate.

*y*

Pixel Y coordinate.

## EC24PixelAccessor::SetPixel

Sets the Pixel at the given coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetPixel(
 EC24 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

Pixel value.

*x*

Pixel X coordinate.

*y*

Pixel Y coordinate.

# 4.33. EC24Vector Class

Vector objects are used to store 1-dimensional data.

### Remarks

Using vectors is very similar to using 1-dimensional arrays, except that the size can vary at runtime. Memory allocation is handled internally. * To create a vector, use its constructor. * To fill a vector with values, first empty it, using the EC24Vector member, and then add elements one at time at the tail by calling the EC24Vector::AddElement member. * To access a vector element, either for reading or writing, use the [] operator. * To inquire for the current number of elements, use member EC24Vector.

**Base Class:** EVector

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddElement | Appends (adds at the tail) an element to the vector. |
| Draw | Draws a plot of the vector element values. |
| EC24Vector | Constructs a vector. |
| GetElement | Returns the vector element at the given index. |
| GetRawDataPtr | Pointer to the vector data. |
| operator[] | Gives access to the vector element at the given index. |
| operator= | Copies all the data from another EC24Vector object into the current EC24Vector object |
| SetElement | Modifies the vector element at the given index by the given value. |

# EC24Vector::AddElement

Appends (adds at the tail) an element to the vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddElement(
 EC24 element
 )
```

### Parameters

*element*

The element to be added.

# EC24Vector::Draw

Draws a plot of the vector element values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 HDC graphicContext,
 float width,
 float height
 )
```

```
void Draw(
 HDC graphicContext,
 float width,
 float height,
 float originX,
 float originY
 )

void Draw(
 HDC graphicContext,
 float width,
 float height,
 float originX,
 float originY,
 const ERGBColor& color0,
 const ERGBColor& color1,
 const ERGBColor& color2
 )

void Draw(
 HDC graphicContext,
 float width,
 float height,
 const ERGBColor& color0,
 const ERGBColor& color1,
 const ERGBColor& color2
 )

void Draw(
 EDrawAdapter* graphicContext,
 float width,
 float height
 )

void Draw(
 EDrawAdapter* graphicContext,
 float width,
 float height,
 float originX,
 float originY
 )
```

```
void Draw(
 EDrawAdapter* graphicContext,
 float width,
 float height,
 float originX,
 float originY,
 const ERGBColor& color0,
 const ERGBColor& color1,
 const ERGBColor& color2
 )
void Draw(
 EDrawAdapter* graphicContext,
 float width,
 float height,
 const ERGBColor& color0,
 const ERGBColor& color1,
 const ERGBColor& color2
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*width*

Outermost horizontal size, in pixels.

*height*

Outermost vertical size, in pixels.

*originX*

Abscissa of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*originY*

Ordinate of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*color0*

The color to be used when drawing the curve of the first color component of the vector, as an RGB color. By default, the current pen is used to draw the curve.

*color1*

The color to be used when drawing the curve of the second color component of the vector, as an RGB color. By default, the current pen is used to draw the curve.

*color2*

The color to be used when drawing the curve of the third color component of the vector, as an RGB color. By default, the current pen is used to draw the curve.

### Remarks

A vector is able to draw itself in a window. The vector plots the element values as a function of the element indices. The drawing appears on a neutral background. Drawing is done in the device context associated to the desired window. To draw the curves, the current pen is used. To draw the annotations, a black pen is used instead. In the special case of the EC24Vector, three curves are drawn instead of one, each corresponding to a color component. Three pen objects must be provided to draw the curves with appropriate attributes.

# EC24Vector::EC24Vector

Constructs a vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EC24Vector(
 )
void EC24Vector(
 OEV_UINT32 maxNumberOfElements
 )
void EC24Vector(
 const EC24Vector& other
 )
```

### Parameters

*maxNumberOfElements*

Optionally, memory can be pre-allocated to accommodate a given number of elements.

*other*

EC24Vector object to be copied

# EC24Vector::GetElement

Returns the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EC24 GetElement(
 OEV_INT32 index
 )
```

### Parameters

*index*

Index, between **0** and EC24Vector (excluded) of the element to be accessed.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# EC24Vector::operator[]

Gives access to the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EC24& operator[](
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index, between **0** and EC24Vector (excluded) of the element to be accessed.

# EC24Vector::operator=

Copies all the data from another EC24Vector object into the current EC24Vector object

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EC24Vector& operator=(
 const EC24Vector& other
 )
```

### Parameters

*other*

EC24Vector object to be copied

# EC24Vector::GetRawDataPtr

Pointer to the vector data.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void* GetRawDataPtr() const
```

# EC24Vector::SetElement

Modifies the vector element at the given index by the given value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetElement(
 OEV_INT32 index,
 EC24 value
 )
```

### Parameters

*index*

Index, between **0** and EC24Vector (excluded), of the element to be modified.

*value*

The new value for the element.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# 4.34. ECalibrationGenerator Class

Represents a 3D calibration model generator, a class made to compute calibration models.

**Derived Class(es):** EObjectBasedCalibrationGenerator
**Namespace:** Euresys::Open_eVision_2_10::Easy3D

# 4.35. ECalibrationModel Class

Represents a 3D calibration model.

**Derived Class(es):** EExplicitGeometricCalibrationModel EObjectBasedCalibrationModel EScaleCalibrationModel
**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| Apply | Applies this model to convert an uncalibrated point to a world position.<br>This method returns a world position. |
| Create | Factory method from a serializer stream: allocates and reads the calibration model from the given serializer.<br>Returns the corresponding calibration model, must be released by caller. |
| GetType | Returns the type of calibration model, see ECalibrationType. |
| Save | Saves the calibration model. The given ESerializer must have been |

# ECalibrationModel::Apply

Applies this model to convert an uncalibrated point to a world position.
This method returns a world position.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPoint Apply(
 E3DPoint uvwPoint
 )
```

### Parameters

*uvwPoint*

The position of a depth map pixel.

# ECalibrationModel::Create

Factory method from a serializer stream: allocates and reads the calibration model from the given serializer. Returns the corresponding calibration model, must be released by caller.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
ECalibrationModel* Create(
 ESerializer* file
 )
```

### Parameters

*file*

A serializer created for reading.

# ECalibrationModel::Save

Saves the calibration model. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

-

# ECalibrationModel::GetType

Returns the type of calibration model, see ECalibrationType.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
Euresys::Open_eVision_2_10::Easy3D::ECalibrationType GetType() const
```

# 4.36. ECannyEdgeDetector Class

Manages a complete context for the Canny edge detector.

### Remarks

The Canny edge detector operates on a grayscale BW8 image and delivers a black-and-white BW8 image where pixels have only 2 possible values: 0 and 255. Pixels corresponding to edges in the source image are set to value 255 in the output image; The other pixels are set to value 0.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Apply | Apply the Canny edge detector on an image/ROI. |
| ECannyEdgeDetector | Constructs a ECannyEdgeDetector object initialized to its default values. |
| GetHighThreshold | Sets the high hysteresis threshold for a pixel to be considered as an edge. |

| | |
|---|---|
| GetLowThreshold | Sets the low hysteresis threshold for a pixel to be considered as an edge. |
| GetSmoothingScale | The scale of the features of interest. |
| GetThresholdingMode | Sets the mode of the hysteresis thresholding. |
| ResetSmoothingScale | Prevents the smoothing of the source image by a Gaussian filter. |
| SetHighThreshold | Sets the high hysteresis threshold for a pixel to be considered as an edge. |
| SetLowThreshold | Sets the low hysteresis threshold for a pixel to be considered as an edge. |
| SetSmoothingScale | The scale of the features of interest. |
| SetThresholdingMode | Sets the mode of the hysteresis thresholding. |

E

C

# annyEdgeDetector::Apply

Apply the Canny edge detector on an image/ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Apply(
 const EROIBW8& source,
 EROIBW8& result
 )
```

### Parameters

*source*

The source image/ROI.

*result*

The output image/ROI.

### Remarks

The output ROI must have the same size than the input ROI.

# ECannyEdgeDetector::ECannyEdgeDetector

Constructs a ECannyEdgeDetector object initialized to its default values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ECannyEdgeDetector(
 const ECannyEdgeDetector& other
 )

void ECannyEdgeDetector(
 )
```

### Parameters

*other*

-

# ECannyEdgeDetector::GetHighThreshold

# ECannyEdgeDetector::SetHighThreshold

Sets the high hysteresis threshold for a pixel to be considered as an edge.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetHighThreshold() const
void SetHighThreshold(float highThreshold)
```

# ECannyEdgeDetector::GetLowThreshold

# ECannyEdgeDetector::SetLowThreshold

Sets the low hysteresis threshold for a pixel to be considered as an edge.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetLowThreshold() const
void SetLowThreshold(float lowThreshold)
```

# ECannyEdgeDetector::ResetSmoothingScale

Prevents the smoothing of the source image by a Gaussian filter.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ResetSmoothingScale(
 )
```

## Remarks

Calling this method is equivalent to set ECannyEdgeDetector::SmoothingScale to zero. It disables the use of the Gaussian filter.

# ECannyEdgeDetector::GetSmoothingScale

# ECannyEdgeDetector::SetSmoothingScale

The scale of the features of interest.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSmoothingScale() const
void SetSmoothingScale(float scale)
```

### Remarks

This scale corresponds to the standard deviation of the Gaussian filter that is used to smooth the source image before the computation of the gradient, hereby selecting the scale of the features of interest.
If this scale is set to zero, no smoothing is achieved: The gradient is computed directly on the raw source image, speeding up the detector, but making the process much less reliable.

## ECannyEdgeDetector::GetThresholdingMode

## ECannyEdgeDetector::SetThresholdingMode

Sets the mode of the hysteresis thresholding.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::ECannyThresholdingMode GetThresholdingMode() const
void SetThresholdingMode(Euresys::Open_eVision_2_10::ECannyThresholdingMode mode)
```

### Remarks

If the threshold mode is set to ECannyThresholdingMode_Absolute, the threshold values are interpreted as absolute thresholds. In this case, the thresholds must be strictly positive real values.

If the threshold mode is set to ECannyThresholdingMode_Relative, the thresholds are expressed as a fraction ranging from 0 to 1 of the maximum value of the gradient of the source image.

In either case, the low threshold must be less than the high threshold.

# 4.37. EChecker Class

Manages a complete context for the inspection tool based on image comparison in EasyOCV.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddPathName | Adds a single file pathname. |
| Attach | Associates a source image to a checker context. |
| BatchLearn | Performs the learning sequence using the specified list of image files. |
| Drag | Moves the relevant ROI by means of its handle. |
| Draw | Draws one of the geometric items that define the EChecker tool. |
| DrawWithCurrentPen | Draws one of the geometric items that define the EChecker tool. |
| EChecker | Constructs an uninitialized checker context. |
| EmptyPathNames | Clears the list of file pathnames. |
| GetAverage | Global intensity of the mother image. |
| GetDarkGray | - |
| GetDegreesOfFreedom | Boolean combination of EDegreesOfFreedom members, that indicates which degrees of freedom are to be considered. |

| | |
|---|---|
| GetDeviation | Global contrast of the mother image. |
| GetHigh | High threshold image for the adaptive segmentation. |
| GetHitHandle | Handle currently hit. |
| GetHitRoi | ROI currently hit. |
| GetLightGray | - |
| GetLow | Low threshold image for the adaptive segmentation. |
| GetNormalize | Current normalization mode. |
| GetNumAverageSamples | Number of samples that were accumulated in the "average" phase of the training. |
| GetNumDeviationSamples | Number of samples that were accumulated in the "deviation" phase of the training. |
| GetPanX | Current horizontal panning factor for use in display operations. |
| GetPanY | Current vertical panning factor for use in display operations. |

| | |
|---|---|
| GetRegistered | Represents the source image, after it has been aligned with the reference image. |
| GetRelativeTolerance | Current tolerance factor to be used for threshold image setup. |
| GetToleranceX | Current horizontal search tolerance, in pixels. |
| GetToleranceY | Current vertical search tolerance, in pixels. |
| GetZoomX | Current horizontal zooming factor for use in display operations. |
| GetZoomY | Current vertical zooming factor for use in display operations. |
| HitTest | Returns **TRUE** if the cursor is over one of the dragging handles. |
| Learn | Accumulates a reference image, following a sequence of operations. |
| Load | - |
| Register | Realigns and normalizes the source image. |
| Save | - |

| | |
|---|---|
| SetDegreesOfFreedom | Boolean combination of EDegreesOfFreedom members, that indicates which degrees of freedom are to be considered. |
| SetNormalize | Current normalization mode. |
| SetPan | Sets the panning factor for use in display operations. |
| SetRelativeTolerance | Current tolerance factor to be used for threshold image setup. |
| SetTolerance | Sets the search margin, i.e. the width and height of the search area surrounding the alignment pattern(s). |
| SetZoom | Sets the zooming factor for use in display operations. |

E

C

# hecker::AddPathName

Adds a single file pathname.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void AddPathName(
 const std::string& pathName
 )
```

### Parameters

*pathName*

**NULL** terminated text string containing the file pathname.

# EChecker::Attach

Associates a source image to a checker context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Attach(
 EROIBW8* source
 )
```

### Parameters

*source*

Pointer to the source image.

### Remarks

The source image is used in all consecutive learning/inspection operations.

# EChecker::GetAverage

Global intensity of the mother image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetAverage()
```

### Remarks

Valid in mode ENormalizationMode_Moments only.

# EChecker::BatchLearn

Performs the learning sequence using the specified list of image files.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BatchLearn(
 Euresys::Open_eVision_2_10::ELearningMode mode
 )
```

### Parameters

*mode*

ELearningMode_RmsDeviation or ELearningMode_AbsDeviation, depending on the preferred method of computing the deviations.

# EChecker::GetDegreesOfFreedom

# EChecker::SetDegreesOfFreedom

Boolean combination of EDegreesOfFreedom members, that indicates which degrees of freedom are to be considered.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetDegreesOfFreedom()
```

```
void SetDegreesOfFreedom(OEV_UINT32 un32DegreesOfFreedom)
```

# EChecker::GetDeviation

Global contrast of the mother image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetDeviation()
```

## Remarks

Valid in mode ENormalizationMode_Moments only.

# EChecker::Drag

Moves the relevant ROI by means of its handle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

## Parameters

*x*

New horizontal cursor position.

*y*

New vertical cursor position.

# EChecker::Draw

Draws one of the geometric items that define the EChecker tool.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Draw(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL handles,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL handles,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL handles,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*graphicContext*

Device context of the drawing window.

*drawingMode*

ROI to be drawn, as defined by EDrawingMode.

*handles*

**TRUE** if the dragging handles must be displayed.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the image is displayed in true scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*color*

The color in which to draw the overlay.

# EChecker::DrawWithCurrentPen

Draws one of the geometric items that define the EChecker tool.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL handles,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*graphicContext*

Device context of the drawing window.

*drawingMode*

ROI to be drawn, as defined by EDrawingMode.

*handles*

**TRUE** if the dragging handles must be displayed.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the image is displayed in true scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

# EChecker::EChecker

Constructs an uninitialized checker context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EChecker(
 const EChecker& other
 )
void EChecker(
 )
```

### Parameters

*other*

-

# EChecker::EmptyPathNames

Clears the list of file pathnames.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EmptyPathNames(
 )
```

# EChecker::GetHigh

High threshold image for the adaptive segmentation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EImageBW8* GetHigh()
```

# EChecker::GetHitHandle

Handle currently hit.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::EDragHandle GetHitHandle()
```

# EChecker::GetHitRoi

ROI currently hit.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::ERoiHit GetHitRoi()
```

# EChecker::HitTest

Returns **TRUE** if the cursor is over one of the dragging handles.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL HitTest(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

## Parameters

*x*

Current horizontal cursor position.

*y*

Current vertical cursor position.

### Remarks

In this case, EChecker::HitRoi returns the name of the ROI that has been hit, and EChecker::HitHandle returns the name of the corresponding handle.

# EChecker::Learn

Accumulates a reference image, following a sequence of operations.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Learn(
 Euresys::Open_eVision_2_10::ELearningMode mode
 )
```

### Parameters

*mode*

Current mode of operation in the learning sequence, as defined by ELearningMode.

### Remarks

First the model is reset; then the matching patterns are shown; next a series of images is presented to estimate the average gray levels; then a second series of images is presented to estimate the gray-level variations; finally, the threshold images are generated. A typical sequence with three reference images goes as follows: For standard deviation estimationEChecker.Learn(ELearningMode_Reset); initializes. EChecker.Register(); realigns and normalizes 1st source image. EChecker.Learn(ELearningMode_RmsDeviation); processes 1st image for deviation info. EChecker.Register(); realigns and normalizes 2nd source image. EChecker.Learn(ELearningMode_RmsDeviation); processes 2nd image for deviation info. EChecker.Register(); realigns and normalizes 3rd source image. EChecker.Learn(ELearningMode_RmsDeviation); processes 3rd image for deviation info. For robust deviation estimationEChecker.Learn(ELearningMode_Reset); initializes. EChecker.Register(); realigns and normalizes 1st source image. EChecker.Learn(ELearningMode_Average); processes 1st image for average info. EChecker.Register(); realigns and normalizes 2nd source image. EChecker.Learn(ELearningMode_Average); processes 2nd image for average info. EChecker.Register(); realigns and normalizes 3rd source image. EChecker.Learn(ELearningMode_Average); processes 3rd image for average info. EChecker.Register(); realigns and normalizes 1st source image. EChecker.Learn(ELearningMode_RmsDeviation); processes 1st image for deviation info. EChecker.Register(); realigns and normalizes 2nd source image. EChecker.Learn(ELearningMode_RmsDeviation); processes 2nd image for deviation info. EChecker.Register(); realigns and normalizes 3rd source image. EChecker.Learn(ELearningMode_RmsDeviation); processes 3rd image for deviation info. EChecker.Learn(ELearningMode_Ready); computes the threshold images.

# EChecker::Load

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

-

# EChecker::GetLow

Low threshold image for the adaptive segmentation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EImageBW8* GetLow()
```

# EChecker::GetNormalize

# EChecker::SetNormalize

Current normalization mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::ENormalizationMode GetNormalize()
void SetNormalize(Euresys::Open_eVision_2_10::ENormalizationMode eNormalize)
```

# EChecker::GetNumAverageSamples

Number of samples that were accumulated in the "average" phase of the training.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumAverageSamples()
```

# EChecker::GetNumDeviationSamples

Number of samples that were accumulated in the "deviation" phase of the training.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumDeviationSamples()
```

# EChecker::GetPanX

Current horizontal panning factor for use in display operations.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetPanX()
```

# EChecker::GetPanY

Current vertical panning factor for use in display operations.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetPanY()
```

# EChecker::Register

Realigns and normalizes the source image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Register(
 )
```

### Remarks

Only the inspected ROI is processed. The first time this function is called, the current pattern ROI are used to define the search patterns. After registration, public member EChecker::Registered contains the realigned, normalized contents of the inspected ROI.

# EChecker::GetRegistered

Represents the source image, after it has been aligned with the reference image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageBW8* GetRegistered()
```

# EChecker::GetRelativeTolerance

# EChecker::SetRelativeTolerance

Current tolerance factor to be used for threshold image setup.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
float GetRelativeTolerance()

void SetRelativeTolerance(float f32RelativeTolerance)
```

# EChecker::Save

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

-

# EChecker::SetPan

Sets the panning factor for use in display operations.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void SetPan(
 float panX,
 float panY
 )
```

### Parameters

*panX*

Horizontal panning factor.

*panY*

Vertical panning factor.

# EChecker::SetTolerance

Sets the search margin, i.e. the width and height of the search area surrounding the alignment pattern(s).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetTolerance(
 OEV_UINT32 toleranceX,
 OEV_UINT32 toleranceY
 )
```

### Parameters

*toleranceX*

Horizontal search tolerance, in pixels.

*toleranceY*

Vertical search tolerance, in pixels.

# EChecker::SetZoom

Sets the zooming factor for use in display operations.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetZoom(
 float zoom
 )
void SetZoom(
 float zoomX,
 float zoomY
 )
```

### Parameters

*zoom*

Magnification factor for zooming in or out in the horizontal and vertical directions (isotropic scaling).

*zoomX*

Magnification factor for zooming in or out in the horizontal direction.

*zoomY*

Magnification factor for zooming in or out in the vertical direction.

# EChecker::GetToleranceX

Current horizontal search tolerance, in pixels.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetToleranceX()
```

# EChecker::GetToleranceY

Current vertical search tolerance, in pixels.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetToleranceY()
```

# EChecker::GetZoomX

Current horizontal zooming factor for use in display operations.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetZoomX()
```

# EChecker::GetZoomY

Current vertical zooming factor for use in display operations.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetZoomY()
```

# 4.38. ECircle Class

Represents a model of a circle (or arc) in EasyGauge.

**Base Class:** EFrame

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| CopyTo | Copies all the data of the current ECircle object into another ECircle object and returns it. |
| Distance | Returns the smallest distance between this ECircle object an another ECircle. |
| ECircle | Constructs a ECircle object. |
| GetAmplitude | Angular amplitude of the ECircle object. |
| GetApex | Apex point coordinates of a ECircle object. |
| GetApexAngle | Angular position at the apex of a ECircle object. |

| | |
|---|---|
| GetArcLength | Circle arc length of a ECircle object. |
| GetDiameter | Diameter of a ECircle object. |
| GetDirect | Flag indicating whether positive angles correspond to a clockwise or an anticlockwise rotation. |
| GetDistanceBetweenLineAndCircle | Computes the distance between a line and a circle. |
| GetDistanceBetweenPointAndCircle | Computes the distance between a point and a circle. |
| GetEnd | End point coordinates of a ECircle object. |
| GetEndAngle | Angular position of the end of a ECircle object. |
| GetFull | Flag indicating whether the ECircle object is a full circle or not. |
| GetIntersectionOfCircles | Computes the intersections between two circles. Returns the number of intersections and stores the found intersections in the provided point parameters. |
| GetIntersectionOfLineAndCircle | Computes the intersections between a line and circle. Returns the number of intersections and stores the found intersections in the provided point parameters. |

| | |
|---|---|
| GetOrg | Origin point coordinates of a ECircle object. |
| GetOrgAngle | Angular position from where the ECircle object extents. |
| GetPoint | Returns the coordinates of a particular point specified by its location along the circle arc. |
| GetProjectionOfPointOnCircle | Computes the projection of a point on a circle. |
| GetRadius | Radius of a ECircle object. |
| operator= | Copies all the data from another ECircle object into the current ECircle object |
| SetAmplitude | Angular amplitude of the ECircle object. |
| SetDiameter | Diameter of a ECircle object. |
| SetFromCenterAndOrigin | Sets the geometric parameters (nominal position and diameter, angular position and amplitude) of an ECircle object. |
| SetFromOriginMiddleEnd | Sets the geometric parameters (nominal position and diameter, angular position and amplitude) of an ECircle object. |
| SetRadius | Radius of a ECircle object. |

# ECircle::GetAmplitude

# ECircle::SetAmplitude

Angular amplitude of the ECircle object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetAmplitude() const
void SetAmplitude(float amplitude)
```

## Remarks

The default value is **360**. A ECircle object is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), the angular position from where it extents, its angular amplitude, and its outline tolerance. The sign of the rotation angle depends whether the field of view is calibrated or not. * When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value. * When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# ECircle::GetApex

Apex point coordinates of a ECircle object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetApex() const
```

# ECircle::GetApexAngle

Angular position at the apex of a ECircle object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetApexAngle() const
```

## Remarks

A ECircle object is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), the angular position from where it extents, its angular amplitude, and its outline tolerance. The sign of the rotation angle depends whether the field of view is calibrated or not. * When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value. * When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# ECircle::GetArcLength

Circle arc length of a ECircle object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetArcLength() const
```

## Remarks

A ECircle object is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), the angular position from where it extents, its angular amplitude, and its outline tolerance.

# ECircle::CopyTo

Copies all the data of the current ECircle object into another ECircle object and returns it.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
ECircle* CopyTo(
 ECircle* other
 )
```

### Parameters

*other*

Pointer to the ECircle object in which the current ECircle object data have to be copied.

### Remarks

In case of a **NULL** pointer, a new ECircle object will be created and returned.

# ECircle::GetDiameter

# ECircle::SetDiameter

Diameter of a ECircle object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetDiameter() const

void SetDiameter(float f32Diameter)
```

### Remarks

A ECircle object is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), the angular position from where it extents, its angular amplitude, and its outline tolerance. By default, the diameter is **100**, which means 100 pixels when the field of view is not calibrated, and 100 physical units in case of a calibrated field of view.

# ECircle::GetDirect

Flag indicating whether positive angles correspond to a clockwise or an anticlockwise rotation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetDirect() const
```

### Remarks

**TRUE** (default) means that angles increase anticlockwisely in a direct coordinate system, and clockwisely in an inverse coordinate system. * When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. * When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards.

# ECircle::Distance

Returns the smallest distance between this ECircle object an another ECircle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float Distance(
 const ECircle& circle
 )
```

### Parameters

*circle*

The other circle

# ECircle::ECircle

Constructs a ECircle object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ECircle(
 )
void ECircle(
 const EPoint& center,
 float diameter,
 float originAngle,
 BOOL direct
 )
void ECircle(
 const EPoint& center,
 const EPoint& origin,
 BOOL direct
 )
void ECircle(
 const EPoint& center,
 float diameter,
 float originAngle,
 float amplitude
 )
void ECircle(
 const EPoint& origin,
 const EPoint& middle,
 const EPoint& end,
 BOOL fullCircle
 )
```

```
void ECircle(
 const ECircle& other
 )
```

### Parameters

*center*

Center coordinates of the circle at its nominal position. The default value is **(0,0)**.

*diameter*

Nominal diameter of the circle. The default value is **100**.

*originAngle*

Nominal angular origin of the circle. The default value is 0.

*direct*

**TRUE** (default) means that angles increase anticlockwisely in a direct coordinate system.

*origin*

Origin point coordinates of the circle.

*amplitude*

Nominal angular amplitude of the circle. The default value is **360**.

*middle*

Middle point coordinates of the circle.

*end*

End point coordinates of the circle.

*fullCircle*

**TRUE** (default) in case of a full turn circle. If **fullCircle** is **FALSE**, **origin** and **end** give the circle's amplitude.

*other*

Another ECircle object to be copied in the new ECircle object.

# ECircle::GetEnd

End point coordinates of a ECircle object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetEnd() const
```

# ECircle::GetEndAngle

Angular position of the end of a ECircle object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetEndAngle() const
```

## Remarks

A ECircle object is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), the angular position from where it extents, its angular amplitude, and its outline tolerance. The sign of the rotation angle depends whether the field of view is calibrated or not. * When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value. * When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# ECircle::GetFull

Flag indicating whether the ECircle object is a full circle or not.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetFull() const
```

### Remarks

By default (**TRUE**), the ECircle object is a full circle.

# ECircle::GetDistanceBetweenLineAndCircle

Computes the distance between a line and a circle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetDistanceBetweenLineAndCircle(
 const ELine& line,
 const ECircle& circle,
 bool limited
 )
```

### Parameters

*line*

The line.

*circle*

The circle.

*limited*

Indicates if the line and circle parameters should be considered as infinite lines and full circles or as a segments and arcs.

# ECircle::GetDistanceBetweenPointAndCircle

Computes the distance between a point and a circle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetDistanceBetweenPointAndCircle(
 const EPoint& pt,
 const ECircle& circle,
 bool limited
 )
```

### Parameters

*pt*

The point.

*circle*

The circle.

*limited*

Indicates if the circle parameter should be considered as a full circle or as an arc.

# ECircle::GetIntersectionOfCircles

Computes the intersections between two circles. Returns the number of intersections and stores the found intersections in the provided point parameters.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetIntersectionOfCircles(
 const ECircle& circle1,
 const ECircle& circle2,
 EPoint& intersection1,
 EPoint& intersection2,
 bool limited
 )
```

### Parameters

*circle1*

The first circle

*circle2*

The second circle

*intersection1*

The first intersection

*intersection2*

The second intersection

*limited*

Indicates if the circle parameters should be considered as full circles or as arcs.

### Remarks

The function returns the number of intersections found. It will return -1 if the two circles are overlapping.

# ECircle::GetIntersectionOfLineAndCircle

Computes the intersections between a line and circle. Returns the number of intersections and stores the found intersections in the provided point parameters.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetIntersectionOfLineAndCircle(
 const ELine& line,
 const ECircle& circle,
 EPoint& intersection1,
 EPoint& intersection2,
 bool limited
 )
```

### Parameters

*line*

The line

*circle*

The circle

*intersection1*

The first intersection

*intersection2*

The second intersection

*limited*

Indicates if the line and circle parameters should be considered as infinite lines and full circles or as a segments and arcs.

### Remarks

The function returns the number of intersections found.

# ECircle::GetPoint

Returns the coordinates of a particular point specified by its location along the circle arc.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetPoint(
 float fraction
 )
```

### Parameters

*fraction*

Point location expressed as a fraction of the circle arc (range **[-1, +1]**).

# ECircle::GetProjectionOfPointOnCircle

Computes the projection of a point on a circle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
EPoint GetProjectionOfPointOnCircle(
 const EPoint& pt,
 const ECircle& circle
 )
```

### Parameters

*pt*

The point.

*circle*

The circle.

# ECircle::operator=

Copies all the data from another ECircle object into the current ECircle object

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
ECircle& operator=(
 const ECircle& other
 )
```

### Parameters

*other*

ECircle object to be copied

# ECircle::GetOrg

Origin point coordinates of a ECircle object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetOrg() const
```

# ECircle::GetOrgAngle

Angular position from where the ECircle object extents.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetOrgAngle() const
```

## Remarks

A ECircle object is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), the angular position from where it extents, its angular amplitude, and its outline tolerance. The sign of the rotation angle depends whether the field of view is calibrated or not. * When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value. * When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# ECircle::GetRadius

# ECircle::SetRadius

Radius of a ECircle object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetRadius() const
void SetRadius(float f32Radius)
```

### Remarks

A ECircle object is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), the angular position from where it extents, its angular amplitude, and its outline tolerance. By default, the radius is **50**, which means 50 pixels when the field of view is not calibrated, and 50 physical units in case of a calibrated field of view.

# ECircle::SetFromCenterAndOrigin

Sets the geometric parameters (nominal position and diameter, angular position and amplitude) of an ECircle object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromCenterAndOrigin(
 const EPoint& center,
 const EPoint& origin,
 BOOL direct
 )
```

### Parameters

*center*

Center coordinates of the circle at its nominal position. The default value is **(0,0)**.

*origin*

Origin point coordinates of the circle.

*direct*

**TRUE** (default) means that angles increase anticlockwisely in a direct coordinate system.

# ECircle::SetFromOriginMiddleEnd

Sets the geometric parameters (nominal position and diameter, angular position and amplitude) of an ECircle object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromOriginMiddleEnd(
 const EPoint& origin,
 const EPoint& middle,
 const EPoint& end,
 BOOL fullCircle
 )
```

### Parameters

*origin*

Origin point coordinates of the circle.

*middle*

Middle point coordinates of the circle.

*end*

End point coordinates of the circle.

*fullCircle*

**TRUE** (default) in case of a full turn circle. If **fullCircle** is **FALSE**, **origin** and **end** give the circle's amplitude.

# 4.39. ECircleGauge Class

Manages a circle fitting gauge.

**Base Class:** ECircleShape

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddSkipRange | Adds an item to the set of skip ranges and returns the index of the newly added range. |
| CopyTo | Copies all the data of the current ECircleGauge object into another ECircleGauge object, and returns it. |
| DisableInnerFiltering | Disables inner sampled point filtering. |
| Drag | Moves a handle to a new position and updates the position parameters of the gauge. |
| Draw | Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode. |
| DrawWithCurrentPen | Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode. |
| ECircleGauge | Constructs a circle measurement context. |
| GetAverageDistance | Average distance between the sampled points and the fitted model. |

| | |
|---|---|
| GetFilteringThreshold | Relative filtering threshold, that is the fraction of the average distance between the sampled points and the fitted model above which a point is filtered out. |
| GetHVConstraint | Status of the restriction on the orientation of the point location gauge or model fitting sample paths. |
| GetInnerFilteringEnabled | Getter method for the **GetInnerFilteringEnabled** property. This property is the flag indicating if the inner sampled point filtering is enabled (**TRUE**), or not. |
| GetInnerFilteringThreshold | Sampled point inner filtering threshold. |
| GetMeasuredCircle | Information pertaining to the fitted circle. |
| GetMeasuredPeak | Returns information pertaining to the default derivative peak, along one of the sample paths of the gauge. |
| GetMeasuredPoint | Returns the coordinates of a sample point, measured along one of the sample paths of the gauge. |
| GetMinAmplitude | - |
| GetMinArea | - |
| GetMinNumFitSamples | Returns the minimum number of samples required for fitting on each side of the shape. |

| | |
|---|---|
| GetNumFilteringPasses | Number of filtering passes for a model fitting operation. |
| GetNumMeasuredPoints | Number of edge-crossing points along the sample path of the gauge that was inspected with the last call to ECircleGauge::MeasureSample. |
| GetNumSamples | Number of sampled points during the model fitting operation. |
| GetNumSkipRanges | Number of skip ranges in the gauge after a call to ECircleGauge::AddSkipRange. |
| GetNumValidSamples | Number of valid sample points remaining after a model fitting operation. |
| GetRectangularSamplingArea | - |
| GetSample | Allows to retrieve the sample points found along the circle. |
| GetSamplingStep | Approximate distance between sampled points during a model fitting operation. |
| GetSkipRange | Allows to retrieve the start and end values of the skip range corresponding to the given index, if it is valid (i.e. if it has previously been created by a call to the ECircleGauge::AddSkipRange method. |

| | |
|---|---|
| GetSmoothing | Number of pixels used for the low-pass filtering operation. |
| GetThickness | Number of parallel segments used to extract the data profile. |
| GetThreshold | - |
| GetTolerance | Searching area half thickness of the circle fitting gauge. |
| GetTransitionChoice | - |
| GetTransitionIndex | - |
| GetTransitionType | - |
| GetType | Shape type. |
| GetValid | Flag indicating if at least one valid transition has been found. |
| HitTest | Checks whether the cursor is positioned over a handle (**TRUE**) or not (**FALSE**). |
| Measure | Triggers the point location or the model fitting operation. |

| | |
|---|---|
| MeasureSample | Computes the sample points along the sample path whose index in the list is given by the **pathIndex** parameter. |
| MeasureWithoutFitting | Triggers the point location without circle fitting operation. |
| operator= | Copies all the data from another ECircleGauge object into the current ECircleGauge object |
| Plot | Draws the profile that is crossed by one of the sample paths of the gauge, as defined by EPlotItem. |
| PlotWithCurrentPen | Draws the profile that is crossed by one of the sample paths of the gauge, as defined by EPlotItem. |
| Process | Triggers the process pertaining to a shape or gauge and all the daughter gauges attach to it. |
| RemoveAllSkipRanges | Removes all the skip ranges previously created by a call to ECircleGauge::AddSkipRange. |
| RemoveSkipRange | After a call to ECircleGauge::AddSkipRange, removes the skip range with the given index. |
| SetActive | Sets the flag indicating whether the gauge is active or not. |

| | |
|---|---|
| SetCircle | Sets the nominal position (center coordinates), diameter, angular origin and amplitude of the circle fitting gauge according to a known circle. |
| SetFilteringThreshold | Relative filtering threshold, that is the fraction of the average distance between the sampled points and the fitted model above which a point is filtered out. |
| SetHVConstraint | Status of the restriction on the orientation of the point location gauge or model fitting sample paths. |
| SetInnerFilteringThreshold | Sampled point inner filtering threshold. |
| SetMinAmplitude | - |
| SetMinArea | - |
| SetMinNumFitSamples | Sets the minimum number of samples required for fitting on each side of the shape. |
| SetNumFilteringPasses | Number of filtering passes for a model fitting operation. |
| SetRectangularSamplingArea | - |
| SetSamplingStep | Approximate distance between sampled points during a model fitting operation. |

| | |
|---|---|
| SetSmoothing | Number of pixels used for the low-pass filtering operation. |
| SetThickness | Number of parallel segments used to extract the data profile. |
| SetThreshold | - |
| SetTolerance | Searching area half thickness of the circle fitting gauge. |
| SetTransitionChoice | - |
| SetTransitionIndex | - |
| SetTransitionType | - |

# ECircleGauge::SetActive

Sets the flag indicating whether the gauge is active or not.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetActive(BOOL active)
```

### Remarks

When complex gauging is required, several gauges can be grouped together. Applying ECircleGauge::Process to the mother gauge or shape triggers the measurement of the whole. Only the active gauges will participate in the process. By default, the gauge is active (**TRUE**).

# ECircleGauge::AddSkipRange

Adds an item to the set of skip ranges and returns the index of the newly added range.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 AddSkipRange(
 const OEV_UINT32 start,
 const OEV_UINT32 end
 )
```

### Parameters

*start*

Beginning of the skip range.

*end*

End of the skip range.

### Remarks

The samples indices between start and end (including the boundaries) will be discarded during the measurement process.
The ECircleGauge::AddSkipRange method allows to define skip ranges in an ECircleGauge. This means that, at measure time, samples belonging to these ranges will not be taken into account.
A sample may belong to more than one skip range; to be discarded, a sample has to pertain to at least one range.
Moreover, the skip ranges are allowed to overlap one another.
The range is allowed to be reversed (i.e. end is not required to be greater than start).
Also, start and end are not required to reference valid indices at the time of the call (i.e. the range may lie outside of the current return value for ECircleGauge::NumSamples).

# ECircleGauge::GetAverageDistance

Average distance between the sampled points and the fitted model.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetAverageDistance()
```

## Remarks

Irrelevant in case of a point location operation.

# ECircleGauge::SetCircle

Sets the nominal position (center coordinates), diameter, angular origin and amplitude of the circle fitting gauge according to a known circle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetCircle(const ECircle& circle)
```

# ECircleGauge::CopyTo

Copies all the data of the current ECircleGauge object into another ECircleGauge object, and returns it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ECircleGauge* CopyTo(
 ECircleGauge* other,
 BOOL recursive
 )
```

### Parameters

*other*

Pointer to the ECircleGauge object in which the current ECircleGauge object data have to be copied.

*recursive*

**TRUE** if the children gauges have to be copied as well, **FALSE** otherwise.

### Remarks

In case of a **NULL** pointer, a new ECircleGauge object will be created and returned.

# ECircleGauge::DisableInnerFiltering

Disables inner sampled point filtering.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DisableInnerFiltering(
 )
```

### Remarks

The inner sampled point filtering is activated as soon as the corresponding ECircleGauge::InnerFilteringThreshold is set.

# ECircleGauge::Drag

Moves a handle to a new position and updates the position parameters of the gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Cursor current X coordinate.

*y*

Cursor current Y coordinate.

# ECircleGauge::Draw

Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )

void Draw(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

```
void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

*color*

The color in which to draw the overlay.

# ECircleGauge::DrawWithCurrentPen

Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

# ECircleGauge::ECircleGauge

Constructs a circle measurement context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ECircleGauge(
 )
void ECircleGauge(
 const ECircleGauge& other
 )
```

## Parameters

*other*

Another ECircleGauge object to be copied in the new ECircleGauge object.

## Remarks

With the default constructor, all the parameters are initialized to their respective default values. With the copy constructor, the circle measurement context is based on a pre-existing ECircleGauge object. The gauge children are also copied. Hierarchy copying through a copy constructor is always recursive. To disable this recursion, use instead the ECircleGauge::CopyTo method.

# ECircleGauge::GetFilteringThreshold

# ECircleGauge::SetFilteringThreshold

Relative filtering threshold, that is the fraction of the average distance between the sampled points and the fitted model above which a point is filtered out.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetFilteringThreshold()
void SetFilteringThreshold(float f32FilteringThreshold)
```

### Remarks

Irrelevant in case of a point location operation.
During a model fitting operation, the "filtering" process can be invoked to remove outliers, i.e. points that were located significantly far away from the fitted model (so that their position is dubious).

# ECircleGauge::GetMeasuredPeak

Returns information pertaining to the default derivative peak, along one of the sample paths of the gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPeak GetMeasuredPeak(
 OEV_UINT32 index
 )
```

### Parameters

*index*

This argument must be left unchanged from its default value, i.e. **~0** (**= 0xFFFFFFFF**).

### Remarks

ECircleGauge::GetMeasuredPeak returns the information about the derivative peak that meets the following two requirements: 1. It lies on the sample path inspected with the last call to ECircleGauge::MeasureSample, and 1. It is associated with the edge-crossing point along the latter sample path that is selected by the transition choice parameter (cf. ECircleGauge::TransitionChoice).

**Note.** For this method to succeed, it is necessary to previously call ECircleGauge::MeasureSample.

# ECircleGauge::GetMeasuredPoint

Returns the coordinates of a sample point, measured along one of the sample paths of the gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetMeasuredPoint(
 OEV_UINT32 index
 )
```

## Parameters

*index*

This argument must be left unchanged from its default value, i.e. **~0** (**= 0xFFFFFFFF**).

## Remarks

These coordinates pertain to the world space; they are expressed in the reference frame to which the current ECircleGauge object belongs. The gauging process uses a list of sample points to find the shape position and size that best fit a given image. These sample points are measured along the sample paths defined by the gauge geometry. ECircleGauge::GetMeasuredPoint returns the coordinates of the sample point that meets the following two requirements: 1. It lies on the sample path inspected with the last call to ECircleGauge::MeasureSample, and 1. Among all the sample points along the latter sample path, it is the one selected by the ECircleGauge::TransitionChoice property.

**Note.** For this method to succeed, it is necessary to previously call ECircleGauge::MeasureSample.

# ECircleGauge::GetMinNumFitSamples

Returns the minimum number of samples required for fitting on each side of the shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetMinNumFitSamples(
 OEV_INT32& side0,
 OEV_INT32& side1,
 OEV_INT32& side2,
 OEV_INT32& side3
 )
```

### Parameters

*side0*

Minimum number of samples on the top side of the rectangle.

*side1*

Minimum number of samples on the left side of the rectangle.

*side2*

Minimum number of samples on the bottom side of the rectangle.

*side3*

Minimum number of samples on the right side of the rectangle.

### Remarks

Irrelevant in case of a point location operation.

# ECircleGauge::GetSample

Allows to retrieve the sample points found along the circle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetSample(
 EPoint& pt,
 OEV_UINT32 index
 )
```

```
void GetSample(
 ESamplePoint& pt,
 OEV_UINT32 index
 )
```

### Parameters

*pt*

EPoint structure to receive the position of the sample point.

*index*

The sample index

### Remarks

The method provides the sample point corresponding to the supplied index. The returned value corresponds to the sample validity.

# ECircleGauge::GetSkipRange

Allows to retrieve the start and end values of the skip range corresponding to the given index, if it is valid (i.e. if it has previously been created by a call to the ECircleGauge::AddSkipRange method.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetSkipRange(
 const OEV_UINT32 index,
 OEV_UINT32& start,
 OEV_UINT32& end
 )
```

### Parameters

*index*

Index of the skip range.

*start*

Beginning of the skip range.

*end*

End of the skip range.

### Remarks

Start is guaranteed to be smaller or equal to end.

# ECircleGauge::HitTest

Checks whether the cursor is positioned over a handle (**TRUE**) or not (**FALSE**).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL HitTest(
 BOOL daughters
 )
```

### Parameters

*daughters*

**TRUE** if the daughters gauges handles have to be considered as well.

# ECircleGauge::GetHVConstraint

# ECircleGauge::SetHVConstraint

Status of the restriction on the orientation of the point location gauge or model fitting sample paths.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetHVConstraint()
```

```
void SetHVConstraint(BOOL bHVConstraint)
```

### Remarks

*Sample paths* are the point location gauges placed along the model to be fitted.

# ECircleGauge::GetInnerFilteringEnabled

Getter method for the **GetInnerFilteringEnabled** property. This property is the flag indicating if the inner sampled point filtering is enabled (**TRUE**), or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetInnerFilteringEnabled()
```

### Remarks

The inner sampled point filtering is activated as soon as the corresponding ECircleGauge::InnerFilteringThreshold is set. To disable inner filtering, use the ECircleGauge::DisableInnerFiltering method.

# ECircleGauge::GetInnerFilteringThreshold

# ECircleGauge::SetInnerFilteringThreshold

Sampled point inner filtering threshold.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetInnerFilteringThreshold()
void SetInnerFilteringThreshold(float f32InnerFilteringThreshold)
```

### Remarks

If inner filtering is enabled, the sampled points that have been found inside the measured circle are filtered in regard of their distance to it. If this distance is greater than the threshold, the sampled point is set as invalid, and removed from the measure. This distance is in physical units. The inner sampled point filtering is activated as soon as the corresponding threshold is set. To disable inner filtering, use the ECircleGauge::DisableInnerFiltering method.

# ECircleGauge::Measure

Triggers the point location or the model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Measure(
 EROIBW8* sourceImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image.

### Remarks

When this method is called, and if not enough valid points are detected, then the method returns directly, and the measured gauge is set to the nominal parameters.

# ECircleGauge::GetMeasuredCircle

Information pertaining to the fitted circle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ECircle GetMeasuredCircle()
```

# ECircleGauge::MeasureSample

Computes the sample points along the sample path whose index in the list is given by the **pathIndex** parameter.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void MeasureSample(
 EROIBW8* sourceImage,
 OEV_UINT32 pathIndex
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*pathIndex*

Sample path index.

## Remarks

This method stores its results into a temporary variable inside the ECircleGauge object.

# ECircleGauge::MeasureWithoutFitting

Triggers the point location without circle fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void MeasureWithoutFitting(
 EROIBW8* sourceImage
 )
```

### Parameters

*sourceImage*

Source image.

### Remarks

This method performs the actual measurement for each transition, but does not perform the circle fitting. This means that individual samples will be available through the ECircleGauge::GetSample method, but the gauge position will not be changed. Please note that the filtering will not be performed in this method, since it relies upon the fitting process. The filtering parameters will thus be unused.

# ECircleGauge::GetMinAmplitude

# ECircleGauge::SetMinAmplitude

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
OEV_UINT32 GetMinAmplitude()
void SetMinAmplitude(OEV_UINT32 un32MinAmplitude)
```

# ECircleGauge::GetMinArea

# ECircleGauge::SetMinArea

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetMinArea()
void SetMinArea(OEV_UINT32 un32MinArea)
```

# ECircleGauge::GetNumFilteringPasses

# ECircleGauge::SetNumFilteringPasses

Number of filtering passes for a model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumFilteringPasses()
```

```
void SetNumFilteringPasses(OEV_UINT32 un32NumFilteringPasses)
```

### Remarks

Irrelevant in case of a point location operation.
During a filtering pass, the points that are too distant from the model are discarded.
During a model fitting operation, the "filtering" process can be invoked to remove outliers, i.e. points that were located significantly far away from the fitted model (so that their position is dubious).
By default (the number of filtering passes is 0), the outliers rejection process is disabled.

# ECircleGauge::GetNumMeasuredPoints

Number of edge-crossing points along the sample path of the gauge that was inspected with the last call to ECircleGauge::MeasureSample.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumMeasuredPoints()
```

### Remarks

**Note.** For this method to succeed, it is necessary to previously call ECircleGauge::MeasureSample.

# ECircleGauge::GetNumSamples

Number of sampled points during the model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumSamples() const
```

### Remarks

Irrelevant in case of a point location operation.
After a model fitting operation, a number of points have been fitted along the model. Among them, some are not reliable because of their **Area** value. Among the remaining ones, some are filtered out (**NumFilteringPasses**, **FilteringThreshold**).

# ECircleGauge::GetNumSkipRanges

Number of skip ranges in the gauge after a call to ECircleGauge::AddSkipRange.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumSkipRanges() const
```

# ECircleGauge::GetNumValidSamples

Number of valid sample points remaining after a model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumValidSamples()
```

### Remarks

Irrelevant in case of a point location operation.
After a model fitting operation, a number of points have been fitted along the model. Among them, some are not reliable because of their **Area** value. Among the remaining ones, some are filtered out (**NumFilteringPasses**, **FilteringThreshold**).

# ECircleGauge::operator=

Copies all the data from another ECircleGauge object into the current ECircleGauge object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ECircleGauge& operator=(
 const ECircleGauge& other
 )
```

### Parameters

*other*

ECircleGauge object to be copied

# ECircleGauge::Plot

Draws the profile that is crossed by one of the sample paths of the gauge, as defined by EPlotItem.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void Plot(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )

void Plot(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )

void Plot(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawItems*

Boolean combination of EPlotItem members, that indicates which items must be displayed.

*width*

Width of the plot.

*height*

Height of the plot.

*originX*

Origin point coordinates of the plot along the X axis.

*originY*

Origin point coordinates of the plot along the Y axis.

*color*

The color in which to draw the overlay.

### Remarks

The sample path that is taken into considered is the one inspected with the last call to ECircleGauge::MeasureSample.

**Note.** For this method to succeed, it is necessary to previously call ECircleGauge::MeasureSample.

# ECircleGauge::PlotWithCurrentPen

Draws the profile that is crossed by one of the sample paths of the gauge, as defined by EPlotItem.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void PlotWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawItems*

Boolean combination of EPlotItem members, that indicates which items must be displayed.

*width*

Width of the plot.

*height*

Height of the plot.

*originX*

Origin point coordinates of the plot along the X axis.

*originY*

Origin point coordinates of the plot along the Y axis.

### Remarks

The sample path that is taken into considered is the one inspected with the last call to ECircleGauge::MeasureSample.

**Note.** For this method to succeed, it is necessary to previously call ECircleGauge::MeasureSample.

# ECircleGauge::Process

Triggers the process pertaining to a shape or gauge and all the daughter gauges attach to it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Process(
 EROIBW8* sourceImage,
 BOOL daughters
 )
```

### Parameters

*sourceImage*

Pointer to the source image.

*daughters*

Flag indicating whether the daughters shapes inherit of the same behavior.

### Remarks

When complex gauging is required, several gauges can be grouped together. Applying **Process** to the mother gauge or shape triggers the measurement of the whole. Only the active gauges will participate in the process.

# ECircleGauge::GetRectangularSamplingArea

# ECircleGauge::SetRectangularSamplingArea

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetRectangularSamplingArea()
void SetRectangularSamplingArea(BOOL bRectangularSamplingArea)
```

# ECircleGauge::RemoveAllSkipRanges

Removes all the skip ranges previously created by a call to ECircleGauge::AddSkipRange.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void RemoveAllSkipRanges(
 )
```

# ECircleGauge::RemoveSkipRange

After a call to ECircleGauge::AddSkipRange, removes the skip range with the given index.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void RemoveSkipRange(
 const OEV_UINT32 index
 )
```

## Parameters

*index*

Index of the skip range to remove, as returned by ECircleGauge::AddSkipRange.

# ECircleGauge::GetSamplingStep

# ECircleGauge::SetSamplingStep

Approximate distance between sampled points during a model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSamplingStep()
void SetSamplingStep(float f32SamplingStep)
```

## Remarks

Irrelevant in case of a point location operation.
To fit a model, a series of point location operations are performed along the model. The point location gauges spacing is given by the sampling step.
By default, the sampling step is set to **5**, which means 5 pixels when the field of view is not calibrated, and 5 physical units in case of a calibrated field of view.
Be aware that if the sampling step is too large, the number of sampled points along the model will not be sufficient enough to accurately locate it.

# ECircleGauge::SetMinNumFitSamples

Sets the minimum number of samples required for fitting on each side of the shape.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetMinNumFitSamples(
 OEV_INT32 side0,
 OEV_INT32 side1,
 OEV_INT32 side2,
 OEV_INT32 side3
 )
```

### Parameters

*side0*

Required number of samples to correctly fit the circle. The default value is **3**. It is the only parameter taken into account.

*side1*

Not used.

*side2*

Not used.

*side3*

Not used.

### Remarks

Irrelevant in case of a point location operation. When the ECircleGauge::Measure method is called, and if not enough valid points are detected, then the method returns directly, and the measured gauge is set to the nominal parameters.

# ECircleGauge::GetSmoothing

# ECircleGauge::SetSmoothing

Number of pixels used for the low-pass filtering operation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetSmoothing()
void SetSmoothing(OEV_UINT32 un32Smoothing)
```

### Remarks

To reduce the effect of noise, the profile data can be low-pass filtered along the point location gauge direction.

# ECircleGauge::GetThickness

# ECircleGauge::SetThickness

Number of parallel segments used to extract the data profile.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetThickness()
void SetThickness(OEV_UINT32 un32Thickness)
```

### Remarks

To reduce the effect of noise and/or strengthen a transition, several parallel profiles can be accumulated.

# ECircleGauge::GetThreshold

# ECircleGauge::SetThreshold

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetThreshold()
void SetThreshold(OEV_UINT32 un32Threshold)
```

# ECircleGauge::GetTolerance

# ECircleGauge::SetTolerance

Searching area half thickness of the circle fitting gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetTolerance()
void SetTolerance(float tolerance)
```

## Remarks

A circle fitting gauge is fully defined knowing its nominal position (given by the coordinates of its center), its nominal radius (diameter), the angular position from where it extents, its angular amplitude and its outline tolerance. By default, the searching area thickness of the circle fitting gauge is **20** (2x10), which means 20 pixels when the field of view is not calibrated, and 20 physical units in case of a calibrated field of view.

# ECircleGauge::GetTransitionChoice

# ECircleGauge::SetTransitionChoice

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::ETransitionChoice GetTransitionChoice()
void SetTransitionChoice(Euresys::Open_eVision_2_10::ETransitionChoice
eTransitionChoice)
```

# ECircleGauge::GetTransitionIndex

# ECircleGauge::SetTransitionIndex

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetTransitionIndex()
void SetTransitionIndex(OEV_UINT32 un32TransitionIndex)
```

# ECircleGauge::GetTransitionType

# ECircleGauge::SetTransitionType

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::ETransitionType GetTransitionType()
void SetTransitionType(Euresys::Open_eVision_2_10::ETransitionType eTransitionType)
```

# ECircleGauge::GetType

Shape type.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EShapeType GetType()
```

# ECircleGauge::GetValid

Flag indicating if at least one valid transition has been found.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetValid()
```

## Remarks

A **FALSE** value means that no measurement has been performed. A **TRUE** value means that a transition was found along the sample path inspected with the last call to ECircleGauge::MeasureSample, and thus a point has been measured.

# 4.40. ECircleRegion Class

Manages a complete context for an ERegion shaped like a circle.

**Base Class:** ERegion

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Drag | Moves the specified handle to a new position and updates all placement parameters of the region. |
| ECircleRegion | Constructs an ECircleRegion context. |
| GetCenter | Center of the region |
| GetRadius | Radius of the region |
| HitTest | Detects if the cursor is placed over one of the dragging handles. |
| Load | Loads the ECircleRegion. The given ESerializer must have been created for reading. |

| | |
|---|---|
| operator!= | Checks if this ECircleRegion instance is not strictly equal to another |
| operator= | Assignment operator. |
| operator== | Checks if this ECircleRegion instance is strictly equal to another |
| Save | |
| | Saves the ECircleRegion. The given ESerializer must have been created for writing. |
| Scale | Creates a new region by scaling the ECircleRegion. |
| SetCenter | Center of the region |
| SetRadius | Radius of the region |
| Translate | Creates a new ECircleRegion by translating the ECircleRegion. |

# ECircleRegion::GetCenter

# ECircleRegion::SetCenter

Center of the region

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetCenter() const
void SetCenter(const EPoint& center)
```

# ECircleRegion::Drag

Moves the specified handle to a new position and updates all placement parameters of the region.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 int x,
 int y,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*x*

x-coordinate of the mouse cursor.

*y*

y-coordinate of the mouse cursor.

*zoomX*

Horizontal zoom factor. By default, true scale is used.

*zoomY*

Vertical zoom factor. By default, true scale is used.

*panX*

Horizontal pan offset. By default, no pan is added.

*panY*

Vertical pan offset. By default, no pan is added.

### Remarks

If zooming and/or panning were used when drawing the region, the same values must be used with ECircleRegion::HitTest and ECircleRegion::Drag.

# ECircleRegion::ECircleRegion

Constructs an ECircleRegion context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ECircleRegion(
 )
void ECircleRegion(
 float centerX,
 float centerY,
 float radius
 )
void ECircleRegion(
 const EPoint& center,
 float radius
 )
```

```
void ECircleRegion(
 const EPoint& pt1,
 const EPoint& pt2,
 const EPoint& pt3
 )

void ECircleRegion(
 const ECircle& circle
 )

void ECircleRegion(
 const ECircleRegion& other
 )
```

## Parameters

*centerX*

The abscissa of the center of the ECircleRegion.

*centerY*

The ordinate of the center of the ECircleRegion.

*radius*

The radius of the ECircleRegion.

*center*

The center of the ECircleRegion.

*pt1*

One of the three points defining the ECircleRegion.

*pt2*

One of the three points defining the ECircleRegion.

*pt3*

One of the three points defining the ECircleRegion.

*circle*

The result of an ECircleGauge object.

*other*

ECircleRegion context to copy.

## Remarks

When defining a ECircleRegion, the resulting radius value must not be **0** else an EError is thrown.
When defining a ECircleRegion, the resulting radius value must be small enough so that the region fit in memory.
When defining a ECircleRegion with three points, they must be non aligned else an EError is thrown.

# ECircleRegion::HitTest

Detects if the cursor is placed over one of the dragging handles.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EEditionMode HitTest(
 int x,
 int y,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*x*

x-coordinate of the mouse cursor.

*y*

y-coordinate of the mouse cursor.

*zoomX*

Horizontal zoom factor. By default, true scale is used.

*zoomY*

Vertical zoom factor. By default, true scale is used.

*panX*

Horizontal pan offset. By default, no pan is added.

*panY*

Vertical pan offset. By default, no pan is added.

## Remarks

Returns a handle identifier, as defined by EEditionMode.
If zooming and/or panning were used when drawing the region, the same values must be used with ECircleRegion::HitTest and ECircleRegion::Drag.

# ECircleRegion::Load

Loads the ECircleRegion. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# ECircleRegion::operator!=

Checks if this ECircleRegion instance is not strictly equal to another

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool operator!=(
 const ECircleRegion& other
 )
```

### Parameters

*other*

Reference to the other ECircleRegion instance

# ECircleRegion::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ECircleRegion& operator=(
 const ECircleRegion& other
 )
```

### Parameters

*other*

Reference to the ECircleRegion used for the assignment

# ECircleRegion::operator==

Checks if this ECircleRegion instance is strictly equal to another

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool operator==(
 const ECircleRegion& other
 )
```

### Parameters

*other*

Reference to the other ECircleRegion instance

# ECircleRegion::GetRadius

# ECircleRegion::SetRadius

Radius of the region

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetRadius() const
void SetRadius(float radius)
```

# ECircleRegion::Save

Saves the ECircleRegion. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Save(
 ESerializer* serializer
 )
```

## Parameters

*serializer*

The serializer.

# ECircleRegion::Scale

Creates a new region by scaling the ECircleRegion.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ECircleRegion Scale(
 float scale
 )
EEllipseRegion Scale(
 float scaleX,
 float scaleY
 )
```

### Parameters

*scale*
Isotropic scale.

*scaleX*
Horizontal scale.

*scaleY*
Vertical scale.

# ECircleRegion::Translate

Creates a new ECircleRegion by translating the ECircleRegion.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
ECircleRegion Translate(
 float dx,
 float dy
 )
```

### Parameters

*dx*

Horizontal translation in pixel value

*dy*

Vertical translation in pixel value

# 4.41. ECircleShape Class

-

**Base Class:** EShape

**Derived Class(es):** ECircleGauge

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Closest | Find the daughter shape that is the closest to this shape. To retrieve the closest shape, use EShape::ClosestShape. |
| CopyTo | Copies all the data of the current ECircleShape object into another ECircleShape object, and returns it. |

| | |
|---|---|
| Drag | Moves a handle to a new position and updates the position parameters of the gauge. |
| Draw | Draws a graphical representation of a shape, as defined by EDrawingMode. |
| DrawWithCurrentPen | Draws a graphical representation of a shape, as defined by EDrawingMode. |
| GetAmplitude | Angular amplitude of the ECircleShape object. |
| GetAngle | - |
| GetApex | Apex point coordinates of a ECircleShape object. |
| GetApexAngle | Angular position at the apex of a ECircleShape object. |
| GetArcLength | Circle arc length of a ECircleShape object. |
| GetCenter | - |
| GetCenterX | Abscissa of the origin point of the frame. |
| GetCenterY | Ordinate of the origin point of the frame. |

| | |
|---|---|
| GetCircle | - |
| GetDiameter | - |
| GetDirect | Flag indicating whether positive angles correspond to a clockwise or an anticlockwise rotation. |
| GetEnd | End point coordinates of a ECircleShape object. |
| GetEndAngle | Angular position of the end of a ECircleShape object. |
| GetFull | Flag indicating whether the ECircleShape object is a full circle or not. |
| GetOrg | Origin point coordinates of a ECircleShape object. |
| GetOrgAngle | Angular position from where the ECircleShape object extents. |
| GetPoint | Returns the coordinates of a particular point specified by its location along the circle arc. |
| GetRadius | - |
| GetScale | - |

| | |
|---|---|
| GetType | Shape type. |
| HitTest | Checks if there is a handle under the cursor. |
| operator= | Copies all the data from another ECircleShape object into the current ECircleShape object |
| SetAmplitude | Angular amplitude of the ECircleShape object. |
| SetAngle | - |
| SetCenter | - |
| SetCenterXY | Sets the center coordinates of a ECircleShape object. |
| SetCircle | - |
| SetDiameter | - |
| SetFromCenterAndOrigin | Sets the geometric parameters (nominal position and diameter, angular position and amplitude) of an ECircleShape object. |
| SetFromOriginMiddleEnd | Sets the geometric parameters (nominal position and diameter, angular position and amplitude) of an ECircleShape object. |

| | |
|---|---|
| SetRadius | - |
| SetScale | - |

# ECircleShape::GetAmplitude

# ECircleShape::SetAmplitude

Angular amplitude of the ECircleShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetAmplitude() const
void SetAmplitude(float ampl)
```

## Remarks

The default value is **360**. A ECircleShape object is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), the angular position from where it extents, its angular amplitude, and its outline tolerance. The sign of the rotation angle depends whether the field of view is calibrated or not. * When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value. * When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# ECircleShape::GetAngle

# ECircleShape::SetAngle

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetAngle() const
void SetAngle(float f32Angle)
```

# ECircleShape::GetApex

Apex point coordinates of a ECircleShape object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetApex() const
```

# ECircleShape::GetApexAngle

Angular position at the apex of a ECircleShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetApexAngle() const
```

### Remarks

A ECircleShape object is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), the angular position from where it extents, its angular amplitude, and its outline tolerance. The sign of the rotation angle depends whether the field of view is calibrated or not. * When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value. * When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# ECircleShape::GetArcLength

Circle arc length of a ECircleShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetArcLength() const
```

### Remarks

A ECircleShape object is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), the angular position from where it extents, its angular amplitude, and its outline tolerance.

# ECircleShape::GetCenter

# ECircleShape::SetCenter

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetCenter() const
void SetCenter(const EPoint& center)
```

# ECircleShape::GetCenterX

Abscissa of the origin point of the frame.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetCenterX() const
```

# ECircleShape::GetCenterY

Ordinate of the origin point of the frame.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetCenterY() const
```

# ECircleShape::GetCircle

# ECircleShape::SetCircle

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ECircle GetCircle() const
void SetCircle(const ECircle& circle)
```

# ECircleShape::Closest

Find the daughter shape that is the closest to this shape. To retrieve the closest shape, use EShape::ClosestShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Closest(
 )
```

# ECircleShape::CopyTo

Copies all the data of the current ECircleShape object into another ECircleShape object, and returns it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ECircleShape* CopyTo(
 ECircleShape* dest,
 BOOL bRecursive
 )
```

### Parameters

*dest*

-

*bRecursive*

-

### Remarks

In case of a **NULL** pointer, a new ECircleShape object will be created and returned.

## ECircleShape::GetDiameter

## ECircleShape::SetDiameter

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetDiameter() const
void SetDiameter(float f32Diameter)
```

# ECircleShape::GetDirect

Flag indicating whether positive angles correspond to a clockwise or an anticlockwise rotation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetDirect() const
```

## Remarks

**TRUE** (default) means that angles increase anticlockwisely in a direct coordinate system, and clockwisely in an inverse coordinate system. * When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. * When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards.

# ECircleShape::Drag

Moves a handle to a new position and updates the position parameters of the gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 OEV_INT32 n32CursorX,
 OEV_INT32 n32CursorY
 )
```

## Parameters

*n32CursorX*

-

*n32CursorY*

-

# ECircleShape::Draw

Draws a graphical representation of a shape, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
void Draw(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

*color*

-

# ECircleShape::DrawWithCurrentPen

Draws a graphical representation of a shape, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

# ECircleShape::GetEnd

End point coordinates of a ECircleShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetEnd() const
```

# ECircleShape::GetEndAngle

Angular position of the end of a ECircleShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetEndAngle() const
```

## Remarks

A ECircleShape object is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), the angular position from where it extents, its angular amplitude, and its outline tolerance. The sign of the rotation angle depends whether the field of view is calibrated or not. * When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value. * When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# ECircleShape::GetFull

Flag indicating whether the ECircleShape object is a full circle or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetFull() const
```

## Remarks

By default (**TRUE**), the ECircleShape object is a full circle.

# ECircleShape::GetPoint

Returns the coordinates of a particular point specified by its location along the circle arc.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetPoint(
 float fraction
 )
```

### Parameters

*fraction*

Point location expressed as a fraction of the circle arc (range **[-1, +1]**).

# ECircleShape::HitTest

Checks if there is a handle under the cursor.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL HitTest(
 BOOL bDaughters
 )
```

### Parameters

*bDaughters*

Indicates if the check must be done in the whole hierarchy or just this object.

# ECircleShape::operator=

Copies all the data from another ECircleShape object into the current ECircleShape object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ECircleShape& operator=(
 const ECircleShape& other
 )
```

### Parameters

*other*

ECircleShape object to be copied

# ECircleShape::GetOrg

Origin point coordinates of a ECircleShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetOrg() const
```

# ECircleShape::GetOrgAngle

Angular position from where the ECircleShape object extents.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetOrgAngle() const
```

### Remarks

A ECircleShape object is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), the angular position from where it extents, its angular amplitude, and its outline tolerance. The sign of the rotation angle depends whether the field of view is calibrated or not. * When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value. * When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# ECircleShape::GetRadius

# ECircleShape::SetRadius

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetRadius() const
void SetRadius(float f32Radius)
```

# ECircleShape::GetScale

# ECircleShape::SetScale

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScale() const
void SetScale(float f32Scale)
```

# ECircleShape::SetCenterXY

Sets the center coordinates of a ECircleShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetCenterXY(
 float centerX,
 float centerY
 )
```

## Parameters

*centerX*

Center coordinates of the ECircleShape object.

*centerY*

Center coordinates of the ECircleShape object.

# ECircleShape::SetFromCenterAndOrigin

Sets the geometric parameters (nominal position and diameter, angular position and amplitude) of an ECircleShape object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetFromCenterAndOrigin(
 const EPoint& center,
 const EPoint& origin,
 BOOL direct
 )
```

### Parameters

*center*

Center coordinates of the circle at its nominal position. The default value is **(0,0)**.

*origin*

Origin point coordinates of the circle.

*direct*

**TRUE** (default) means that angles increase anticlockwisely in a direct coordinate system.

# ECircleShape::SetFromOriginMiddleEnd

Sets the geometric parameters (nominal position and diameter, angular position and amplitude) of an ECircleShape object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetFromOriginMiddleEnd(
 const EPoint& origin,
 const EPoint& middle,
 const EPoint& end,
 BOOL fullCircle
 )
```

### Parameters

*origin*

Origin point coordinates of the circle.

*middle*

Middle point coordinates of the circle.

*end*

End point coordinates of the circle.

*fullCircle*

**TRUE** (default) in case of a full turn circle. If **fullCircle** is **FALSE**, **origin** and **end** give the circle's amplitude.

## ECircleShape::GetType

Shape type.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EShapeType GetType()
```

# 4.42. EClassificationDataset Class

EClassificationDataset manages a dataset of labeled images to be used for classification.
The dataset must contain at least two different labels and each label can be assigned to any number of image in the dataset.
Labels are represented by strings. Images can be given to a EClassificationDataset as a string containing the path to a stored image or using a supported Open eVision image structure. Supported structures are 8-bits monochrome (EImageBW8), 16-bits monochrome (EImageBW16), and 24-bits color (EROIC24, EImageC24).
A EClassificationDataset object is also responsible for providing tools to do data augmentation.
Data augmentation is the process of generating new images on-the-fly by applying affine transformations to those already in the dataset. Data augmentation allows a deep neural network to be invariant to the applied transformation without having to capture and label real world images containing those transformations.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

## Methods

| | |
|---|---|
| AddImage | Adds an image with its label to the dataset. The image can be specified by its path on the filesystem (parameter `imagePath`) or by an Open eVision image buffer (parameter `img`).<br>The method returns **-1** if there was an error when inserting the image in the dataset or a numeric identifier greater or equal to **0** that can be used to access and manipulate the image in the dataset. |
| AddImages | Adds all the images present in the directory specified by the parameter path and whose filename matches the filter and associates them to the corresponding label.<br>The method returns the number of images added to the dataset. |
| Clear | Clears the datasets. |
| EClassificationDataset | Constructs a EClassificationDataset object. |
| Export | Exports the dataset and its images to the given directory. An export will create a sub-directory for each label and export the images of the dataset to their corresponding label sub-directories in the PNG format.<br>Finally, a new EClassificationDataset object with relative paths to the images is saved into the given directory. |
| GetChannels | Number of channels of the first image added to the dataset. |
| GetEnableDataAugmentation | Enable data augmentation. |

| | |
|---|---|
| GetEnableHorizontalFlip | Enable horizontal flipping in data augmentation. |
| GetEnableVerticalFlip | Enable vertical flipping in data augmentation. |
| GetGaussianNoiseMaximumStandardDeviation | Sets the Gaussian noise maximum standard deviation.<br>The Gaussian noise is an additive noise sampled from a Gaussian distribution of deviation between EClassificationDataset::GaussianNoiseMinimumStandardDeviation and EClassificationDataset::GaussianNoiseMaximumStandardDeviation (also called the normal distribution).<br>Its value must be superior or equal to EClassificationDataset::GaussianNoiseMinimumStandardDeviation. |
| GetGaussianNoiseMinimumStandardDeviation | Sets the Gaussian noise minimum standard deviation.<br>The Gaussian noise is an additive noise sampled from a Gaussian distribution of deviation between EClassificationDataset::GaussianNoiseMinimumStandardDeviation and EClassificationDataset::GaussianNoiseMaximumStandardDeviation (also called the normal distribution).<br>Its value must be betwteen **0** and EClassificationDataset::GaussianNoiseMaximumStandardDeviation. |
| GetHeight | Height of the first image added to the dataset. |
| GetImage | Gets a copy of the i-th image of the dataset. The user is responsible for clearing the memory of the returned pointer. |
| GetImageLabel | Gets the label of the i-th image of the dataset. |

| | |
|---|---|
| GetImagePath | Gets the path of the i-th image of the dataset. If the image was not given using a path, the method will throw an exception. |
| GetImages | Gets a copy of all images in the dataset. If data augmentation is enabled, the returned images will be augmented versions of the ones in the dataset.<br>The caller is responsible for clearing the memory allocated for each image. |
| GetImagesIndexesWithLabel | Gets a list of index corresponding to the images associated with the given label |
| GetLabel | Gets the i-th label of the dataset (starting from **0** to EClassificationDataset::NumLabels **- 1**) |
| GetLabelWeight | Gets the weight associated to the i-th label of the dataset (starting from **0** to EClassificationDataset::NumLabels **- 1**) |
| GetMaxBrightnessOffset | Maximum absolute brightness offset. Its value must be between **0** and **1**. |
| GetMaxContrastGain | Maximum contrast gain. Its value must be strictly positive and over EClassificationDataset::MinContrastGain. |
| GetMaxGamma | Maximum gamma for gamma correction. Its value must be higher than EClassificationDataset::MinGamma. |

GetMaxHorizontalShear

Maximum absolute horizontal shear.
It is represented as an angle from the vertical direction. Its value must be between **0** and **90°**.

GetMaxHorizontalShift

Maximum horizontal shift for data augmentation.
The horizontal shift will be between **-**EClassificationDataset::MaxHorizontalShift and **+**EClassificationDataset::MaxHorizontalShift.

GetMaxHueOffset

Maximum absolute hue offset. Its value must be between **0** and **180°**.

GetMaxRotationAngle

Maximum rotation angle for data augmentation.
The rotation angle will be between **-**EClassificationDataset::MaxRotationAngle and **+**EClassificationDataset::MaxRotationAngle.

GetMaxSaturationGain

Maximum saturation gain. Its value must be over or equal to EClassificationDataset::MinSaturationGain.

GetMaxScale

Maximum scaling allowed for data augmentation.

GetMaxVerticalShear

Maximum absolute vertical shear.
It is represented as an angle from the horizontal direction. Its value must be between **0** and **90°**.

| | |
|---|---|
| GetMaxVerticalShift | Maximum vertical shift for data augmentation.<br>The vertical shift will be between **-**EClassificationDataset::MaxHorizontalShift and **+**EClassificationDataset::MaxHorizontalShift. |
| GetMinContrastGain | Minimum contrast gain. Its value must be strictly positive and below EClassificationDataset::MaxContrastGain. |
| GetMinGamma | Minimum gamma for gamma correction. Its value must be strictly positive and below EClassificationDataset::MaxGamma. |
| GetMinSaturationGain | Minimum saturation gain. Its value must be strictly positive. |
| GetMinScale | Minimum scaling allowed for data augmentation. |
| GetNumImages | Number of images in the dataset. |
| GetNumLabels | Number of labels in the dataset. |
| GetSaltAndPepperNoiseMaximumDensity | Sets the maximum density of the salt and pepper noise.<br>The salt and pepper noise sets the value of a number of randomly selected (between EClassificationDataset::SaltAndPepperNoiseMinimumDensity and EClassificationDataset::SaltAndPepperNoiseMaximumDensity) pixels to its minimum or maximum value.<br>Its value must be betwteen EClassificationDataset::SaltAndPepperNoiseMinimumDensity and **100**. |

| | |
|---|---|
| GetSaltAndPepperNoiseMinimumDensity | Sets the minimum density of the salt and pepper noise.<br>The salt and pepper noise sets the value of a number of randomly selected (between EClassificationDataset::SaltAndPepperNoiseMinimumDensity and EClassificationDataset::SaltAndPepperNoiseMaximumDensity) pixels to its minimum or maximum value.<br>Its value must be betwteen **0** and EClassificationDataset::SaltAndPepperNoiseMaximumDensity. |
| GetSpeckleNoiseMaximumStandardDeviation | Sets the speckle noise maximum standard deviation.<br>The speckle noise is a multiplicative noise sampled from a Gamma distribution with a mean of 1 and with its standard deviation between EClassificationDataset::SpeckleNoiseMinimumStandardDeviation and EClassificationDataset::SpeckleNoiseMaximumStandardDeviation.<br>Its value must be strictly higher than EClassificationDataset::SpeckleNoiseMinimumStandardDeviation. |
| GetSpeckleNoiseMinimumStandardDeviation | Sets the speckle noise minimum standard deviation.<br>The speckle noise is a multiplicative noise sampled from a Gamma distribution with a mean of 1 and with its standard deviation between EClassificationDataset::SpeckleNoiseMinimumStandardDeviation and EClassificationDataset::SpeckleNoiseMaximumStandardDeviation.<br>Its value must be strictly positive and lower than EClassificationDataset::SpeckleNoiseMaximumStandardDeviation. |
| GetWidth | Width of the first image added to the dataset. |
| Load | Loads a classification dataset from disk. |
| operator= | Assignment operator |

| | |
|---|---|
| Save | Saves a classification dataset to disk, containing the file paths to the images in the dataset and their associated label. |
| Serialize | Serializes the dataset state, which is the file paths to the images in the dataset and their associated label. |
| SetEnableDataAugmentation | Enable data augmentation. |
| SetEnableHorizontalFlip | Enable horizontal flipping in data augmentation. |
| SetEnableVerticalFlip | Enable vertical flipping in data augmentation. |
| SetGaussianNoiseMaximumStandardDeviation | Sets the Gaussian noise maximum standard deviation.<br>The Gaussian noise is an additive noise sampled from a Gaussian distribution of deviation between EClassificationDataset::GaussianNoiseMinimumStandardDeviation and EClassificationDataset::GaussianNoiseMaximumStandardDeviation (also called the normal distribution).<br>Its value must be superior or equal to EClassificationDataset::GaussianNoiseMinimumStandardDeviation. |
| SetGaussianNoiseMinimumStandardDeviation | Sets the Gaussian noise minimum standard deviation.<br>The Gaussian noise is an additive noise sampled from a Gaussian distribution of deviation between EClassificationDataset::GaussianNoiseMinimumStandardDeviation and EClassificationDataset::GaussianNoiseMaximumStandardDeviation (also called the normal distribution).<br>Its value must be betwteen **0** and EClassificationDataset::GaussianNoiseMaximumStandardDeviation. |

| | |
|---|---|
| SetImageLabel | Sets the label of images in the dataset. |
| SetLabel | Sets the i-th label of the dataset (starting from **0** to EClassificationDataset::NumLabels **- 1**). This operation does not add a new label to the dataset but simply renames an existing label. |
| SetLabelWeight | Sets the weight associated to the i-th label of the dataset (starting from **0** to EClassificationDataset::NumLabels **- 1**). This operation does not add a new label. |
| SetMaxBrightnessOffset | Maximum absolute brightness offset. Its value must be between **0** and **1**. |
| SetMaxContrastGain | Maximum contrast gain. Its value must be strictly positive and over EClassificationDataset::MinContrastGain. |
| SetMaxGamma | Maximum gamma for gamma correction. Its value must be higher than EClassificationDataset::MinGamma. |
| SetMaxHorizontalShear | Maximum absolute horizontal shear.<br>It is represented as an angle from the vertical direction. Its value must be between **0** and **90°**. |
| SetMaxHorizontalShift | Maximum horizontal shift for data augmentation.<br>The horizontal shift will be between **-**EClassificationDataset::MaxHorizontalShift and **+**EClassificationDataset::MaxHorizontalShift. |

SetMaxHueOffset

Maximum absolute hue offset. Its value must be between **0** and **180°**.

SetMaxRotationAngle

Maximum rotation angle for data augmentation.
The rotation angle will be between **-**EClassificationDataset::MaxRotationAngle and **+**EClassificationDataset::MaxRotationAngle.

SetMaxSaturationGain

Maximum saturation gain. Its value must be over or equal to EClassificationDataset::MinSaturationGain.

SetMaxScale

Maximum scaling allowed for data augmentation.

SetMaxVerticalShear

Maximum absolute vertical shear.
It is represented as an angle from the horizontal direction. Its value must be between **0** and **90°**.

SetMaxVerticalShift

Maximum vertical shift for data augmentation.
The vertical shift will be between **-**EClassificationDataset::MaxHorizontalShift and **+**EClassificationDataset::MaxHorizontalShift.

SetMinContrastGain

Minimum contrast gain. Its value must be strictly positive and below EClassificationDataset::MaxContrastGain.

SetMinGamma

Minimum gamma for gamma correction. Its value must be strictly positive and below EClassificationDataset::MaxGamma.

| | |
|---|---|
| SetMinSaturationGain | Minimum saturation gain. Its value must be strictly positive. |
| SetMinScale | Minimum scaling allowed for data augmentation. |
| SetSaltAndPepperNoiseMaximumDensity | Sets the maximum density of the salt and pepper noise.<br>The salt and pepper noise sets the value of a number of randomly selected (between EClassificationDataset::SaltAndPepperNoiseMinimumDensity and EClassificationDataset::SaltAndPepperNoiseMaximumDensity) pixels to its minimum or maximum value.<br>Its value must be betwteen EClassificationDataset::SaltAndPepperNoiseMinimumDensity and **100**. |
| SetSaltAndPepperNoiseMinimumDensity | Sets the minimum density of the salt and pepper noise.<br>The salt and pepper noise sets the value of a number of randomly selected (between EClassificationDataset::SaltAndPepperNoiseMinimumDensity and EClassificationDataset::SaltAndPepperNoiseMaximumDensity) pixels to its minimum or maximum value.<br>Its value must be betwteen **0** and EClassificationDataset::SaltAndPepperNoiseMaximumDensity. |
| SetSpeckleNoiseMaximumStandardDeviation | Sets the speckle noise maximum standard deviation.<br>The speckle noise is a multiplicative noise sampled from a Gamma distribution with a mean of 1 and with its standard deviation between EClassificationDataset::SpeckleNoiseMinimumStandardDeviation and EClassificationDataset::SpeckleNoiseMaximumStandardDeviation.<br>Its value must be strictly higher than EClassificationDataset::SpeckleNoiseMinimumStandardDeviation. |

| | |
|---|---|
| SetSpeckleNoiseMinimumStandardDeviation | Sets the speckle noise minimum standard deviation.<br>The speckle noise is a multiplicative noise sampled from a Gamma distribution with a mean of 1 and with its standard deviation between EClassificationDataset::SpeckleNoiseMinimumStandardDeviation and EClassificationDataset::SpeckleNoiseMaximumStandardDeviation.<br>Its value must be strictly positive and lower than EClassificationDataset::SpeckleNoiseMaximumStandardDeviation. |
| SplitDataset | Splits the dataset in two parts to be used for training and validation respectively. |

C

# lassificationDataset::AddImage

Adds an image with its label to the dataset. The image can be specified by its path on the filesystem (parameter `imagePath`) or by an Open eVision image buffer (parameter `img`).
The method returns **-1** if there was an error when inserting the image in the dataset or a numeric identifier greater or equal to **0** that can be used to access and manipulate the image in the dataset.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
int AddImage(
 const std::string& imagePath,
 const std::string& label
 )
int AddImage(
 const EBaseROI& img,
 const std::string& label
 )
```

## Parameters

*imagePath*

The path to an image

*label*

The label

*img*

The image

# EClassificationDataset::AddImages

Adds all the images present in the directory specified by the parameter path and whose filename matches the filter and associates them to the corresponding label.
The method returns the number of images added to the dataset.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
int AddImages(
 const std::string& filter,
 const std::string& label
 )
```

## Parameters

*filter*

A glob filter

*label*

A label.

## Remarks

The filter is a glob pattern. This means the wildcard characters "*" and "?" correspond to "zero or more character" and "a single character" respectively. For example, the filter "*_good_*.png" will match any filename that contains the string "_good_" and has a png extension.

# EClassificationDataset::GetChannels

Number of channels of the first image added to the dataset.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
OEV_UINT32 GetChannels() const
```

# EClassificationDataset::Clear

Clears the datasets.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
void Clear(
 )
```

# EClassificationDataset::EClassificationDataset

Constructs a EClassificationDataset object.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
void EClassificationDataset(
 )
```

```
void EClassificationDataset(
 const EClassificationDataset& other
 )
```

### Parameters

*other*

Reference to the EClassificationDataset object that should be copied

# EClassificationDataset::GetEnableDataAugmentation

# EClassificationDataset::SetEnableDataAugmentation

Enable data augmentation.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
bool GetEnableDataAugmentation() const
void SetEnableDataAugmentation(bool enable)
```

# EClassificationDataset::GetEnableHorizontalFlip

# EClassificationDataset::SetEnableHorizontalFlip

Enable horizontal flipping in data augmentation.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
bool GetEnableHorizontalFlip() const
void SetEnableHorizontalFlip(bool enable)
```

# EClassificationDataset::GetEnableVerticalFlip

# EClassificationDataset::SetEnableVerticalFlip

Enable vertical flipping in data augmentation.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
bool GetEnableVerticalFlip() const
void SetEnableVerticalFlip(bool enable)
```

# EClassificationDataset::Export

Exports the dataset and its images to the given directory. An export will create a sub-directory for each label and export the images of the dataset to their corresponding label sub-directories in the PNG format.
Finally, a new EClassificationDataset object with relative paths to the images is saved into the given directory.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
```

```
void Export(
 const std::string& directory
 )
```

### Parameters

*directory*

A string containing the full path to the directory.

# EClassificationDataset::GetGaussianNoiseMaximumStandardDeviation

# EClassificationDataset::SetGaussianNoiseMaximumStandardDeviation

Sets the Gaussian noise maximum standard deviation.
The Gaussian noise is an additive noise sampled from a Gaussian distribution of deviation between EClassificationDataset::GaussianNoiseMinimumStandardDeviation and EClassificationDataset::GaussianNoiseMaximumStandardDeviation (also called the normal distribution).
Its value must be superior or equal to EClassificationDataset::GaussianNoiseMinimumStandardDeviation.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
float GetGaussianNoiseMaximumStandardDeviation() const
void SetGaussianNoiseMaximumStandardDeviation(float gaussianMaximumDeviation)
```

### Remarks

This noise is computed before the salt and paper noise.

# EClassificationDataset::GetGaussianNoiseMinimumStandardDeviation

# EClassificationDataset::SetGaussianNoiseMinimumStandardDeviation

Sets the Gaussian noise minimum standard deviation.
The Gaussian noise is an additive noise sampled from a Gaussian distribution of deviation between EClassificationDataset::GaussianNoiseMinimumStandardDeviation and EClassificationDataset::GaussianNoiseMaximumStandardDeviation (also called the normal distribution).
Its value must be betwteen **0** and EClassificationDataset::GaussianNoiseMaximumStandardDeviation.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
float GetGaussianNoiseMinimumStandardDeviation() const
void SetGaussianNoiseMinimumStandardDeviation(float gaussianMinimumDeviation)
```

### Remarks

This noise is computed before the salt and paper noise.

# EClassificationDataset::GetImage

Gets a copy of the i-th image of the dataset. The user is responsible for clearing the memory of the returned pointer.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
EBaseROI* GetImage(
 int i
 )
```

### Parameters

*i*

The index of the image.

# EClassificationDataset::GetImageLabel

Gets the label of the i-th image of the dataset.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
std::string GetImageLabel(
 int i
 )
```

### Parameters

*i*

The index of the image for which to get the label.

# EClassificationDataset::GetImagePath

Gets the path of the i-th image of the dataset. If the image was not given using a path, the method will throw an exception.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
std::string GetImagePath(
 int i
 )
```

### Parameters

*i*

The index of the image for which to get the path.

# EClassificationDataset::GetImages

Gets a copy of all images in the dataset. If data augmentation is enabled, the returned images will be augmented versions of the ones in the dataset.
The caller is responsible for clearing the memory allocated for each image.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
std::vector<Euresys::Open_eVision_2_10::EBaseROI*> GetImages(
 )

std::vector<Euresys::Open_eVision_2_10::EBaseROI*> GetImages(
 std::string label
 )
```

### Parameters

*label*

The label.

# EClassificationDataset::GetImagesIndexesWithLabel

Gets a list of index corresponding to the images associated with the given label

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
std::vector<OEV_UINT32> GetImagesIndexesWithLabel(
 const std::string& label
 )
std::vector<OEV_UINT32> GetImagesIndexesWithLabel(
 int labelIndex
 )
```

### Parameters

*label*

The label

*labelIndex*

The index of the label (starting from **0** to EClassificationDataset::NumLabels **- 1**)

# EClassificationDataset::GetLabel

Gets the i-th label of the dataset (starting from **0** to EClassificationDataset::NumLabels **- 1**)

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
std::string GetLabel(
 int i
 )
```

### Parameters

*i*

Label index

# EClassificationDataset::GetLabelWeight

Gets the weight associated to the i-th label of the dataset (starting from **0** to EClassificationDataset::NumLabels **- 1**)

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
float GetLabelWeight(
 int i
 )
```

### Parameters

*i*

Label index

# EClassificationDataset::GetHeight

Height of the first image added to the dataset.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
OEV_UINT32 GetHeight() const
```

# EClassificationDataset::Load

Loads a classification dataset from disk.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
void Load(
 const std::string& path
 )
```

## Parameters

*path*

A string containing the full path to the dataset file.

# EClassificationDataset::GetMaxBrightnessOffset

# EClassificationDataset::SetMaxBrightnessOffset

Maximum absolute brightness offset. Its value must be between **0** and **1**.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
float GetMaxBrightnessOffset() const
void SetMaxBrightnessOffset(float offset)
```

### Remarks

Brightness transformation is performed by adding a value taken between **-**EClassificationDataset::MaxBrightnessOffset and **+**EClassificationDataset::MaxBrightnessOffset to each pixel of the normalized image.

# EClassificationDataset::GetMaxContrastGain

# EClassificationDataset::SetMaxContrastGain

Maximum contrast gain. Its value must be strictly positive and over EClassificationDataset::MinContrastGain.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
float GetMaxContrastGain() const
void SetMaxContrastGain(float val)
```

### Remarks

Contrast transformation is performed by multiplying each mean-centered pixel value by a gain value taken between EClassificationDataset::MinContrastGain and EClassificationDataset::MaxContrastGain. A contrast transformation does not change the overall brightness of an image.

# EClassificationDataset::GetMaxGamma

# EClassificationDataset::SetMaxGamma

Maximum gamma for gamma correction. Its value must be higher than EClassificationDataset::MinGamma.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
float GetMaxGamma() const
void SetMaxGamma(float gamma)
```

## Remarks

Gamma correction transformation is performed by raising each normalized pixel value to the power of gamma with gamma taken between EClassificationDataset::MinGamma and EClassificationDataset::MaxGamma.

# EClassificationDataset::GetMaxHorizontalShear

# EClassificationDataset::SetMaxHorizontalShear

Maximum absolute horizontal shear.
It is represented as an angle from the vertical direction. Its value must be between **0** and **90°**.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
float GetMaxHorizontalShear() const
void SetMaxHorizontalShear(float hShear)
```

# EClassificationDataset::GetMaxHorizontalShift

# EClassificationDataset::SetMaxHorizontalShift

Maximum horizontal shift for data augmentation.
The horizontal shift will be between **-**EClassificationDataset::MaxHorizontalShift and **+**EClassificationDataset::MaxHorizontalShift.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
OEV_INT32 GetMaxHorizontalShift() const
void SetMaxHorizontalShift(OEV_INT32 maxShift)
```

# EClassificationDataset::GetMaxHueOffset

# EClassificationDataset::SetMaxHueOffset

Maximum absolute hue offset. Its value must be between **0** and **180°**.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
float GetMaxHueOffset() const
void SetMaxHueOffset(float hueOffset)
```

### Remarks

The hue is represented as an angle between **0** and **360°**. The hue transformation is performed by rotating the hue of each pixel by a value between **-**EClassificationDataset::MaxHueOffsetand **+**EClassificationDataset::MaxHueOffset. This transformation only works for color images.

# EClassificationDataset::GetMaxRotationAngle

# EClassificationDataset::SetMaxRotationAngle

Maximum rotation angle for data augmentation.
The rotation angle will be between **-**EClassificationDataset::MaxRotationAngle and **+**EClassificationDataset::MaxRotationAngle.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
float GetMaxRotationAngle() const
void SetMaxRotationAngle(float maxAngle)
```

# EClassificationDataset::GetMaxSaturationGain

# EClassificationDataset::SetMaxSaturationGain

Maximum saturation gain. Its value must be over or equal to EClassificationDataset::MinSaturationGain.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
```

```
float GetMaxSaturationGain() const
void SetMaxSaturationGain(float saturationGain)
```

### Remarks

The saturation transformation is performed by multiplying the saturation of each pixel by a value between EClassificationDataset::MinSaturationGain and EClassificationDataset::MaxSaturationGain.

# EClassificationDataset::GetMaxScale

# EClassificationDataset::SetMaxScale

Maximum scaling allowed for data augmentation.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
float GetMaxScale() const
void SetMaxScale(float maxScale)
```

# EClassificationDataset::GetMaxVerticalShear

# EClassificationDataset::SetMaxVerticalShear

Maximum absolute vertical shear.
It is represented as an angle from the horizontal direction. Its value must be between **0** and **90°**.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
float GetMaxVerticalShear() const
void SetMaxVerticalShear(float vShear)
```

# EClassificationDataset::GetMaxVerticalShift

# EClassificationDataset::SetMaxVerticalShift

Maximum vertical shift for data augmentation.
The vertical shift will be between -EClassificationDataset::MaxHorizontalShift and +EClassificationDataset::MaxHorizontalShift.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
OEV_INT32 GetMaxVerticalShift() const
void SetMaxVerticalShift(OEV_INT32 maxShift)
```

# EClassificationDataset::GetMinContrastGain

# EClassificationDataset::SetMinContrastGain

Minimum contrast gain. Its value must be strictly positive and below EClassificationDataset::MaxContrastGain.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
float GetMinContrastGain() const
void SetMinContrastGain(float val)
```

### Remarks

Contrast transformation is performed by multiplying each mean-centered pixel value by a gain value taken between EClassificationDataset::MinContrastGain and EClassificationDataset::MaxContrastGain. A contrast transformation does not change the overall brightness of an image.

# EClassificationDataset::GetMinGamma

# EClassificationDataset::SetMinGamma

Minimum gamma for gamma correction. Its value must be strictly positive and below EClassificationDataset::MaxGamma.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
float GetMinGamma() const
void SetMinGamma(float gamma)
```

### Remarks

Gamma correction transformation is performed by raising each normalized pixel value to the power of gamma with gamma taken between EClassificationDataset::MinGamma and EClassificationDataset::MaxGamma.

# EClassificationDataset::GetMinSaturationGain

# EClassificationDataset::SetMinSaturationGain

Minimum saturation gain. Its value must be strictly positive.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
float GetMinSaturationGain() const
void SetMinSaturationGain(float saturationGain)
```

### Remarks

The saturation transformation is performed by multiplying the saturation of each pixel by a value between EClassificationDataset::MinSaturationGain and EClassificationDataset::MaxSaturationGain.

# EClassificationDataset::GetMinScale

# EClassificationDataset::SetMinScale

Minimum scaling allowed for data augmentation.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
float GetMinScale() const
void SetMinScale(float minScale)
```

# EClassificationDataset::GetNumImages

Number of images in the dataset.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
int GetNumImages() const
```

# EClassificationDataset::GetNumLabels

Number of labels in the dataset.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
int GetNumLabels() const
```

# EClassificationDataset::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
EClassificationDataset& operator=(
 const EClassificationDataset& other
 )
```

### Parameters

*other*

Reference to the EClassificationDataset object used for the assignment

# EClassificationDataset::GetSaltAndPepperNoiseMaximumDensity

# EClassificationDataset::SetSaltAndPepperNoiseMaximumDensity

Sets the maximum density of the salt and pepper noise.
The salt and pepper noise sets the value of a number of randomly selected (between EClassificationDataset::SaltAndPepperNoiseMinimumDensity and EClassificationDataset::SaltAndPepperNoiseMaximumDensity) pixels to its minimum or maximum value.
Its value must be betwteen EClassificationDataset::SaltAndPepperNoiseMinimumDensity and **100**.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
float GetSaltAndPepperNoiseMaximumDensity() const

void SetSaltAndPepperNoiseMaximumDensity(float saltAndPepperMaximumDensity)
```

### Remarks

This noise is computed after all the other noises.

# EClassificationDataset::GetSaltAndPepperNoiseMinimumDensity

# EClassificationDataset::SetSaltAndPepperNoiseMinimumDensity

Sets the minimum density of the salt and pepper noise.
The salt and pepper noise sets the value of a number of randomly selected (between EClassificationDataset::SaltAndPepperNoiseMinimumDensity and EClassificationDataset::SaltAndPepperNoiseMaximumDensity) pixels to its minimum or maximum value.
Its value must be betwteen **0** and EClassificationDataset::SaltAndPepperNoiseMaximumDensity.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
float GetSaltAndPepperNoiseMinimumDensity() const
void SetSaltAndPepperNoiseMinimumDensity(float saltAndPepperMinimumDensity)
```

### Remarks

This noise is computed after all the other noises.

# EClassificationDataset::Save

Saves a classification dataset to disk, containing the file paths to the images in the dataset and their associated label.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
void Save(
 const std::string& path
 )
```

### Parameters

*path*

A string containing the full path to the dataset file.

### Remarks

This method only save the image that were given to this EClassificationDataset instance as EBaseROI pointers.
To obtain a portable EClassificationDataset file, please use EClassificationDataset::Export.

# EClassificationDataset::Serialize

Serializes the dataset state, which is the file paths to the images in the dataset and their associated label.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

Pointer to the ESerializer object that is read from or written to.

# EClassificationDataset::SetImageLabel

Sets the label of images in the dataset.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
void SetImageLabel(
 int i,
 const std::string& label
 )
void SetImageLabel(
 const std::string& filter,
 const std::string& label
 )
```

### Parameters

*i*

The index of the image for which to set the label.

*label*

The label

*filter*

A glob filter

### Remarks

The filter is a glob pattern. This means the wildcard characters "*" and "?" correspond to "zero or more character" and "a single character" respectively. For example, the filter "*_good_*.png" will match any filename that contains the string "_good_" and has a png extension.

# EClassificationDataset::SetLabel

Sets the i-th label of the dataset (starting from **0** to EClassificationDataset::NumLabels **- 1**). This operation does not add a new label to the dataset but simply renames an existing label.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
```

```
void SetLabel(
 int i,
 const std::string& label
 )
```

### Parameters

*i*

Label index

*label*

Replacement label

# EClassificationDataset::SetLabelWeight

Sets the weight associated to the i-th label of the dataset (starting from **0** to EClassificationDataset::NumLabels **- 1**). This operation does not add a new label.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
void SetLabelWeight(
 int i,
 const float& weight
 )
```

### Parameters

*i*

Label index

*weight*

-

# EClassificationDataset::GetSpeckleNoiseMaximumStandardDeviation

# EClassificationDataset::SetSpeckleNoiseMaximumStandardDeviation

Sets the speckle noise maximum standard deviation.
The speckle noise is a multiplicative noise sampled from a Gamma distribution with a mean of 1 and with its standard deviation between EClassificationDataset::SpeckleNoiseMinimumStandardDeviation and EClassificationDataset::SpeckleNoiseMaximumStandardDeviation.
Its value must be strictly higher than EClassificationDataset::SpeckleNoiseMinimumStandardDeviation.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
float GetSpeckleNoiseMaximumStandardDeviation() const
void SetSpeckleNoiseMaximumStandardDeviation(float speckleMaximumDeviation)
```

### Remarks

This noise is computed before the salt and paper noise.

# EClassificationDataset::GetSpeckleNoiseMinimumStandardDeviation

# EClassificationDataset::SetSpeckleNoiseMinimumStandardDeviation

Sets the speckle noise minimum standard deviation.
The speckle noise is a multiplicative noise sampled from a Gamma distribution with a mean of 1 and with its standard deviation between EClassificationDataset::SpeckleNoiseMinimumStandardDeviation and EClassificationDataset::SpeckleNoiseMaximumStandardDeviation.
Its value must be strictly positive and lower than EClassificationDataset::SpeckleNoiseMaximumStandardDeviation.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
float GetSpeckleNoiseMinimumStandardDeviation() const
void SetSpeckleNoiseMinimumStandardDeviation(float speckleMinimumDeviation)
```

### Remarks

This noise is computed before the salt and paper noise.

# EClassificationDataset::SplitDataset

Splits the dataset in two parts to be used for training and validation respectively.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
void SplitDataset(
 EClassificationDataset& d1,
 EClassificationDataset& d2,
 float proportion,
 bool random
 )
```

### Parameters

*d1*

First part of the dataset

*d2*

Second part of the dataset

*proportion*

Proportion of image of each class to put into the first part. The remaining images are put in `d2`

*random*

Randomly sample the images.

# EClassificationDataset::GetWidth

Width of the first image added to the dataset.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
OEV_UINT32 GetWidth() const
```

# 4.43. EClassificationMetrics Class

Collection of metrics used to evaluate the state of an EClassifier.
A metric is a value summarizing a collection of classification results (EClassificationResult). New results can be added to the object individually with EClassificationMetrics::AddResult or collectively with EClassificationMetrics::AddMetrics.
EClassificationMetrics contains the following metrics:
- the accuracy (see EClassificationMetrics::Accuracy) - the error (see EClassificationMetrics::Error)

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

## Methods

| | |
|---|---|
| AddMetrics | Adds the other metrics to the current metrics of this object. |
| AddResult | Adds the given result with the corresponding groundtruth label to the metrics. |
| EClassificationMetrics | Constructs an EClassificationMetrics object. |
| GetAccuracy | The accuracy of the classifier.<br>The accuracy is the number of images that were correctly classified (also called the true positives) over the total number of images that was used to evaluate the classifier. |
| GetConfusion | Confusion value of one label with another.<br>The confusion value of a label with another is the number of images belonging to this label that are classified as belonging to the other label. |

| | |
|---|---|
| GetError | The error of the classifier.<br>The error, which is also called the loss, is the quantity that is minimized during the training of the deep neural network. For classification, the error is the crossentropy. |
| IsValid | Indicates whether the object contains at least one classification result. |
| Load | Loads a classification metric. The given ESerializer must have been created for reading. |
| operator= | Assignment operator |
| Save | Saves a classification metric. The given ESerializer must have been created for writing. |
| Serialize | Serializes the metrics. |

# EClassificationMetrics::GetAccuracy

The accuracy of the classifier.
The accuracy is the number of images that were correctly classified (also called the true positives) over the total number of images that was used to evaluate the classifier.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
float GetAccuracy() const
```

# EClassificationMetrics::AddMetrics

Adds the other metrics to the current metrics of this object.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
void AddMetrics(
 const EClassificationMetrics& other
 )
```

## Parameters

*other*

Classification metrics

# EClassificationMetrics::AddResult

Adds the given result with the corresponding groundtruth label to the metrics.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
void AddResult(
 EClassificationResult& result,
 const std::string& groundtruthLabel
 )
```

### Parameters

*result*

A reference to an EClassificationResult object.

*groundtruthLabel*

The groundtruth label corresponding to the result

# EClassificationMetrics::EClassificationMetrics

Constructs an EClassificationMetrics object.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
void EClassificationMetrics(
 )
void EClassificationMetrics(
 const EClassificationMetrics& other
 )
```

### Parameters

*other*

Reference to the EClassificationMetrics object that should be copied

# EClassificationMetrics::GetError

The error of the classifier.
The error, which is also called the loss, is the quantity that is minimized during the training of the deep neural network. For classification, the error is the crossentropy.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
float GetError() const
```

# EClassificationMetrics::GetConfusion

Confusion value of one label with another.
The confusion value of a label with another is the number of images belonging to this label that are classified as belonging to the other label.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
OEV_UINT32 GetConfusion(
 const std::string& trueClass,
 const std::string& predictedClass
 )
```

### Parameters

*trueClass*

The label for which to obtain the confusion value

*predictedClass*

The label with which there is a confusion

# EClassificationMetrics::IsValid

Indicates whether the object contains at least one classification result.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
bool IsValid(
 )
```

# EClassificationMetrics::Load

Loads a classification metric. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
void Load(
 ESerializer* serializer
 )
```

## Parameters

*serializer*

Pointer to ESerializer created for reading.

# EClassificationMetrics::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
EClassificationMetrics& operator=(
 const EClassificationMetrics& other
 )
```

### Parameters

*other*

Reference to the EClassificationMetrics object used for the assignment

# EClassificationMetrics::Save

Saves a classification metric. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

Pointer to ESerializer created for writing.

# EClassificationMetrics::Serialize

Serializes the metrics.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

Pointer to ESerializer

# 4.44. EClassificationResult Class

An EClassificationResult object represents the result of a classification.
The most probable label and its probability are accessible through the methods EClassificationResult::BestLabel and EClassificationResult::BestProbability.
The probability and ranking of all labels are accessible through the EClassificationResult and EClassificationResult methods.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

## Methods

| | |
|---|---|
| EClassificationResult | Constructs a non-valid EClassificationResult. |
| GetBestLabel | Gets the most probable label. |
| GetBestLabelId | Gets the most probable label id. |
| GetBestProbability | Gets the probability associated with the most probable label |

| | |
|---|---|
| GetProbability | Gets the probability corresponding to the given label. |
| GetRanking | Gets the ranking corresponding to the given label. The ranking goes from **1** (most probable label) to EClassifier::NumLabels (least probable label). |
| IsValid | Indicates whether the result was produced by EClassifier.<br>A default contructed EClassificationResult is not valid. |
| operator= | Assignment operator |

# EClassificationResult::GetBestLabel

Gets the most probable label.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
std::string GetBestLabel() const
```

# EClassificationResult::GetBestLabelId

Gets the most probable label id.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
int GetBestLabelId() const
```

# EClassificationResult::GetBestProbability

Gets the probability associated with the most probable label

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
float GetBestProbability() const
```

# EClassificationResult::EClassificationResult

Constructs a non-valid EClassificationResult.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
void EClassificationResult(
 )

void EClassificationResult(
 const EClassificationResult& other
 )
```

## Parameters

*other*

Reference to the EClassificationResult object that should be copied

# EClassificationResult::GetProbability

Gets the probability corresponding to the given label.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
float GetProbability(
 std::string label
 )
```

### Parameters

*label*

The label

# EClassificationResult::GetRanking

Gets the ranking corresponding to the given label. The ranking goes from **1** (most probable label) to EClassifier::NumLabels (least probable label).

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
int GetRanking(
 std::string label
 )
```

### Parameters

*label*

The label

# EClassificationResult::IsValid

Indicates whether the result was produced by EClassifier.
A default contructed EClassificationResult is not valid.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
bool IsValid(
 )
```

# EClassificationResult::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
EClassificationResult& operator=(
 const EClassificationResult& other
 )
```

## Parameters

*other*

Reference to the EClassificationResult object used for the assignment

# 4.45. EClassifier Class

EClassifier allows to train a classifier using an EClassificationDataset object and classify new images.

As required by Deep Learning techniques, the input image of EClassifier must be of the same format (width, height, number of channels). By default, this format will be the one of the first image added to the dataset used for training unless its width and height is smaller than the minimum width and height supported by the classifier (See EClassifier::MinimumWidth and EClassifier::MinimumHeight). In this case, the input resolution will be the minimum resolution supported by the classifier. The format can also be specified by the EClassifier::Width, EClassifier::Height and EClassifier::Channels. methods.

By default, images that don't satisfy the image format of the classifier are automatically reformatted. This behavior can be controlled through the EClassifier::EnableAutomaticImageReformat method. When the automatic image reformatting is disabled, training or classifying an image that doesn't satisfy the input image format will result in an exception.

Once trained, the input image format cannot be changed.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

## Methods

| | |
|---|---|
| Classify | Classifies images and returns the complete results as an EClassificationResult object.<br>The method throws an exception if the input image does not fulfill the input specification. |
| EClassifier | Constructs a EClassifier object. |
| Evaluate | Evaluates the EClassifier using the given EClassificationDataset. |

GetBatchSize

Batch size. The batch size is the number of images that are processed together during training and batch classification (EClassifier::Classify).
When using multi-GPUs processing (see EClassifier::GPUIndexes), the batch size is the number of images that each GPU will process at once.
A large batch size will increase the processing speed on GPU but also the memory requirements.
The batch size must be bigger or equal to **2** and it is commonly choosen to be a power of 2.

GetBatchSizeForMaximumClassificationSpeed

Computes the batch size that will maximize the classification speed on a GPU.

GetBestIteration

Iteration at which the minimum validation error was reached. After training, the classifier is in the state it was at this best iteration.

GetChannels

Number of channels for input images of the classifier. The number of channels can be either 1 (monochrome image) or 3 (RGB image). By default, this value will be set from the format of the first image added to the training dataset.

GetCurrentTrainingFinishedIterations

Number of iterations that are already finished in the current training.

GetCurrentTrainingNumIterations

Total number of training iterations that will be performed during the current training of the classifier.

| | |
|---|---|
| GetCurrentTrainingProgression | Current training progression as a percentage (between 0 and 1). |
| GetEnableAutomaticImageReformat | Enable automatic image reformat. |
| GetEnableGPU | Enable the use of a GPU for training and classification. |
| GetEnableHistogramEqualization | Enable histogram equalization of all images passing through the classifier. |
| GetGPUIndexes | Indexes of the GPUs to use for computations.<br>Using multiple GPUs is only possible when we can process multiple images at once, i.e. during training (EClassifier::Train) or batch classification (EClassifier::Classify). Thus, when performing a classification for a single image, only the first GPU in the parameter 'indexes' will be used.<br>By default, all the detected GPUs will be used. |
| GetHeatMap | Gets a heatmap associated with the given label. A heatmap is an image that indicates which pixels in the original image best explain the given label. |
| GetHeight | Height for input images of the classifier. By default, this value will be set from the format of the first image of the dataset used for training. |

| | |
|---|---|
| GetImageCacheSize | Size in byte of the image cache. The cache is used during training to store reformatted and normalized images. A correctly sized cache can reduce the hard drive accesses and the preprocessing time for each image. |
| GetLabel | Gets the label from its index. If the classifier is not trained, the method will throw an exception. |
| GetMinimumHeight | Minimum height for input images of the classifier. This value is equal to **128**. |
| GetMinimumWidth | Minimum width for input images of the classifier. This value is equal to **128**. |
| GetNumGPUs | Number of detected GPU. |
| GetNumLabels | Number of labels of this classifier. If the classifier is not trained, the method will throw an exception. |
| GetNumTrainedIterations | Number of iterations that were performed to train this classifier. |
| GetOptimizeBatchSize | Indicates whether to optimize the batch size (see EClassifier::BatchSize) at training to maximize the training speed according to EClassifier::EnableGPU and the available memory. |

| | |
|---|---|
| GetTrainingMetrics | Gets the metrics obtained with the training dataset at the given iteration.<br>The iterations are indexed between **0** and EClassifier::NumTrainedIterations **- 1**. |
| GetValidationMetrics | Gets the metrics obtained with the validation dataset at the given iteration.<br>The iterations are indexed between **0** and EClassifier::NumTrainedIterations **- 1**. |
| GetWidth | Width for input images of the classifier. By default, this value will be set from the format of the first image of the dataset used for training. |
| IsTrained | Tells whether the classifier has been trained. |
| IsTraining | Indicates whether the object is currently training. |
| Load | Loads a classifier. The given ESerializer must have been created for reading. |
| operator= | Assignment operator |
| Save | Saves a classifier. The given ESerializer must have been created for writing. |
| Serialize | Serializes the classifier. |

| | |
|---|---|
| SetBatchSize | Batch size. The batch size is the number of images that are processed together during training and batch classification (EClassifier::Classify).<br>When using multi-GPUs processing (see EClassifier::GPUIndexes), the batch size is the number of images that each GPU will process at once.<br>A large batch size will increase the processing speed on GPU but also the memory requirements.<br>The batch size must be bigger or equal to **2** and it is commonly choosen to be a power of 2. |
| SetChannels | Number of channels for input images of the classifier. The number of channels can be either 1 (monochrome image) or 3 (RGB image). By default, this value will be set from the format of the first image added to the training dataset. |
| SetEnableAutomaticImageReformat | Enable automatic image reformat. |
| SetEnableGPU | Enable the use of a GPU for training and classification. |
| SetEnableHistogramEqualization | Enable histogram equalization of all images passing through the classifier. |
| SetGPUIndexes | Indexes of the GPUs to use for computations.<br>Using multiple GPUs is only possible when we can process multiple images at once, i.e. during training (EClassifier::Train) or batch classification (EClassifier::Classify). Thus, when performing a classification for a single image, only the first GPU in the parameter 'indexes' will be used.<br>By default, all the detected GPUs will be used. |

SetHeight

Height for input images of the classifier. By default, this value will be set from the format of the first image of the dataset used for training.

SetImageCacheSize

Size in byte of the image cache. The cache is used during training to store reformatted and normalized images. A correctly sized cache can reduce the hard drive accesses and the preprocessing time for each image.

SetOptimizeBatchSize

Indicates whether to optimize the batch size (see EClassifier::BatchSize) at training to maximize the training speed according to EClassifier::EnableGPU and the available memory.

SetWidth

Width for input images of the classifier. By default, this value will be set from the format of the first image of the dataset used for training.

StopTraining

Stops training and returns the last completed iteration. If the parameter 'wait' is set to true, the method will wait for the training thread to completely stop. Otherwise, the method will return immediately.

| | |
|---|---|
| Train | Trains the EClassifier with the given dataset for the specified number of iterations.<br>At the end of the training, the classifier is in the state it was at the iteration that gave the minimum validation error. See EClassifier::BestIteration. |
| WaitForIterationCompletion | Waits until an iteration is complete. A call to this method will block the calling thread until a training iteration in the training thread is finished. This method returns the number of trained iterations. |
| WaitForTrainingCompletion | Waits until the training is complete or the timeout is expired. A call to this method will block the calling thread for the shortest time between the timeout and the time it takes for the training to complete.<br>A negative timeout means that the method will wait until the training is complete.<br>The default value is set to **-1**.<br>The method returns the number of trained iterations. |

## EClassifier::GetBatchSize

## EClassifier::SetBatchSize

Batch size. The batch size is the number of images
that are processed together during training and batch classification (EClassifier::Classify).
When using multi-GPUs processing (see EClassifier::GPUIndexes), the batch size is the number of images that each GPU will process at once.
A large batch size will increase the processing speed on GPU but also the memory requirements.
The batch size must be bigger or equal to **2** and it is commonly choosen to be a power of 2.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
OEV_INT32 GetBatchSize() const
```

```
void SetBatchSize(OEV_INT32 size)
```

### Remarks

If EClassifier::OptimizeBatchSize is 'true', then the value of this property can be changed automatically during training.
See also EClassifier::BatchSizeForMaximumClassificationSpeed.

# EClassifier::GetBatchSizeForMaximumClassificationSpeed

Computes the batch size that will maximize the classification speed on a GPU.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
OEV_INT32 GetBatchSizeForMaximumClassificationSpeed() const
```

### Remarks

This value is given as an indication and should not necessarily be used in practice.
You must choose a tradeoff between the overall classification speed (also called the throughput), which is limited by this value, and the time it takes to classify a whole batch (also called the latency), which is minimized by making classifications image per image (i.e. a batch size of 1).
The tradeoff depends on your particular application.

# EClassifier::GetBestIteration

Iteration at which the minimum validation error was reached. After training, the classifier is in the state it was at this best iteration.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
int GetBestIteration() const
```

# EClassifier::GetChannels

# EClassifier::SetChannels

Number of channels for input images of the classifier. The number of channels can be either 1 (monochrome image) or 3 (RGB image). By default, this value will be set from the format of the first image added to the training dataset.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
OEV_UINT32 GetChannels() const
void SetChannels(OEV_UINT32 channel)
```

### Remarks

If the classifier is not trained or the value was not explicitely set, its value will be **0**.

# EClassifier::Classify

Classifies images and returns the complete results as an EClassificationResult object.
The method throws an exception if the input image does not fulfill the input specification.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
EClassificationResult Classify(
 const EBaseROI& img
 )

std::vector<Euresys::Open_eVision_2_10::EasyDeepLearning::EClassificationResult>
Classify(
 std::vector<const Euresys::Open_eVision_2_10::EBaseROI*>& imgList
 )

std::vector<Euresys::Open_eVision_2_10::EasyDeepLearning::EClassificationResult>
Classify(
 std::vector<Euresys::Open_eVision_2_10::EImageBW8>& imgList
 )

std::vector<Euresys::Open_eVision_2_10::EasyDeepLearning::EClassificationResult>
Classify(
 std::vector<Euresys::Open_eVision_2_10::EImageBW16>& imgList
 )

std::vector<Euresys::Open_eVision_2_10::EasyDeepLearning::EClassificationResult>
Classify(
 std::vector<Euresys::Open_eVision_2_10::EImageC24>& imgList
 )
```

### Parameters

*img*

Image to classify

*imgList*

Vector of images to classify

### Remarks

Classifying a set of images is usually faster than classifying each image sequentially.
To maximize the classification speed on a GPU, EClassifier::BatchSize and the size of the set of input images must be equal to the value returned by EClassifier.

# EClassifier::GetCurrentTrainingFinishedIterations

Number of iterations that are already finished in the current training.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
OEV_UINT32 GetCurrentTrainingFinishedIterations()
```

# EClassifier::GetCurrentTrainingNumIterations

Total number of training iterations that will be performed during the current training of the classifier.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
OEV_UINT32 GetCurrentTrainingNumIterations()
```

# EClassifier::GetCurrentTrainingProgression

Current training progression as a percentage (between 0 and 1).

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
float GetCurrentTrainingProgression()
```

# EClassifier::EClassifier

Constructs a EClassifier object.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
void EClassifier(
 )
void EClassifier(
 const EClassifier& other
 )
```

### Parameters

*other*

Reference to the EClassifier object that should be copied

# EClassifier::GetEnableAutomaticImageReformat

# EClassifier::SetEnableAutomaticImageReformat

Enable automatic image reformat.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
bool GetEnableAutomaticImageReformat() const
void SetEnableAutomaticImageReformat(bool val)
```

# EClassifier::GetEnableGPU

# EClassifier::SetEnableGPU

Enable the use of a GPU for training and classification.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
bool GetEnableGPU() const
void SetEnableGPU(bool enable)
```

# EClassifier::GetEnableHistogramEqualization

# EClassifier::SetEnableHistogramEqualization

Enable histogram equalization of all images passing through the classifier.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
bool GetEnableHistogramEqualization() const
void SetEnableHistogramEqualization(bool val)
```

# EClassifier::Evaluate

Evaluates the EClassifier using the given EClassificationDataset.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
```

```
EClassificationMetrics Evaluate(
 EClassificationDataset& dataset
 )
```

### Parameters

*dataset*

EClassificationDataset with which to evaluate the classifier

# EClassifier::GetHeatMap

Gets a heatmap associated with the given label. A heatmap is an image that indicates which pixels in the original image best explain the given label.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
EImageBW8 GetHeatMap(
 const EBaseROI& image,
 const std::string& label
 )
```

### Parameters

*image*

A reference to the image we want to generate the heatmap from.

*label*

A reference to the string representing the label we want to explain for the given image.

# EClassifier::GetLabel

Gets the label from its index. If the classifier is not trained, the method will throw an exception.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
std::string GetLabel(
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index of the label

# EClassifier::GetTrainingMetrics

Gets the metrics obtained with the training dataset at the given iteration.
The iterations are indexed between **0** and EClassifier::NumTrainedIterations **- 1**.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
EClassificationMetrics GetTrainingMetrics(
 int id
 )
```

### Parameters

*id*

The iteration index

# EClassifier::GetValidationMetrics

Gets the metrics obtained with the validation dataset at the given iteration.
The iterations are indexed between **0** and EClassifier::NumTrainedIterations **- 1**.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
EClassificationMetrics GetValidationMetrics(
 int id
 )
```

### Parameters

*id*

The iteration index

# EClassifier::GetGPUIndexes

# EClassifier::SetGPUIndexes

Indexes of the GPUs to use for computations.
Using multiple GPUs is only possible when we can process multiple images at once, i.e. during training (EClassifier::Train) or batch classification (EClassifier::Classify). Thus, when performing a classification for a single image, only the first GPU in the parameter 'indexes' will be used.
By default, all the detected GPUs will be used.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
std::vector<OEV_UINT32> GetGPUIndexes() const
void SetGPUIndexes(const std::vector<OEV_UINT32>& indexes)
```

### Remarks

The GPU are indexed from **0** to EClassifier::NumGPUs **- 1**.

# EClassifier::GetHeight

# EClassifier::SetHeight

Height for input images of the classifier. By default, this value will be set from the format of the first image of the dataset used for training.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
OEV_UINT32 GetHeight() const
void SetHeight(OEV_UINT32 height)
```

## Remarks

If the classifier is not trained or the value was not explicitely set, its value will be **0**.

# EClassifier::GetImageCacheSize

# EClassifier::SetImageCacheSize

Size in byte of the image cache. The cache is used during training to store reformatted and normalized images. A correctly sized cache can reduce the hard drive accesses and the preprocessing time for each image.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
OEV_UINT64 GetImageCacheSize() const
void SetImageCacheSize(OEV_UINT64 size)
```

# EClassifier::IsTrained

Tells whether the classifier has been trained.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
bool IsTrained(
 )
```

### Remarks

When the classifier has been trained, the input image format of the classifier is fixed and can be obtained with the methods EClassifier::Width, EClassifier::Height, EClassifier::Channels, and EClassifier.

# EClassifier::IsTraining

Indicates whether the object is currently training.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
bool IsTraining(
 )
```

# EClassifier::Load

Loads a classifier. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

Pointer to the ESerializer created for reading.

# EClassifier::GetMinimumHeight

Minimum height for input images of the classifier. This value is equal to **128**.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
OEV_UINT32 GetMinimumHeight() const
```

# EClassifier::GetMinimumWidth

Minimum width for input images of the classifier. This value is equal to **128**.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
OEV_UINT32 GetMinimumWidth() const
```

# EClassifier::GetNumGPUs

Number of detected GPU.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
OEV_UINT32 GetNumGPUs() const
```

# EClassifier::GetNumLabels

Number of labels of this classifier. If the classifier is not trained, the method will throw an exception.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
OEV_UINT32 GetNumLabels() const
```

# EClassifier::GetNumTrainedIterations

Number of iterations that were performed to train this classifier.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
OEV_UINT32 GetNumTrainedIterations()
```

### Remarks

This number of iteration may result from the addition of the iterations performed in several calls to EClassifier::Train. An iteration can also be called an epoch.

# EClassifier::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
EClassifier& operator=(
 const EClassifier& other
 )
```

### Parameters

*other*

Reference to the EClassifier object used for the assignment

# EClassifier::GetOptimizeBatchSize

# EClassifier::SetOptimizeBatchSize

Indicates whether to optimize the batch size (see EClassifier::BatchSize) at training to maximize the training speed according to EClassifier::EnableGPU and the available memory.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
bool GetOptimizeBatchSize() const
void SetOptimizeBatchSize(bool optimize)
```

# EClassifier::Save

Saves a classifier. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

Pointer to the ESerializer created for writing.

# EClassifier::Serialize

Serializes the classifier.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
```

```
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

Pointer to ESerializer

# EClassifier::StopTraining

Stops training and returns the last completed iteration. If the parameter 'wait' is set to true, the method will wait for the training thread to completely stop. Otherwise, the method will return immediately.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
int StopTraining(
 bool wait
 )
```

### Parameters

*wait*

Whether to wait for the training to completely stop

# EClassifier::Train

Trains the EClassifier with the given dataset for the specified number of iterations.
At the end of the training, the classifier is in the state it was at the iteration that gave the minimum validation error.
See EClassifier::BestIteration.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
void Train(
 EClassificationDataset& dataset,
 int iterations
 )
void Train(
 EClassificationDataset& trainingDataset,
 EClassificationDataset& validationDataset,
 int iterations
 )
```

### Parameters

*dataset*

EClassificationDataset with which to train and validate the classifier

*iterations*

Number of iterations for training.

*trainingDataset*

EClassificationDataset with which to train the classifier

*validationDataset*

EClassificationDataset with which to validate the classifier

# EClassifier::WaitForIterationCompletion

Waits until an iteration is complete. A call to this method will block the calling thread until a training iteration in the training thread is finished. This method returns the number of trained iterations.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

```
[C++]
int WaitForIterationCompletion(
 )
```

# EClassifier::WaitForTrainingCompletion

Waits until the training is complete or the timeout is expired. A call to this method will block the calling thread for the shortest time between the timeout and the time it takes for the training to complete.
A negative timeout means that the method will wait until the training is complete.
The default value is set to **-1**.
The method returns the number of trained iterations.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
int WaitForTrainingCompletion(
 int timeout
 )
```

### Parameters

*timeout*

Timeout in second

# EClassifier::GetWidth

# EClassifier::SetWidth

Width for input images of the classifier. By default, this value will be set from the format of the first image of the dataset used for training.

**Namespace:** Euresys::Open_eVision_2_10::EasyDeepLearning

[C++]

```
OEV_UINT32 GetWidth() const
```

```
void SetWidth(OEV_UINT32 width)
```

### Remarks

If the classifier is not trained or the value was not explicitely set, its value will be **0**.

# 4.46. ECodedElement Class

This class encapsulates either an object or a hole in an object, in a coded image.

### Remarks

This abstract class provides a large set of methods applicable to a particular coded element. The set includes methods to get the features of a coded element, to draw coded elements, and to render flexible masks.

**Derived Class(es):** EObject EHole

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AsHole | Down-casts the coded element as a hole. |
| AsObject | Down-casts the coded element as an object. |
| ComputeConvexHull | Computes the convex hull of the coded element. |
| ComputeFeretBox | Computes the Feret box at a specific orientation. |
| ComputePixelGrayAverage | Computes the average gray-level value of the pixels of a given image over the coded element. |

| | |
|---|---|
| ComputePixelGrayDeviation | Computes the standard deviation of the gray-level values of a given image over the coded element. |
| ComputePixelGrayVariance | Computes the variance of the gray-level values of a given image over the coded element. |
| ComputePixelMax | Computes the maximum gray level of the pixels of a given image over the coded element. |
| ComputePixelMin | Computes the minimum gray level of the pixels of a given image over the coded element. |
| ComputeWeightedGravityCenter | Computes the gravity center of a given image over the coded element. |
| GetArea | Returns the number of pixels inside the coded element. |
| GetBottomLimit | Returns the highest (integer) Y-coordinate of all the pixels of the coded element. |
| GetBoundingBox | Returns the bounding box of the coded element (Feret box at orientation 0°). |
| GetBoundingBoxCenter | Returns the coordinates of the center of the bounding box of the coded element. |

| | |
|---|---|
| GetBoundingBoxCenterX | Returns the abscissa of the center of the bounding box of the coded element. |
| GetBoundingBoxCenterY | Returns the ordinate of the center of the bounding box of the coded element. |
| GetBoundingBoxHeight | Returns the height of the bounding box (Feret diameter at 90°). |
| GetBoundingBoxWidth | Returns the width of the bounding box (Feret diameter at 0°). |
| GetCentralMoment | Computes the central, two-dimensional moment of order (p,q). |
| GetContour | Returns the coordinates of the starting point of the countour of the coded element. |
| GetContourX | Returns the abscissa of the starting point of the countour of the coded element. |
| GetContourY | Returns the ordinate of the starting point of the countour of the coded element. |
| GetEccentricity | Returns the eccentricity of the ellipse of inertia. |
| GetElementIndex | Returns the index of the coded element. |
| GetEllipseAngle | Returns the angle of the ellipse of inertia. |

| | |
|---|---|
| GetEllipseHeight | Returns the length of the short axis of the ellipse of inertia. |
| GetEllipseWidth | Returns the length of the long axis of the ellipse of inertia. |
| GetFeretBox22Box | Returns the Feret box at orientation 22.5°. |
| GetFeretBox22Center | Returns the coordinates of the center of the Feret box oriented at 22.5°. |
| GetFeretBox22CenterX | Returns the abscissa of the center of the Feret box oriented at 22.5°. |
| GetFeretBox22CenterY | Returns the ordinate of the center of the Feret box oriented at 22.5°. |
| GetFeretBox22Height | Returns the height of the Feret box oriented at 22.5° (Feret diameter at 112.5°). |
| GetFeretBox22Width | Returns the width of the Feret box oriented at 22.5° (Feret diameter at 22.5°). |
| GetFeretBox45Box | Returns the Feret box at orientation 45°. |
| GetFeretBox45Center | Returns the coordinates of the center of the Feret box oriented at 45°. |

| | |
|---|---|
| GetFeretBox45CenterX | Returns the abscissa of the center of the Feret box oriented at 45°. |
| GetFeretBox45CenterY | Returns the ordinate of the center of the Feret box oriented at 45°. |
| GetFeretBox45Height | Returns the height of the Feret box oriented at 45° (Feret diameter at 135°). |
| GetFeretBox45Width | Returns the width of the Feret box oriented at 45° (Feret diameter at 45°). |
| GetFeretBox68Box | Returns the Feret box at orientation 67.5°. |
| GetFeretBox68Center | Returns the coordinates of the center of the Feret box oriented at 67.5°. |
| GetFeretBox68CenterX | Returns the abscissa of the center of the Feret box oriented at 67.5°. |
| GetFeretBox68CenterY | Returns the ordinate of the center of the Feret box oriented at 67.5°. |
| GetFeretBox68Height | Returns the height of the Feret box oriented at 67.5° (Feret diameter at 157.5°). |
| GetFeretBox68Width | Returns the width of the Feret box oriented at 67.5° (Feret diameter at 67.5°). |

| | |
|---|---|
| GetGravityCenter | Returns the gravity center of the coded element. |
| GetGravityCenterX | Returns the abscissa of the gravity center of the coded element. |
| GetGravityCenterY | Returns the ordinate of the gravity center of the coded element. |
| GetLargestRun | Returns the length of the largest run inside the coded element. |
| GetLayerIndex | Returns the index of the layer in the coded image to which the coded element belongs. |
| GetLeftLimit | Returns the lowest (integer) X-coordinate of all the pixels of the coded element. |
| GetMinimumEnclosingRectangle | Returns the Minimum-Area Enclosing Rectangle. |
| GetMinimumEnclosingRectangleAngle | Returns the angle of the Minimum-Area Enclosing Rectangle. |
| GetMinimumEnclosingRectangleCenter | Returns the coordinates of the center of the Minimum-Area Enclosing Rectangle. |
| GetMinimumEnclosingRectangleCenterX | Returns the abscissa of the center of the Minimum-Area Enclosing Rectangle. |

| | |
|---|---|
| GetMinimumEnclosingRectangleCenterY | Returns the ordinate of the center of the Minimum-Area Enclosing Rectangle. |
| GetMinimumEnclosingRectangleHeight | Returns the height of the Minimum-Area Enclosing Rectangle. |
| GetMinimumEnclosingRectangleWidth | Returns the width of the Minimum-Area Enclosing Rectangle. |
| GetMoment | Computes the raw, two-dimensional moment of order (p,q). |
| GetNormalizedCentralMoment | Computes the scale-invariant, central, two-dimensional moment of order (p,q). |
| GetRightLimit | Returns the highest (integer) X-coordinate of all the pixels of the coded element. |
| GetRunCount | Returns the number of runs inside the coded element. |
| GetRunsIterator | Returns an iterator to the runs of the coded element. |
| GetSigmaX | Returns the centered moment of inertia around X (average squared X-deviation). |

| | |
|---|---|
| GetSigmaXX | Returns the centered cross moment of inertia (average X-deviation * Y-deviation). |
| GetSigmaXY | Returns the reduced, centered moment of inertia (around the principal inertia axis). |
| GetSigmaY | Returns the centered moment of inertia around Y (average squared Y-deviation). |
| GetSigmaYY | Returns the reduced, centered moment of inertia (around the secondary inertia axis). |
| GetTopLimit | Returns the lowest (integer) Y-coordinate of all the pixels of the coded element. |
| IsCodedElement | Tests whether the coded element is an object, a hole or a coded element. |
| IsHole | Tests whether the coded element is an object, a hole or a coded element. |
| IsObject | Tests whether the coded element is an object, a hole or a coded element. |

| | |
|---|---|
| RenderMask | Creates a Flexible Mask from the coded element. |

# ECodedElement::GetArea

Returns the number of pixels inside the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetArea()
```

## Remarks

Equivalently, the area corresponds to the sum of the length of the runs of the coded element.

# ECodedElement::AsHole

Down-casts the coded element as a hole.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EHole& AsHole(
 )
```

## Remarks

This method throws an exception if the coded element is in fact an object.

# ECodedElement::AsObject

Down-casts the coded element as an object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EObject& AsObject(
 )
```

## Remarks

This method throws an exception if the coded element is in fact a hole.

# ECodedElement::GetBottomLimit

Returns the highest (integer) Y-coordinate of all the pixels of the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetBottomLimit()
```

## Remarks

For a coded element E, this value is defined as: <latex>\left\lceil \max \left\{ y \mid (\exists x) (x,y) \in E \right\} \right\rceil </latex>

$$\left\lceil \max \left\{ y \mid (\exists x) (x,y) \in E \right\} \right\rceil$$

# ECodedElement::GetBoundingBox

Returns the bounding box of the coded element (Feret box at orientation 0°).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERotatedBoundingBox GetBoundingBox()
```

# ECodedElement::GetBoundingBoxCenter

Returns the coordinates of the center of the bounding box of the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetBoundingBoxCenter()
```

# ECodedElement::GetBoundingBoxCenterX

Returns the abscissa of the center of the bounding box of the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
float GetBoundingBoxCenterX()
```

# ECodedElement::GetBoundingBoxCenterY

Returns the ordinate of the center of the bounding box of the coded element.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetBoundingBoxCenterY()
```

# ECodedElement::GetBoundingBoxHeight

Returns the height of the bounding box (Feret diameter at 90°).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetBoundingBoxHeight()
```

# ECodedElement::GetBoundingBoxWidth

Returns the width of the bounding box (Feret diameter at 0°).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetBoundingBoxWidth()
```

# ECodedElement::ComputeConvexHull

Computes the convex hull of the coded element.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ComputeConvexHull(
 EPathVector& result
 )
```

## Parameters

*result*

The output vector where to store the convex hull.

# ECodedElement::ComputeFeretBox

Computes the Feret box at a specific orientation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
ERotatedBoundingBox ComputeFeretBox(
 float angle
 )
```

### Parameters

*angle*

The orientation of interest (in the current angle units).

# ECodedElement::ComputePixelGrayAverage

Computes the average gray-level value of the pixels of a given image over the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float ComputePixelGrayAverage(
 const EROIBW8& image
 )
```

### Parameters

*image*

The input image.

# ECodedElement::ComputePixelGrayDeviation

Computes the standard deviation of the gray-level values of a given image over the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float ComputePixelGrayDeviation(
 const EROIBW8& image
 )
```

### Parameters

*image*

The input image.

# ECodedElement::ComputePixelGrayVariance

Computes the variance of the gray-level values of a given image over the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
double ComputePixelGrayVariance(
 const EROIBW8& image
 )
```

### Parameters

*image*

The input image.

# ECodedElement::ComputePixelMax

Computes the maximum gray level of the pixels of a given image over the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW8 ComputePixelMax(
 const EROIBW8& image
 )
```

### Parameters

*image*

The input image.

# ECodedElement::ComputePixelMin

Computes the minimum gray level of the pixels of a given image over the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW8 ComputePixelMin(
 const EROIBW8& image
 )
```

### Parameters

*image*

The input image.

# ECodedElement::ComputeWeightedGravityCenter

Computes the gravity center of a given image over the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint ComputeWeightedGravityCenter(
 const EROIBW8& image
 )
```

### Parameters

*image*

The input image.

# ECodedElement::GetContour

Returns the coordinates of the starting point of the countour of the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetContour()
```

### Remarks

More precisely, the leftmost pixel over the topmost row of the coded element is taken into consideration.

# ECodedElement::GetContourX

Returns the abscissa of the starting point of the countour of the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
int GetContourX()
```

# ECodedElement::GetContourY

Returns the ordinate of the starting point of the countour of the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetContourY()
```

# ECodedElement::GetEccentricity

Returns the eccentricity of the ellipse of inertia.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetEccentricity()
```

## Remarks

The ellipse of inertia is defined as the ellipse that has the same second order moments as the original coded element. The eccentricity is zero for circular objects and one for a line-shaped objects.

# ECodedElement::GetElementIndex

Returns the index of the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetElementIndex() const
```

## Remarks

If the coded element is an object, its index is relative to the layer to which it belongs. If the coded element is a hole, its index is relative to its parent object.

# ECodedElement::GetEllipseAngle

Returns the angle of the ellipse of inertia.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetEllipseAngle()
```

## Remarks

The ellipse of inertia is defined as the ellipse that has the same second order moments as the original coded element.

# ECodedElement::GetEllipseHeight

Returns the length of the short axis of the ellipse of inertia.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetEllipseHeight()
```

### Remarks

The ellipse of inertia is defined as the ellipse that has the same second order moments as the original coded element.

## ECodedElement::GetEllipseWidth

Returns the length of the long axis of the ellipse of inertia.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetEllipseWidth()
```

### Remarks

The ellipse of inertia is defined as the ellipse that has the same second order moments as the original coded element.

## ECodedElement::GetFeretBox22Box

Returns the Feret box at orientation 22.5°.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
ERotatedBoundingBox GetFeretBox22Box()
```

# ECodedElement::GetFeretBox22Center

Returns the coordinates of the center of the Feret box oriented at 22.5°.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetFeretBox22Center()
```

# ECodedElement::GetFeretBox22CenterX

Returns the abscissa of the center of the Feret box oriented at 22.5°.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetFeretBox22CenterX()
```

# ECodedElement::GetFeretBox22CenterY

Returns the ordinate of the center of the Feret box oriented at 22.5°.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetFeretBox22CenterY()
```

# ECodedElement::GetFeretBox22Height

Returns the height of the Feret box oriented at 22.5° (Feret diameter at 112.5°).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetFeretBox22Height()
```

# ECodedElement::GetFeretBox22Width

Returns the width of the Feret box oriented at 22.5° (Feret diameter at 22.5°).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetFeretBox22Width()
```

# ECodedElement::GetFeretBox45Box

Returns the Feret box at orientation 45°.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
ERotatedBoundingBox GetFeretBox45Box()
```

# ECodedElement::GetFeretBox45Center

Returns the coordinates of the center of the Feret box oriented at 45°.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetFeretBox45Center()
```

# ECodedElement::GetFeretBox45CenterX

Returns the abscissa of the center of the Feret box oriented at 45°.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetFeretBox45CenterX()
```

# ECodedElement::GetFeretBox45CenterY

Returns the ordinate of the center of the Feret box oriented at 45°.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetFeretBox45CenterY()
```

# ECodedElement::GetFeretBox45Height

Returns the height of the Feret box oriented at 45° (Feret diameter at 135°).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetFeretBox45Height()
```

# ECodedElement::GetFeretBox45Width

Returns the width of the Feret box oriented at 45° (Feret diameter at 45°).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetFeretBox45Width()
```

# ECodedElement::GetFeretBox68Box

Returns the Feret box at orientation 67.5°.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
ERotatedBoundingBox GetFeretBox68Box()
```

# ECodedElement::GetFeretBox68Center

Returns the coordinates of the center of the Feret box oriented at 67.5°.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetFeretBox68Center()
```

# ECodedElement::GetFeretBox68CenterX

Returns the abscissa of the center of the Feret box oriented at 67.5°.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetFeretBox68CenterX()
```

# ECodedElement::GetFeretBox68CenterY

Returns the ordinate of the center of the Feret box oriented at 67.5°.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetFeretBox68CenterY()
```

# ECodedElement::GetFeretBox68Height

Returns the height of the Feret box oriented at 67.5° (Feret diameter at 157.5°).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetFeretBox68Height()
```

# ECodedElement::GetFeretBox68Width

Returns the width of the Feret box oriented at 67.5° (Feret diameter at 67.5°).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetFeretBox68Width()
```

# ECodedElement::GetCentralMoment

Computes the central, two-dimensional moment of order (p,q).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetCentralMoment(
 OEV_UINT32 p,
 OEV_UINT32 q
 )
```

### Parameters

*p*

Order of the moment along the X-axis.

*q*

Order of the moment along the Y-axis.

### Remarks

<latex>\mu_{pq} = \sum_{x} \sum_{y} (x - \bar{x})^p(y - \bar{y})^q I(x,y)</latex>

$$\mu_{pq} = \sum_{x} \sum_{y} (x - \bar{x})^p (y - \bar{y})^q I(x, y)$$

# ECodedElement::GetMoment

Computes the raw, two-dimensional moment of order (p,q).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
double GetMoment(
 OEV_UINT32 p,
 OEV_UINT32 q
 )
```

### Parameters

*p*

Order of the moment along the X-axis.

*q*

Order of the moment along the Y-axis.

### Remarks

$$M_{pq} = \sum_x \sum_y x^p y^q I(x, y)$$

<latex>M_{pq} = \sum_x \sum_y x^p y^q I(x,y)</latex>

# ECodedElement::GetNormalizedCentralMoment

Computes the scale-invariant, central, two-dimensional moment of order (p,q).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetNormalizedCentralMoment(
 OEV_UINT32 p,
 OEV_UINT32 q
 )
```

### Parameters

*p*

Order of the moment along the X-axis.

*q*

Order of the moment along the Y-axis.

### Remarks

$$\eta_{pq} \frac{\mu_{pq}}{\mu_{00}^{\left(1+\frac{p+q}{2}\right)}}$$

<latex>\eta_{pq} = \frac{\mu_{pq}} {\mu_{00}^{\left(1 + \frac{p+q}{2}\right)}}</latex>

# ECodedElement::GetGravityCenter

Returns the gravity center of the coded element.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetGravityCenter()
```

# ECodedElement::GetGravityCenterX

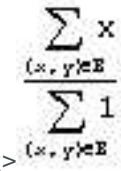
Returns the abscissa of the gravity center of the coded element.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetGravityCenterX()
```

## Remarks

For a coded element E, this value is defined as: <latex>\frac{\sum_{(x,y)\in E} x}{\sum_{(x,y)\in E} 1}</latex> 

# ECodedElement::GetGravityCenterY

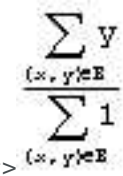
Returns the ordinate of the gravity center of the coded element.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetGravityCenterY()
```

### Remarks

For a coded element E, this value is defined as: <latex>\frac{\sum_{(x,y)\in E} y}{\sum_{(x,y)\in E} 1}</latex> 

# ECodedElement::GetIsCodedElement

Tests whether the coded element is an object, a hole or a coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool IsCodedElement() const
```

# ECodedElement::GetIsHole

Tests whether the coded element is an object, a hole or a coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool IsHole() const
```

# ECodedElement::GetIsObject

Tests whether the coded element is an object, a hole or a coded element.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
bool IsObject() const
```

# ECodedElement::GetLargestRun

Returns the length of the largest run inside the coded element.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetLargestRun()
```

# ECodedElement::GetLayerIndex

Returns the index of the layer in the coded image to which the coded element belongs.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetLayerIndex() const
```

## Remarks

If the coded element is a hole, its layer index is defined as that of its parent object.

# ECodedElement::GetLeftLimit

Returns the lowest (integer) X-coordinate of all the pixels of the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetLeftLimit()
```

## Remarks

For a coded element E, this value is defined as: <latex>\left\lfloor \min \left\{ x \mid (\exists y) (x,y) \in E \right\} \right\rfloor </latex> $\left\lfloor \min \{x \mid (\exists y)\ (x,y) \in E\} \right\rfloor$

# ECodedElement::GetMinimumEnclosingRectangle

Returns the Minimum-Area Enclosing Rectangle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERotatedBoundingBox GetMinimumEnclosingRectangle()
```

## Remarks

The Minimum-Area Enclosing Rectangle is defined as the Feret box with the minimum surface among all the possible orientations.

# ECodedElement::GetMinimumEnclosingRectangleAngle

Returns the angle of the Minimum-Area Enclosing Rectangle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetMinimumEnclosingRectangleAngle()
```

## Remarks

The angle always lies in the range [0 ; pi[.

# ECodedElement::GetMinimumEnclosingRectangleCenter

Returns the coordinates of the center of the Minimum-Area Enclosing Rectangle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetMinimumEnclosingRectangleCenter()
```

# ECodedElement::GetMinimumEnclosingRectangleCenterX

Returns the abscissa of the center of the Minimum-Area Enclosing Rectangle.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetMinimumEnclosingRectangleCenterX()
```

# ECodedElement::GetMinimumEnclosingRectangleCenterY

Returns the ordinate of the center of the Minimum-Area Enclosing Rectangle.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetMinimumEnclosingRectangleCenterY()
```

# ECodedElement::GetMinimumEnclosingRectangleHeight

Returns the height of the Minimum-Area Enclosing Rectangle.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetMinimumEnclosingRectangleHeight()
```

# ECodedElement::GetMinimumEnclosingRectangleWidth

Returns the width of the Minimum-Area Enclosing Rectangle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetMinimumEnclosingRectangleWidth()
```

# ECodedElement::RenderMask

Creates a Flexible Mask from the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RenderMask(
 EROIBW8& destination,
 int offsetX,
 int offsetY
 )
void RenderMask(
 EROIBW8& destination
 )
```

## Parameters

*destination*

The image in which the generated mask will be stored.

*offsetX*

The X-offset that must be applied to bring the zero X-coordinate in the coded image on the first column of the result image (defaults to zero).

*offsetY*

The Y-offset that must be applied to bring the zero Y-coordinate in the coded image on the first row of the result image (defaults to zero).

## Remarks

The size of the result image will not be changed: It must be properly sized beforehand.

# ECodedElement::GetRightLimit

Returns the highest (integer) X-coordinate of all the pixels of the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetRightLimit()
```

## Remarks

For a coded element E, this value is defined as: <latex>\left\lceil \max \left\{ x \mid (\exists y) (x,y) \in E \right\} \right\rceil </latex> $\lceil \max \{x \mid (\exists y) (x,y) \in E\} \rceil$

# ECodedElement::GetRunCount

Returns the number of runs inside the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetRunCount()
```

# ECodedElement::GetRunsIterator

Returns an iterator to the runs of the coded element.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
const EObjectRunsIterator& GetRunsIterator() const
```

# ECodedElement::GetSigmaX

Returns the centered moment of inertia around X (average squared X-deviation).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetSigmaX()
```

# ECodedElement::GetSigmaXX

Returns the centered cross moment of inertia (average X-deviation * Y-deviation).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetSigmaXX()
```

# ECodedElement::GetSigmaXY

Returns the reduced, centered moment of inertia (around the principal inertia axis).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetSigmaXY()
```

# ECodedElement::GetSigmaY

Returns the centered moment of inertia around Y (average squared Y-deviation).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetSigmaY()
```

# ECodedElement::GetSigmaYY

Returns the reduced, centered moment of inertia (around the secondary inertia axis).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetSigmaYY()
```

# ECodedElement::GetTopLimit

Returns the lowest (integer) Y-coordinate of all the pixels of the coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetTopLimit()
```

### Remarks

For a coded element E, this value is defined as: <latex>\left\lfloor \min \left\{ y \mid (\exists x) (x,y) \in E \right\} \right\rfloor </latex> $\lfloor \min \{ y \mid (\exists x) (x,y) \in E \} \rfloor$

# 4.47. ECodedImage Class

This class handles runs, objects and features in EasyObject.

### Remarks

These entities are stored into three separate dynamic lists for efficient storage. This class pertains to the EasyObject legacy API and should not be used for new developments. It has been replaced by ECodedImage2.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddFeat | Adds a feature to the list of features. |
| AnalyseObjects | After an image segmentation (see ECodedImage::BuildObjects), computes the values of given "features", i.e. geometric parameters. |
| BlankFeatures | Resets all values of all features. |
| BuildHoles | Creates holes. |

| | |
|---|---|
| BuildLabeledObjects | Segments an image into connected blobs comprised of pixels of the same class. |
| BuildLabeledRuns | Extracts the runs from the image by comparing adjacent pixel values. |
| BuildObjects | Groups runs to form separated objects (connected blobs), from runs detected by ECodedImage::BuildRuns or from a source image/ROI. |
| BuildRuns | Converts the specified ROI to classes, and extracts the runs from it. |
| DrawObject | Draws the designated object in solid color. |
| DrawObjectFeature | Draws a graphical representation of a feature of the designated object in solid color. |
| DrawObjectFeatureWithCurrentPen | Draws a graphical representation of a feature of the designated object in solid color. |
| DrawObjects | Draws all objects in solid color. |
| DrawObjectsFeature | Draws a graphical representation of a feature. |
| DrawObjectsFeatureWithCurrentPen | Draws a graphical representation of a feature. |

| | |
|---|---|
| DrawObjectsWithCurrentPen | Draws all objects in solid color. |
| DrawObjectWithCurrentPen | Draws the designated object in solid color. |
| ECodedImage | Constructs a void coded image. |
| FeatureAverage | Computes the average of the features of all currently selected objects. |
| FeatureDeviation | Computes the average and standard deviation of the features of all currently selected objects. |
| FeatureMaximum | Computes the maximum of the features of all currently selected objects. |
| FeatureMinimum | Computes the minimum of the features of all currently selected objects. |
| FeatureVariance | Computes the average and variance of the features of all currently selected objects. |
| GetBlackClass | Black class index (below the lower threshold). |
| GetConnexity | Connexity mode, that is how neighboring pixels are considered to belong to the same objects. |

| | |
|---|---|
| GetContinuous | Flag indicating whether the objects have to be built in the continuous mode or in the normal mode. |
| GetCurrentObjData | Returns the data of the current object. |
| GetCurrentObjPtr | Pointer to the current objects list item, or **NULL**. |
| GetCurrentRunData | Returns the data of the current run. |
| GetCurrentRunPtr | Pointer to the current run list item, or **NULL**. |
| GetDrawDiagonals | Flag indicating whether the limit rectangle diagonals must be drawn or not. |
| GetFeatData | Gets the EFeatureData associated to a given feature list item. |
| GetFeatDataSize | Returns the data size of the specified feature. |
| GetFeatDataType | Returns the data type of the specified feature. |
| GetFeatNum | Returns the code number of the specified feature. |
| GetFeatPtrByNum | Returns a pointer to the feature list item for a given feature number, or **NULL**. |

| | |
|---|---|
| GetFeatSize | Returns the size (number of elements) of the feature array for the specified feature. |
| GetFirstHole | Returns a pointer to the first hole related to the specified object. |
| GetFirstObjData | Moves the cursor to the first object and returns the associated data. |
| GetFirstObjPtr | Pointer to the first objects list item, or **NULL**. |
| GetFirstRunData | Moves the cursor to the first run and returns the associated data. |
| GetFirstRunPtr | Pointer to the first run list item, or **NULL**. |
| GetHighColorThreshold | Upper threshold (color) used for image segmentation. |
| GetHighImage | Image used as an adaptive upper threshold. |
| GetHighThreshold | Upper threshold (gray level) used for image segmentation. |
| GetHoleParentObject | Returns a pointer to the real object including the specified hole. |
| GetLastObjData | Moves the cursor to the last object and returns the associated data. |

| | |
|---|---|
| GetLastObjPtr | Pointer to the last objects list item, or **NULL**. |
| GetLastRunData | Moves the cursor to the last run and returns the associated data. |
| GetLastRunPtr | Pointer to the last run list item, or **NULL**. |
| GetLimitAngle | Angle of the skewed bounding box feature, in the current angle unit. |
| GetLowColorThreshold | Lower threshold (color) used for image segmentation. |
| GetLowImage | Image used as an adaptive lower threshold. |
| GetLowThreshold | Lower threshold (gray level) used for image segmentation. |
| GetMaxObjects | Maximum number of objects to look for. |
| GetNeutralClass | Neutral class index (between both thresholds). |
| GetNextHole | Returns a pointer to the hole following the specified hole, in the objects list. |
| GetNextObjData | Moves the cursor to the next object and returns the associated data. |

| | |
|---|---|
| GetNextObjPtr | Returns a pointer to the next objects list item, or **NULL**. |
| GetNextRunData | Moves the cursor to the next run and returns the associated data. |
| GetNextRunPtr | Returns a pointer to the next run list item, or **NULL**. |
| GetNumFeatures | Number of features currently in use. |
| GetNumHoleRuns | Total number of hole runs in the list of object runs. |
| GetNumHoles | Returns the number of holes related to the specified object. |
| GetNumObjectRuns | Returns the number of runs comprised in a given object. |
| GetNumObjects | Number of objects in the coded image. |
| GetNumRuns | Total number of runs in the list of object runs. |
| GetNumSelectedObjects | Number of objects currently selected. |
| GetObjDataPtr | Gets the EObjectData associated to a given objects list item. |

| | |
|---|---|
| GetObjectData | If an object identification number is given, moves the cursor to the object with a given identification number and returns the associated data. If a list item is given, returns the object data associated with an object list item (cf. EListItem). See also ECodedImage::GetCurrentObjData. |
| GetObjectFeature | Allows retrieving the value of a feature of a given object. |
| GetObjFirstRunPtr | Returns a pointer to the first run list item of an object. |
| GetObjLastRunPtr | Returns a pointer to the last run list item of an object. |
| GetObjPtr | Returns a pointer to the given objects list item. |
| GetObjPtrByCoordinates | Returns a pointer to the objects list item that contains the point of given coordinates, or **NULL**. |
| GetObjPtrByPos | Returns a pointer to the objects list item of given absolute position, or **NULL**. |
| GetPreviousObjData | Moves the cursor to the previous object and returns the associated data. |
| GetPreviousObjPtr | Returns a pointer to the previous objects list item, or **NULL**. |

| | |
|---|---|
| GetPreviousRunData | Moves the cursor to the previous run and returns the associated data. |
| GetPreviousRunPtr | Returns a pointer to the previous run list item, or **NULL**. |
| GetRunData | Returns the run data associated to the specified run. |
| GetRunDataPtr | Gets the ERunData associated to a given run list item. |
| GetRunPtr | Returns a pointer to the run list item of given absolute position, or **NULL**. |
| GetRunPtrByCoordinates | Returns a pointer to the run list item that contains the point of given coordinates, or **NULL**. |
| GetThreshold | Threshold mode (gray level) used for image segmentation. |
| GetTrueThreshold | Absolute threshold level, when using a single threshold. |
| GetWhiteClass | White class index (above the upper threshold). |
| IsHole | Returns **TRUE** if the specified object is a hole, **FALSE** otherwise. |
| IsObjectSelected | Sets **bSelected** to **TRUE** if an object is selected. |

| | |
|---|---|
| ObjectConvexHull | Computes the convex hull of an object and stores it in a **EPathVector**. |
| RemoveAllFeats | Deletes all features from the features list. |
| RemoveAllObjects | Deletes all objects from the objects list. |
| RemoveAllRuns | Deletes all runs from the runs list. |
| RemoveHoles | Permanently erases, from the objects list, holes related to the specified object. |
| RemoveObject | Deletes an object from the objects list. |
| RemoveRun | Deletes a run from the runs list. |
| ResetContinuousMode | When the continuous mode is activated, this method resets the sequence of images. |
| SelectAllObjects | Selects all objects. |
| SelectHoles | Selects the holes related to the specified object. |
| SelectObject | Selects an object. |

| | |
|---|---|
| SelectObjectsUsingFeature | Selects or deselects objects, according to the value of a specified feature. |
| SelectObjectsUsingPosition | Selects or deselects objects, using a delimiting rectangle. |
| SetBlackClass | Black class index (below the lower threshold). |
| SetConnexity | Connexity mode, that is how neighboring pixels are considered to belong to the same objects. |
| SetContinuous | Flag indicating whether the objects have to be built in the continuous mode or in the normal mode. |
| SetDrawDiagonals | Flag indicating whether the limit rectangle diagonals must be drawn or not. |
| SetFeatInfo | Sets the appropriate data size and type for a predefined feature. |
| SetFirstRunPtr | Sets the first run list item of an object. |
| SetHighColorThreshold | Upper threshold (color) used for image segmentation. |
| SetHighImage | Image used as an adaptive upper threshold. |
| SetHighThreshold | Upper threshold (gray level) used for image segmentation. |

| | |
|---|---|
| SetLastRunPtr | Sets the last run list item of an object. |
| SetLimitAngle | Angle of the skewed bounding box feature, in the current angle unit. |
| SetLowColorThreshold | Lower threshold (color) used for image segmentation. |
| SetLowImage | Image used as an adaptive lower threshold. |
| SetLowThreshold | Lower threshold (gray level) used for image segmentation. |
| SetMaxObjects | Maximum number of objects to look for. |
| SetNeutralClass | Neutral class index (between both thresholds). |
| SetNumSelectedObjects | Number of objects currently selected. |
| SetThreshold | Threshold mode (gray level) used for image segmentation. |
| SetThresholdImage | Single threshold used for image segmentation. |
| SetWhiteClass | White class index (above the upper threshold). |
| SortObjectsUsingFeature | Sorts objects according to the value of some feature. |

| | |
|---|---|
| UnselectAllObjects | Deselects all objects. |
| UnselectHoles | Unselects the holes related to the specified object. |
| UnselectObject | Deselects an object. |

E

C

# odedImage::AddFeat

Adds a feature to the list of features.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddFeat(
 EFeatureData* feature,
 OEV_INT32 numberOfObjects
 )
```

## Parameters

*feature*

Pointer to an EFeatureData describing the feature.

*numberOfObjects*

Number of objects for which the feature will be stored.

# ECodedImage::AnalyseObjects

After an image segmentation (see ECodedImage::BuildObjects), computes the values of given "features", i.e. geometric parameters.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AnalyseObjects(
 Euresys::Open_eVision_2_10::ELegacyFeature feature1,
 Euresys::Open_eVision_2_10::ELegacyFeature feature2,
 Euresys::Open_eVision_2_10::ELegacyFeature feature3,
 Euresys::Open_eVision_2_10::ELegacyFeature feature4,
 Euresys::Open_eVision_2_10::ELegacyFeature feature5,
 Euresys::Open_eVision_2_10::ELegacyFeature feature6,
 Euresys::Open_eVision_2_10::ELegacyFeature feature7,
 Euresys::Open_eVision_2_10::ELegacyFeature feature8,
 Euresys::Open_eVision_2_10::ELegacyFeature feature9,
 Euresys::Open_eVision_2_10::ELegacyFeature feature10
 )
```

## Parameters

*feature1*

Feature code, as defined by ELegacyFeature.

*feature2*

Feature code, as defined by ELegacyFeature.

*feature3*

Feature code, as defined by ELegacyFeature.

*feature4*

Feature code, as defined by ELegacyFeature.

*feature5*

Feature code, as defined by ELegacyFeature.

*feature6*

Feature code, as defined by ELegacyFeature.

*feature7*

Feature code, as defined by ELegacyFeature.

*feature8*

Feature code, as defined by ELegacyFeature.

*feature9*

Feature code, as defined by ELegacyFeature.

*feature10*

Feature code, as defined by ELegacyFeature.

# ECodedImage::GetBlackClass

# ECodedImage::SetBlackClass

Black class index (below the lower threshold).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT16 GetBlackClass()
void SetBlackClass(OEV_INT16 n16BlackClass)
```

## Remarks

Non zero when the black runs (below the lower threshold) are coded. **0** means "do not code this class". <!-- **1** by default. -->

# ECodedImage::BlankFeatures

Resets all values of all features.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BlankFeatures(
 )
```

# ECodedImage::BuildHoles

Creates holes.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BuildHoles(
 )
void BuildHoles(
 EListItem* object_
 )
```

## Parameters

*object_*

Pointer to the objects list item, for which the holes have to be computed.

## Remarks

If no argument, the holes are related to all the previously selected real objects. If holes already exist (resulting from a previous call to the ECodedImage::BuildHoles function), they will be removed from the objects list before the new hole building. Otherwise, the holes are related only to the specified object. Previously created holes are not removed before the new holes are built. If holes related to **object** have already been constructed, they won't be recreated. If **object** is a hole or is **NULL**, no hole will be built. The newly created holes will be added to the list of the objects found in the image. Building holes requires two preliminary steps: the construction of real objects and the selection of objects on which the hole detection has to be performed. At the end of the object construction, all the objects are selected.

# ECodedImage::BuildLabeledObjects

Segments an image into connected blobs comprised of pixels of the same class.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BuildLabeledObjects(
 EROIBW8* sourceImage
 )
void BuildLabeledObjects(
 EROIBW16* sourceImage
 )
```

### Parameters

*sourceImage*

Pointer to a source ROI.

### Remarks

Uses EBW8 (EBW16) information for class indices, i.e. 255 (65,535) possible classes. Class 0 is not coded. Building objects is the process of grouping pixels from an image to form connected blobs. The pixels are assigned class indices based either on thresholding (ECodedImage::BuildObjects) or on the pixel values themselves (**BuildLabeledObjects**). A blob is a set of connected pixels of the same class.

# ECodedImage::BuildLabeledRuns

Extracts the runs from the image by comparing adjacent pixel values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void BuildLabeledRuns(
 EROIBW8* sourceImage
 )

void BuildLabeledRuns(
 EROIBW16* sourceImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

### Remarks

Uses EBW8 (EBW16) information for class indices, i.e. 255 (65,535) possible classes. Class 0 is not coded. Building runs is the process of grouping pixels from an image to form horizontal segments. The pixels are assigned class indices based either on thresholding (ECodedImage::BuildRuns) or on the pixel values themselves (**BuildLabeledRuns**). A run is a set of horizontally connected pixels of the same class.

# ECodedImage::BuildObjects

Groups runs to form separated objects (connected blobs), from runs detected by ECodedImage::BuildRuns or from a source image/ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BuildObjects(
 EROIBW1* sourceImage
 )

void BuildObjects(
 EROIBW8* sourceImage
 )

void BuildObjects(
 EROIC24* sourceImage
 )
```

```
void BuildObjects(
 )
```

### Parameters

*sourceImage*

Pointer to a source ROI.

### Remarks

Without argument, the method groups the runs detected by ECodedImage::BuildRuns to form separate objects, i.e. connected components. With a source ROI as argument, the method segments it into connected blobs comprised of pixels of the same class. The EROIBW8 parameter is converted to white/neutral/black classes, using two thresholds. The EROIC24 parameter is converted to white/black classes, using two color thresholds. Then, the method extracts runs from them, and groups the runs to form separate objects, i.e. connected components. Building objects is the process of grouping pixels from an image to form connected blobs. The pixels are assigned class indices based either on thresholding (**BuildObjects**) or on the pixel values themselves (ECodedImage::BuildLabeledObjects). A blob is a set of connected pixels of the same class.

# ECodedImage::BuildRuns

Converts the specified ROI to classes, and extracts the runs from it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void BuildRuns(
 EROIBW1* sourceImage
 )
void BuildRuns(
 EROIBW8* sourceImage
 )
void BuildRuns(
 EROIC24* sourceImage
 )
```

### Parameters

*sourceImage*

Pointer to the source ROI.

### Remarks

The EROIBW8 parameter is converted to white/neutral/black classes, using two thresholds. The EROIC24 parameter is converted to white/black classes, using two color thresholds. Then, the method extracts runs from them. Building runs is the process of grouping pixels from an image to form horizontal segments. The pixels are assigned class indices based either on thresholding (**BuildRuns**) or on the pixel values themselves (ECodedImage::BuildLabeledRuns). A run is a set of horizontally connected pixels of the same class.

# ECodedImage::GetConnexity

# ECodedImage::SetConnexity

Connexity mode, that is how neighboring pixels are considered to belong to the same objects.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EConnexity GetConnexity()
void SetConnexity(Euresys::Open_eVision_2_10::EConnexity eConnexity)
```

# ECodedImage::GetContinuous

# ECodedImage::SetContinuous

Flag indicating whether the objects have to be built in the continuous mode or in the normal mode.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetContinuous()
void SetContinuous(BOOL bContinuous)
```

### Remarks

**TRUE** if objects are built in the continuous mode, **FALSE** if objects are built in the normal mode.

## ECodedImage::GetCurrentObjPtr

Pointer to the current objects list item, or **NULL**.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EListItem* GetCurrentObjPtr()
```

## ECodedImage::GetCurrentRunPtr

Pointer to the current run list item, or **NULL**.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EListItem* GetCurrentRunPtr()
```

# ECodedImage::GetDrawDiagonals

# ECodedImage::SetDrawDiagonals

Flag indicating whether the limit rectangle diagonals must be drawn or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetDrawDiagonals()
void SetDrawDiagonals(BOOL bDrawDiagonals)
```

## Remarks

If **TRUE** (default), diagonals are drawn.

# ECodedImage::DrawObject

Draws the designated object in solid color.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawObject(
 HDC graphicContext,
 OEV_INT32 objectNumber,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void DrawObject(
 HDC graphicContext,
 EListItem* object_,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawObject(
 HDC graphicContext,
 const ERGBColor& color,
 OEV_INT32 objectNumber,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawObject(
 HDC graphicContext,
 const ERGBColor& color,
 EListItem* object_,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawObject(
 EDrawAdapter* graphicContext,
 OEV_INT32 objectNumber,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawObject(
 EDrawAdapter* graphicContext,
 EListItem* object_,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*objectNumber*

Number of the object to be drawn.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0**, the default value, the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*object_*

Pointer to the list item (from the objects list) corresponding to the object to be drawn.

*color*

The color in which to draw the overlay.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used. The objects can be specified either by number of by list pointer.

# ECodedImage::DrawObjectFeature

Draws a graphical representation of a feature of the designated object in solid color.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawObjectFeature(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 OEV_INT32 objectNumber,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawObjectFeature(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 EListItem* object_,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawObjectFeature(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 EListItem* object_,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawObjectFeature(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 OEV_INT32 objectNumber,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void DrawObjectFeature(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 OEV_INT32 objectNumber,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawObjectFeature(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 EListItem* object_,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*feature*

Feature to be drawn, as defined by ELegacyFeature (use any value matching the aforementioned features, f.i. ELegacyFeature_EllipseWidth will draw the ellipse of inertia).

*objectNumber*

Number of the object to be drawn.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0**, the default value, the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*object_*

Pointer to the list item (from the objects list) corresponding to the object to be drawn.

*color*

The color in which to draw the overlay.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used. The objects can be specified either by number of by list pointer. If a required feature has not been computed for some object, nothing is drawn and no error message is issued!

# ECodedImage::DrawObjectFeatureWithCurrentPen

Draws a graphical representation of a feature of the designated object in solid color.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawObjectFeatureWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 EListItem* objectNumber,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
void DrawObjectFeatureWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 OEV_INT32 objectNumber,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*feature*

Feature to be drawn, as defined by ELegacyFeature (use any value matching the aforementioned features, f.i. ELegacyFeature_EllipseWidth will draw the ellipse of inertia).

*objectNumber*

Number of the object to be drawn.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0**, the default value, the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used. The objects can be specified either by number of by list pointer. If a required feature has not been computed for some object, nothing is drawn and no error message is issued!

# ECodedImage::DrawObjects

Draws all objects in solid color.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawObjects(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag selectionFlag,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void DrawObjects(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::ESelectionFlag selectionFlag,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawObjects(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag selectionFlag,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*selectionFlag*

Tells how the selected/unselected state of the objects is handled, as defined by ESelectionFlag.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*color*

The color in which to draw the overlay.

## Remarks

Drawing is done in the device context associated to the desired window. The current pen is used for all objects (to vary the colors, draw the objects separately using the ECodedImage::DrawObject method instead). Optionally, only the selected or deselected objects can be drawn.

# ECodedImage::DrawObjectsFeature

Draws a graphical representation of a feature.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawObjectsFeature(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 Euresys::Open_eVision_2_10::ESelectionFlag selectionFlag,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawObjectsFeature(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 Euresys::Open_eVision_2_10::ESelectionFlag selectionFlag,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawObjectsFeature(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 Euresys::Open_eVision_2_10::ESelectionFlag selectionFlag,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*feature*

Feature to be drawn, as defined by ELegacyFeature (use any value matching the aforementioned features, f.i. ELegacyFeature_EllipseWidth will draw the ellipse of inertia).

*selectionFlag*

Tells how the selected/unselected state of the objects is handled, as defined by ESelectionFlag.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*color*

The color in which to draw the overlay.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used for all objects (to vary the colors, draw the objects separately using the ECodedImage::DrawObjectFeature method instead). Optionally, only the selected or deselected objects can be drawn. Only the following features can be drawn: * ELegacyFeature_GravityCenter: upright cross; * ELegacyFeature_Centroid: skewed cross; * ELegacyFeature_Limit: upright bounding rectangle with diagonals; * ELegacyFeature_Limit45: skewed bounding rectangle with diagonals; * ELegacyFeature_EllipseWidth: ellipse of inertia (edge and main axis). If a required feature has not been computed for some object, nothing is drawn and no error message is issued!

# ECodedImage::DrawObjectsFeatureWithCurrentPen

Draws a graphical representation of a feature.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawObjectsFeatureWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 Euresys::Open_eVision_2_10::ESelectionFlag selectionFlag,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*feature*

Feature to be drawn, as defined by ELegacyFeature (use any value matching the aforementioned features, f.i. ELegacyFeature_EllipseWidth will draw the ellipse of inertia).

*selectionFlag*

Tells how the selected/unselected state of the objects is handled, as defined by ESelectionFlag.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

## Remarks

Drawing is done in the device context associated to the desired window. The current pen is used for all objects (to vary the colors, draw the objects separately using the ECodedImage::DrawObjectFeature method instead). Optionally, only the selected or deselected objects can be drawn. Only the following features can be drawn: * ELegacyFeature_GravityCenter: upright cross; * ELegacyFeature_Centroid: skewed cross; * ELegacyFeature_Limit: upright bounding rectangle with diagonals; * ELegacyFeature_Limit45: skewed bounding rectangle with diagonals; * ELegacyFeature_EllipseWidth: ellipse of inertia (edge and main axis). If a required feature has not been computed for some object, nothing is drawn and no error message is issued!

# ECodedImage::DrawObjectsWithCurrentPen

Draws all objects in solid color.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawObjectsWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag selectionFlag,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*selectionFlag*

Tells how the selected/unselected state of the objects is handled, as defined by ESelectionFlag.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

## Remarks

Drawing is done in the device context associated to the desired window. The current pen is used for all objects (to vary the colors, draw the objects separately using the ECodedImage::DrawObject method instead). Optionally, only the selected or deselected objects can be drawn.

# ECodedImage::DrawObjectWithCurrentPen

Draws the designated object in solid color.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawObjectWithCurrentPen(
 HDC graphicContext,
 OEV_INT32 objectNumber,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
void DrawObjectWithCurrentPen(
 HDC graphicContext,
 EListItem* object_,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*objectNumber*

Number of the object to be drawn.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0**, the default value, the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*object_*

Pointer to the list item (from the objects list) corresponding to the object to be drawn.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used. The objects can be specified either by number of by list pointer.

# ECodedImage::ECodedImage

Constructs a void coded image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ECodedImage(
 const ECodedImage& other
 )
void ECodedImage(
 )
```

### Parameters

*other*

-

# ECodedImage::FeatureAverage

Computes the average of the features of all currently selected objects.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void FeatureAverage(
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 float& average
 )
```

### Parameters

*feature*

Feature code, as defined by ELegacyFeature.

*average*

Reference to the feature average.

### Remarks

This measures the central tendency of a population of objects.

# ECodedImage::FeatureDeviation

Computes the average and standard deviation of the features of all currently selected objects.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void FeatureDeviation(
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 float& average,
 float& deviation
 )
```

### Parameters

*feature*

Feature code, as defined by ELegacyFeature.

*average*

Reference to the feature average.

*deviation*

Reference to the feature standard deviation.

# ECodedImage::FeatureMaximum

Computes the maximum of the features of all currently selected objects.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void FeatureMaximum(
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 float& maximum
 )
```

### Parameters

*feature*

Feature code, as defined by ELegacyFeature.

*maximum*

Reference to the feature maximum.

# ECodedImage::FeatureMinimum

Computes the minimum of the features of all currently selected objects.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void FeatureMinimum(
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 float& minimum
 )
```

### Parameters

*feature*

Feature code, as defined by ELegacyFeature.

*minimum*

Reference to the feature minimum.

# ECodedImage::FeatureVariance

Computes the average and variance of the features of all currently selected objects.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void FeatureVariance(
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 float& average,
 float& variance
 )
```

### Parameters

*feature*

Feature code, as defined by ELegacyFeature.

*average*

Reference to the feature average.

*variance*

Reference to the feature variance.

### Remarks

This measures the central tendency and the dispersion of a population of objects.

# ECodedImage::GetFirstObjPtr

Pointer to the first objects list item, or **NULL**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ElistItem* GetFirstObjPtr()
```

# ECodedImage::GetCurrentObjData

Returns the data of the current object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetCurrentObjData(
 EObjectData* objectData
 )
```

## Parameters

*objectData*

Pointer to an EObjectData structure to receive the data.

## Remarks

# ECodedImage::GetCurrentRunData

Returns the data of the current run.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetCurrentRunData(
 ERunData* run
 )
```

### Parameters

*run*

Pointer to an ERunData to receive the data.

# ECodedImage::GetFeatData

Gets the EFeatureData associated to a given feature list item.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetFeatData(
 EListItem* currentFeature,
 EFeatureData* featureData
 )
```

### Parameters

*currentFeature*

Pointer to the feature list item.

*featureData*

Pointer to a EFeatureData to receive the data.

# ECodedImage::GetFeatDataSize

Returns the data size of the specified feature.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EDataSize GetFeatDataSize(
 OEV_INT32 position
 )
Euresys::Open_eVision_2_10::EDataSize GetFeatDataSize(
 EListItem* currentFeature
 )
```

### Parameters

*position*

Absolute position in the features list, counting from **0** on.

*currentFeature*

Pointer to the feature list item.

### Remarks

The features data sizes are defined in EDataSize.

# ECodedImage::GetFeatDataType

Returns the data type of the specified feature.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EDataType GetFeatDataType(
 OEV_INT32 position
 )
Euresys::Open_eVision_2_10::EDataType GetFeatDataType(
 EListItem* currentFeature
 )
```

### Parameters

*position*

Absolute position in the features list, counting from **0** on.

*currentFeature*

Pointer to the feature list item.

### Remarks

The features data types are defined in EDataType.

# ECodedImage::GetFeatNum

Returns the code number of the specified feature.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetFeatNum(
 OEV_INT32 position
 )
OEV_INT32 GetFeatNum(
 EListItem* currentFeature
 )
```

### Parameters

*position*

Absolute position in the features list, counting from **0** on.

*currentFeature*

Pointer to the feature list item.

### Remarks

The features code numbers are defined in ELegacyFeature.

# ECodedImage::GetFeatPtrByNum

Returns a pointer to the feature list item for a given feature number, or **NULL**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EListItem* GetFeatPtrByNum(
 OEV_INT32 numFeat
 )
```

## Parameters

*numFeat*

Feature number, as defined by ELegacyFeature.

# ECodedImage::GetFeatSize

Returns the size (number of elements) of the feature array for the specified feature.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetFeatSize(
 OEV_INT32 position
 )

OEV_INT32 GetFeatSize(
 EListItem* currentFeature
 )
```

## Parameters

*position*

Absolute position in the features list, counting from **0** on.

*currentFeature*

Pointer to the feature list item.

# ECodedImage::GetFirstHole

Returns a pointer to the first hole related to the specified object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EListItem* GetFirstHole(
 EListItem* parentObject
 )
```

## Parameters

*parentObject*

Pointer to the objects list item, for which the first hole has to be pointed out.

## Remarks

If **parentObject** refers to a hole (instead of a real object) or to an object comprising no hole, the ECodedImage::GetFirstHole function returns **NULL**.

# ECodedImage::GetFirstObjData

Moves the cursor to the first object and returns the associated data.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void GetFirstObjData(
 EObjectData* object_
 )
```

### Parameters

*object_*

Pointer to an EObjectData to receive the data.

### Remarks

# ECodedImage::GetFirstRunData

Moves the cursor to the first run and returns the associated data.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetFirstRunData(
 ERunData* run
 )
```

### Parameters

*run*

Pointer to an ERunData to receive the data.

# ECodedImage::GetFirstRunPtr

Pointer to the first run list item, or **NULL**.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EListItem* GetFirstRunPtr(
 )
```

# ECodedImage::GetHoleParentObject

Returns a pointer to the real object including the specified hole.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EListItem* GetHoleParentObject(
 EListItem* hole
 )
```

## Parameters

*hole*

Pointer to the hole list item, for which the parent object has to be pointed out.

# ECodedImage::GetLastObjData

Moves the cursor to the last object and returns the associated data.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void GetLastObjData(
 EObjectData* object_
 )
```

### Parameters

*object_*

Pointer to an EObjectData structure to receive the data.

# ECodedImage::GetLastRunData

Moves the cursor to the last run and returns the associated data.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetLastRunData(
 ERunData* run
 )
```

### Parameters

*run*

Pointer to an ERunData to receive the data.

# ECodedImage::GetLastRunPtr

Pointer to the last run list item, or **NULL**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ElistItem* GetLastRunPtr(
 )
```

# ECodedImage::GetNextHole

Returns a pointer to the hole following the specified hole, in the objects list.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ElistItem* GetNextHole(
 ElistItem* hole
 )
```

## Parameters

*hole*

Pointer to the hole list item, for which the following hole has to be pointed out.

## Remarks

If there is no hole yet in the list, the ECodedImage::GetNextHole function returns **NULL**.

# ECodedImage::GetNextObjData

Moves the cursor to the next object and returns the associated data.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetNextObjData(
 EObjectData* object_
 )
```

### Parameters

*object_*

Pointer to an EObjectData to receive the data.

# ECodedImage::GetNextObjPtr

Returns a pointer to the next objects list item, or **NULL**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EListItem* GetNextObjPtr(
 EListItem* listItem
 )
```

### Parameters

*listItem*

Pointer to the current objects list item.

# ECodedImage::GetNextRunData

Moves the cursor to the next run and returns the associated data.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void GetNextRunData(
 ERunData* run
 )
void GetNextRunData(
 ERunData* run,
 EListItem* listItem
 )
```

### Parameters

*run*

Pointer to an ERunData to receive the data.

*listItem*

Pointer to the current run list item.

# ECodedImage::GetNextRunPtr

Returns a pointer to the next run list item, or **NULL**.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EListItem* GetNextRunPtr(
 EListItem* listItem
 )
```

### Parameters

*listItem*

Pointer to the current run list item.

# ECodedImage::GetNumHoles

Returns the number of holes related to the specified object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetNumHoles(
 EListItem* object_
 )
```

### Parameters

*object_*

Pointer to the object list item whose holes have to be counted.

### Remarks

By default, the parameter **object** is set to **NULL**, meaning that the function returns the total number of holes added to the objects list. After a call to the ECodedImage::BuildHoles function, the ECodedImage::NumObjects and ECodedImage::NumSelectedObjects properties contain the total number of objects (i.e. real objects + holes).

# ECodedImage::GetNumObjectRuns

Returns the number of runs comprised in a given object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetNumObjectRuns(
 OEV_INT32 objectNumber
 )
```

```
OEV_INT32 GetNumObjectRuns(
 EListItem* listItem
 )
```

### Parameters

*objectNumber*

Object identification number.

*listItem*

Pointer to an objects list item.

# ECodedImage::GetObjDataPtr

Gets the EObjectData associated to a given objects list item.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetObjDataPtr(
 EListItem* currentFeature,
 EObjectData* objectData
 )
```

### Parameters

*currentFeature*

Pointer to the current objects list item.

*objectData*

Pointer to a EObjectData to receive the data.

# ECodedImage::GetObjectData

If an object identification number is given, moves the cursor to the object with a given identification number and returns the associated data. If a list item is given, returns the object data associated with an object list item (cf. EListItem). See also ECodedImage::GetCurrentObjData.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetObjectData(
 EObjectData* object_,
 OEV_INT32 objectNumber
 )
void GetObjectData(
 EObjectData* object_,
 EListItem* listItem
 )
```

### Parameters

*object_*

Pointer to an EObjectData structure to receive the object data.

*objectNumber*

Object identification number.

*listItem*

Pointer to the current object list item (cf. EListItem).

# ECodedImage::GetObjectFeature

Allows retrieving the value of a feature of a given object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void GetObjectFeature(
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 EListItem* objectNumber,
 OEV_INT8& result
 )

void GetObjectFeature(
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 EListItem* object_,
 OEV_INT16& result
 )

void GetObjectFeature(
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 EListItem* object_,
 OEV_INT32& result
 )

void GetObjectFeature(
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 EListItem* object_,
 float& result
 )

void GetObjectFeature(
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 EListItem* object_,
 double& result
 )

void GetObjectFeature(
 OEV_INT32 feature,
 OEV_INT32 objectNumber,
 OEV_INT8& result
 )

void GetObjectFeature(
 OEV_INT32 feature,
 OEV_INT32 objectNumber,
 OEV_INT16& result
 )
```

```
void GetObjectFeature(
 OEV_INT32 feature,
 OEV_INT32 objectNumber,
 OEV_INT32& result
 )

void GetObjectFeature(
 OEV_INT32 feature,
 OEV_INT32 objectNumber,
 float& result
 )

void GetObjectFeature(
 OEV_INT32 feature,
 OEV_INT32 objectNumber,
 double& result
 )

void GetObjectFeature(
 EListItem* feature,
 OEV_INT32 objectNumber,
 OEV_INT8& result
 )

void GetObjectFeature(
 EListItem* feature,
 OEV_INT32 objectNumber,
 OEV_INT16& result
 )

void GetObjectFeature(
 EListItem* feature,
 OEV_INT32 objectNumber,
 OEV_INT32& result
 )

void GetObjectFeature(
 EListItem* feature,
 OEV_INT32 objectNumber,
 float& result
 )

void GetObjectFeature(
 EListItem* feature,
 OEV_INT32 objectNumber,
 double& result
 )
```

### Parameters

*feature*

Pointer to the feature list item.

*objectNumber*

Object number.

*result*

Reference to the feature value.

*object_*

Pointer to the list item (from the objects list) corresponding to the object.

# ECodedImage::GetObjFirstRunPtr

Returns a pointer to the first run list item of an object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EListItem* GetObjFirstRunPtr(
 OEV_INT32 objectNumber
 )

EListItem* GetObjFirstRunPtr(
 EListItem* listItem
 )
```

### Parameters

*objectNumber*

Object identification number.

*listItem*

Pointer to the objects list item.

# ECodedImage::GetObjLastRunPtr

Returns a pointer to the last run list item of an object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ElistItem* GetObjLastRunPtr(
 OEV_INT32 objectNumber
 )
ElistItem* GetObjLastRunPtr(
 ElistItem* objectNumber
 )
```

### Parameters

*objectNumber*

Object identification number.

# ECodedImage::GetObjPtr

Returns a pointer to the given objects list item.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ElistItem* GetObjPtr(
 OEV_INT32 objectNumber
 )
```

### Parameters

*objectNumber*

Object identification number.

# ECodedImage::GetObjPtrByCoordinates

Returns a pointer to the objects list item that contains the point of given coordinates, or **NULL**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EListItem* GetObjPtrByCoordinates(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Point abscissa.

*y*

Point ordinate.

### Remarks

This function is useful for object selection with a mouse.

# ECodedImage::GetObjPtrByPos

Returns a pointer to the objects list item of given absolute position, or **NULL**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EListItem* GetObjPtrByPos(
 OEV_INT32 position
 )
```

### Parameters

*position*

Absolute position in the objects list, counting from **0** on.

# ECodedImage::GetPreviousObjData

Moves the cursor to the previous object and returns the associated data.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetPreviousObjData(
 EObjectData* object_
 )
```

### Parameters

*object_*

Pointer to an EObjectData to receive the data.

# ECodedImage::GetPreviousObjPtr

Returns a pointer to the previous objects list item, or **NULL**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EListItem* GetPreviousObjPtr(
 EListItem* listItem
 )
```

### Parameters

*listItem*

Pointer to the current objects list item.

# ECodedImage::GetPreviousRunData

Moves the cursor to the previous run and returns the associated data.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetPreviousRunData(
 ERunData* run,
 EListItem* listItem
 )
void GetPreviousRunData(
 ERunData* run
 )
```

### Parameters

*run*

Pointer to an ERunData to receive the data.

*listItem*

Pointer to the current run list item.

# ECodedImage::GetPreviousRunPtr

Returns a pointer to the previous run list item, or **NULL**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EListItem* GetPreviousRunPtr(
 EListItem* listItem
 )
```

### Parameters

*listItem*

Pointer to the current run list item.

# ECodedImage::GetRunData

Returns the run data associated to the specified run.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetRunData(
 ERunData* run,
 OEV_INT32 position
 )
void GetRunData(
 ERunData* run,
 EListItem* listItem
 )
```

### Parameters

*run*

Pointer to an ERunData to receive the data.

*position*

Absolute position in the run list, counting from **0** on.

*listItem*

Pointer to the current run list item.

# ECodedImage::GetRunDataPtr

Gets the ERunData associated to a given run list item.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetRunDataPtr(
 EListItem* currentFeature,
 ERunData* runData
 )
```

## Parameters

*currentFeature*

Pointer to the current run list item.

*runData*

Pointer to a ERunData to receive the data.

# ECodedImage::GetRunPtr

Returns a pointer to the run list item of given absolute position, or **NULL**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EListItem* GetRunPtr(
 OEV_INT32 position
 )
```

## Parameters

*position*

Absolute position in the run list, counting from 0 on.

# ECodedImage::GetRunPtrByCoordinates

Returns a pointer to the run list item that contains the point of given coordinates, or **NULL**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EListItem* GetRunPtrByCoordinates(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

## Parameters

*x*

Point abscissa.

*y*

Point ordinate.

## Remarks

This function is useful for run selection with a mouse.

# ECodedImage::GetHighColorThreshold

# ECodedImage::SetHighColorThreshold

Upper threshold (color) used for image segmentation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EC24 GetHighColorThreshold()
void SetHighColorThreshold(EC24 c24HighThreshold)
```

### Remarks

The threshold value is constant over the whole image.

# ECodedImage::GetHighImage

# ECodedImage::SetHighImage

Image used as an adaptive upper threshold.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIBW8* GetHighImage()
void SetHighImage(EROIBW8* pImage)
```

### Remarks

The threshold is adaptive (specified pixel by pixel).

# ECodedImage::GetHighThreshold

# ECodedImage::SetHighThreshold

Upper threshold (gray level) used for image segmentation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetHighThreshold()
void SetHighThreshold(OEV_UINT32 un32HighThreshold)
```

### Remarks

The threshold value is constant over the whole image.

# ECodedImage::IsHole

Returns **TRUE** if the specified object is a hole, **FALSE** otherwise.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL IsHole(
 EListItem* object_
 )
```

### Parameters

*object_*

Pointer to the objects list item.

# ECodedImage::IsObjectSelected

Sets **bSelected** to **TRUE** if an object is selected.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void IsObjectSelected(
 OEV_INT32 objectNumber,
 BOOL& selected
 )

void IsObjectSelected(
 EListItem* listItem,
 BOOL& selected
 )
```

### Parameters

*objectNumber*

Object identification number.

*selected*

Reference to the result.

*listItem*

Pointer to the objects list item.

### Remarks

# ECodedImage::GetLastObjPtr

Pointer to the last objects list item, or **NULL**.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EListItem* GetLastObjPtr()
```

# ECodedImage::GetLimitAngle

# ECodedImage::SetLimitAngle

Angle of the skewed bounding box feature, in the current angle unit.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetLimitAngle()
void SetLimitAngle(float f32Angle)
```

# ECodedImage::GetLowColorThreshold

# ECodedImage::SetLowColorThreshold

Lower threshold (color) used for image segmentation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EC24 GetLowColorThreshold()
void SetLowColorThreshold(EC24 c24LowThreshold)
```

## Remarks

The threshold value is constant over the whole image.

# ECodedImage::GetLowImage

# ECodedImage::SetLowImage

Image used as an adaptive lower threshold.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIBW8* GetLowImage()
void SetLowImage(EROIBW8* pImage)
```

## Remarks

The threshold value is adaptive (specified pixel by pixel).

# ECodedImage::GetLowThreshold

# ECodedImage::SetLowThreshold

Lower threshold (gray level) used for image segmentation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetLowThreshold()
void SetLowThreshold(OEV_UINT32 un32LowThreshold)
```

### Remarks

The threshold value is constant over the whole image.

# ECodedImage::GetMaxObjects

# ECodedImage::SetMaxObjects

Maximum number of objects to look for.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetMaxObjects()
void SetMaxObjects(OEV_UINT32 un32MaxObjects)
```

### Remarks

After having found that amount of object, the process will stop and return an error message. If not set, no maximum value is defined.

# ECodedImage::GetNeutralClass

# ECodedImage::SetNeutralClass

Neutral class index (between both thresholds).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
OEV_INT16 GetNeutralClass()
void SetNeutralClass(OEV_INT16 n16NeutralClass)
```

### Remarks

Non zero when the neutral runs (between both thresholds) are coded. **0** means "do not code this class". <!-- **2** by default. -->

# ECodedImage::GetNumFeatures

Number of features currently in use.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetNumFeatures()
```

# ECodedImage::GetNumHoleRuns

Total number of hole runs in the list of object runs.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetNumHoleRuns()
```

### Remarks

After a call to ECodedImage::BuildHoles, ECodedImage::NumRuns contains the total number of runs (real object runs + hole runs).

# ECodedImage::GetNumObjects

Number of objects in the coded image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetNumObjects()
```

## Remarks

After a call to ECodedImage::BuildHoles, ECodedImage::NumObjects returns the total number of objects (real objects + holes).

# ECodedImage::GetNumRuns

Total number of runs in the list of object runs.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetNumRuns()
```

## Remarks

After a call to ECodedImage::BuildHoles, ECodedImage::NumRuns returns the total number of runs (real object runs + hole runs).

# ECodedImage::GetNumSelectedObjects

# ECodedImage::SetNumSelectedObjects

Number of objects currently selected.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetNumSelectedObjects()
void SetNumSelectedObjects(OEV_INT32 n32Nb_Selected_Objects)
```

## Remarks

After a call to ECodedImage::BuildHoles, ECodedImage::NumSelectedObjects returns the total number of objects (real objects + holes).

# ECodedImage::ObjectConvexHull

Computes the convex hull of an object and stores it in a **EPathVector**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ObjectConvexHull(
 EListItem* object_,
 EPathVector* destinationVector
 )
```

## Parameters

*object_*

Pointer to the objects list item.

*destinationVector*

Vector containing the vertices coordinates of the convex hull.

# ECodedImage::RemoveAllFeats

Deletes all features from the features list.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveAllFeats(
 )
```

# ECodedImage::RemoveAllObjects

Deletes all objects from the objects list.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveAllObjects(
 )
```

# ECodedImage::RemoveAllRuns

Deletes all runs from the runs list.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveAllRuns(
 )
```

# ECodedImage::RemoveHoles

Permanently erases, from the objects list, holes related to the specified object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveHoles(
 EListItem* object_
 )
```

## Parameters

*object_*

Pointer to the objects list item, for which the holes have to be erased.

## Remarks

If the parameter is **NULL**, all the holes are deleted. By default, the parameter is set to **NULL**, meaning that all the holes have to be erased from the objects list.

# ECodedImage::RemoveObject

Deletes an object from the objects list.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void RemoveObject(
 OEV_INT32 objectNumber
 )

void RemoveObject(
 EListItem* listItem
 )
```

### Parameters

*objectNumber*

Object identification number.

*listItem*

Pointer to the current objects list item.

# ECodedImage::RemoveRun

Deletes a run from the runs list.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void RemoveRun(
 OEV_INT32 position
 )

void RemoveRun(
 EListItem* listItem
 )
```

### Parameters

*position*

Absolute position in the run list, counting from 0 on.

*listItem*

Pointer to the current run list item.

# ECodedImage::ResetContinuousMode

When the continuous mode is activated, this method resets the sequence of images.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ResetContinuousMode(
 )
```

### Remarks

Thus, the next call for an object building will not take into account any previous image. If the continuous mode is disabled, this method does nothing.

# ECodedImage::SelectAllObjects

Selects all objects.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SelectAllObjects(
 )
```

# ECodedImage::SelectHoles

Selects the holes related to the specified object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SelectHoles(
 EListItem* parentObject
 )
```

### Parameters

*parentObject*

Pointer to the objects list item, for which the holes have to be selected.

### Remarks

If the parameter is **NULL**, all the holes are selected. By default, the parameter is set to **NULL**, meaning that all the holes have to be selected. If **parentObject** is a hole (instead of a real object) or has no hole, no selection is performed.

# ECodedImage::SelectObject

Selects an object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SelectObject(
 OEV_INT32 objectNumber
 )
void SelectObject(
 EListItem* listItem
 )
```

### Parameters

*objectNumber*

Object identification number.

*listItem*

Pointer to the objects list item.

# ECodedImage::SelectObjectsUsingFeature

Selects or deselects objects, according to the value of a specified feature.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SelectObjectsUsingFeature(
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 OEV_INT8 minimum,
 OEV_INT8 maximum,
 Euresys::Open_eVision_2_10::ESelectOption operation
 )

void SelectObjectsUsingFeature(
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 OEV_INT16 minimum,
 OEV_INT16 maximum,
 Euresys::Open_eVision_2_10::ESelectOption operation
 )

void SelectObjectsUsingFeature(
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 OEV_INT32 minimum,
 OEV_INT32 maximum,
 Euresys::Open_eVision_2_10::ESelectOption operation
 )

void SelectObjectsUsingFeature(
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 float minimum,
 float maximum,
 Euresys::Open_eVision_2_10::ESelectOption operation
 )

void SelectObjectsUsingFeature(
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 double minimum,
 double maximum,
 Euresys::Open_eVision_2_10::ESelectOption operation
 )
```

### Parameters

*feature*

Feature code, as defined by EFeature.

*minimum*

Selection interval lower bound.

*maximum*

Selection interval upper bound.

*operation*

Selection mode, as defined by ESelectOption.

# ECodedImage::SelectObjectsUsingPosition

Selects or deselects objects, using a delimiting rectangle.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SelectObjectsUsingPosition(
 EBaseROI* roi,
 Euresys::Open_eVision_2_10::ESelectByPosition operation
 )

void SelectObjectsUsingPosition(
 OEV_INT32 originX,
 OEV_INT32 originY,
 OEV_INT32 width,
 OEV_INT32 height,
 Euresys::Open_eVision_2_10::ESelectByPosition operation
 )
```

### Parameters

*roi*

Pointer to an image/ROI whose position parameters will define the selection rectangle.

*operation*

Selection mode, as defined by ESelectByPosition.

*originX*

Abscissa of the upper left corner of the rectangle.

*originY*

Ordinate of the upper left corner of the rectangle.

*width*

Rectangle width, in pixels.

*height*

Rectangle height, in pixels.

### Remarks

The rectangle coordinates are always specified with respect to the whole image.

# ECodedImage::SetFeatInfo

Sets the appropriate data size and type for a predefined feature.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFeatInfo(
 EFeatureData* feature,
 Euresys::Open_eVision_2_10::ELegacyFeature featureCode
 )
```

### Parameters

*feature*

Feature code, as defined by ELegacyFeature.

*featureCode*

Pointer to a EFeatureData structure describing the feature.

# ECodedImage::SetFirstRunPtr

Sets the first run list item of an object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFirstRunPtr(
 EListItem* firstRun,
 OEV_INT32 objectNumber
 )
void SetFirstRunPtr(
 EListItem* firstRun,
 EListItem* currentObject
 )
```

### Parameters

*firstRun*

Pointer to the first run of the object.

*objectNumber*

Object identification number.

*currentObject*

Pointer to the objects list item.

# ECodedImage::SetLastRunPtr

Sets the last run list item of an object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetLastRunPtr(
 EListItem* lastRun,
 OEV_INT32 objectNumber
 )
void SetLastRunPtr(
 EListItem* lastRun,
 EListItem* currentObject
 )
```

### Parameters

*lastRun*

Pointer to the last run of the object.

*objectNumber*

Object identification number.

*currentObject*

Pointer to the objects list item.

# ECodedImage::SortObjectsUsingFeature

Sorts objects according to the value of some feature.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SortObjectsUsingFeature(
 Euresys::Open_eVision_2_10::ELegacyFeature feature,
 Euresys::Open_eVision_2_10::ESortOption Operation
 )
```

### Parameters

*feature*

Feature code, as defined by ELegacyFeature.

*Operation*

Selection mode, as defined by ESortOption.

# ECodedImage::GetThreshold

# ECodedImage::SetThreshold

Threshold mode (gray level) used for image segmentation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetThreshold()
void SetThreshold(OEV_UINT32 un32Threshold)
```

### Remarks

By default, the "minimum residue" mode is used to determine the threshold. The threshold is constant over the whole image. When using a single threshold instead of a low threshold and a high threshold, the neutral class is ignored. Only the black and white classes are relevant.

# ECodedImage::SetThresholdImage

Single threshold used for image segmentation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetThresholdImage(EROIBW8* pImage)
```

### Remarks

The threshold value is adaptive (specified pixel by pixel). When using a single threshold instead of a low threshold and a high threshold, the neutral class is ignored. Only the black and white classes are relevant.

# ECodedImage::GetTrueThreshold

Absolute threshold level, when using a single threshold.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetTrueThreshold()
```

# ECodedImage::UnselectAllObjects

Deselects all objects.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void UnselectAllObjects(
 )
```

# ECodedImage::UnselectHoles

Unselects the holes related to the specified object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void UnselectHoles(
 EListItem* parentObject
 )
```

## Parameters

*parentObject*

Pointer to the objects list item, for which the holes have to be unselected.

### Remarks

If the parameter is **NULL**, all the holes are unselected. By default, the parameter is set to **NULL**, meaning that all the holes have to be unselected. If **parentObject** is a hole (instead of a real object) or if **parentObject** has no hole, nothing is changed.

# ECodedImage::UnselectObject

Deselects an object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void UnselectObject(
 OEV_INT32 objectNumber
 )
void UnselectObject(
 EListItem* listItem
 )
```

### Parameters

*objectNumber*

Object identification number.

*listItem*

Pointer to the objects list item.

### Remarks

Once an object has been unselected, it doesn't allow browsing a list of selected objects anymore (using ECodedImage::GetPreviousObjPtr or ECodedImage::GetNextObjPtr).

# ECodedImage::GetWhiteClass

# ECodedImage::SetWhiteClass

White class index (above the upper threshold).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT16 GetWhiteClass()
void SetWhiteClass(OEV_INT16 n16WhiteClass)
```

### Remarks

Non zero when the white runs (above the upper threshold) are coded. **0** means "do not code this class". <!-- **3** by default. -->

# 4.48. ECodedImage2 Class

The main class of the EasyObject API that represents a coded image, as produced by the encoder.

### Remarks

It provides methods to get features of the encoded image, to access an object in a particular layer of the encoded image, to draw objects or objects features in a particular layer of the encoded image, to render a mask, etc...

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| ClearFeatureCache | Clears the internal cache for the computed features. |

| | |
|---|---|
| Draw | Draw the designated coded element, all the objects from a layer, or all the coded elements from a given selection. |
| DrawFeature | Draw a given feature of the designated coded element, of all the objects from a layer, or of all the coded elements from a given selection. |
| DrawFeatureWithCurrentPen | Draw a given feature of the designated coded element, of all the objects from a layer, or of all the coded elements from a given selection. |
| DrawHole | Draw the designated hole. |
| DrawHoleFeature | Draw a given feature of the designated hole. |
| DrawHoleFeatureWithCurrentPen | Draw a given feature of the designated hole. |
| DrawHoleWithCurrentPen | Draw the designated hole. |
| DrawObject | Draw the designated object. |
| DrawObjectFeature | Draw a given feature of the designated object. |
| DrawObjectFeatureWithCurrentPen | Draw a given feature of the designated object. |
| DrawObjectWithCurrentPen | Draw the designated object. |

| | |
|---|---|
| DrawWithCurrentPen | Draw the designated coded element, all the objects from a layer, or all the coded elements from a given selection. |
| ECodedImage2 | Constructs a coded image. |
| FindObject | Finds an object in the coded image using its coordinates. |
| GetHeight | Returns the height of the coded image. |
| GetLayerCount | Returns the number of layers that are encoded. |
| GetObj | Random access to an object in a given layer. |
| GetObjCount | Returns the number of objects in a given layer. |
| GetParentObject | Returns a reference to the object that contains a given hole. |
| GetStartY | Returns the lowest row index, for all the runs of all the objects in the coded image. |
| GetWidth | Returns the width of the coded image. |
| RenderMask | Creates a Flexible Mask from a specified layer of the encoded image. |

# ECodedImage2::ClearFeatureCache

Clears the internal cache for the computed features.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ClearFeatureCache(
 )
```

### Remarks

This is useful to reduce memory consumption.

# ECodedImage2::Draw

Draw the designated coded element, all the objects from a layer, or all the coded elements from a given selection.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 EDrawAdapter* graphicContext,
 ECodedElement& element,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 OEV_UINT32 layerIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 EObjectSelection& selection,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 EObjectSelection& selection,
 OEV_UINT32 elementIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 ECodedElement& element,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 ECodedElement& element,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 OEV_UINT32 layerIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 OEV_UINT32 layerIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void Draw(
 HDC graphicContext,
 EObjectSelection& selection,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 EObjectSelection& selection,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 EObjectSelection& selection,
 OEV_UINT32 elementIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 EObjectSelection& selection,
 OEV_UINT32 elementIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*graphicContext*

Graphic context on which to draw.

*element*

The coded element to draw.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*layerIndex*

Index of the layer of interest. If no layer index is specified, the default layer is taken into consideration. Note that this methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

*selection*

The selection of coded elements to draw.

*elementIndex*

The index in the selection of the coded element to draw. If no element index is specified, all the elements of the selection are take into consideration.

*color*

The color in which to draw the overlay.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used.

# ECodedImage2::DrawFeature

Draw a given feature of the designated coded element, of all the objects from a layer, or of all the coded elements from a given selection.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawFeature(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawFeature(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 layerIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawFeature(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 ECodedElement& element,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawFeature(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 EObjectSelection& selection,
 OEV_UINT32 elementIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
```

```
void DrawFeature(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 EObjectSelection& selection,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawFeature(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawFeature(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawFeature(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 layerIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
```

```
void DrawFeature(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 layerIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawFeature(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 ECodedElement& element,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawFeature(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 ECodedElement& element,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawFeature(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 EObjectSelection& selection,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
```

```
void DrawFeature(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 EObjectSelection& selection,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawFeature(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 EObjectSelection& selection,
 OEV_UINT32 elementIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawFeature(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 EObjectSelection& selection,
 OEV_UINT32 elementIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
```

## Parameters

*graphicContext*

Graphic context on which to draw.

*feature*

The feature of interest.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*drawDiagonals*

Specifies whether or not the diagonals of the ellipses and of the rectangles are to be drawn.

*layerIndex*

Index of the layer of interest. If no layer index is specified, the default layer is taken into consideration. Note that this methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

*element*

The coded element to draw.

*selection*

The selection of coded elements to draw.

*elementIndex*

The index in the selection of the coded element to draw. If no element index is specified, all the elements of the selection are take into consideration.

*color*

The color in which to draw the overlay.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used.

The features EDrawableFeature_FeretBox and EDrawableFeature_WeightedGravityCenter are only at one's disposal when drawing selections.

# ECodedImage2::DrawFeatureWithCurrentPen

Draw a given feature of the designated coded element, of all the objects from a layer, or of all the coded elements from a given selection.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawFeatureWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 layerIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawFeatureWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 ECodedElement& element,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawFeatureWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 EObjectSelection& selection,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawFeatureWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 EObjectSelection& selection,
 OEV_UINT32 elementIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
```

```
void DrawFeatureWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
```

### Parameters

*graphicContext*

Graphic context on which to draw.

*feature*

The feature of interest.

*layerIndex*

Index of the layer of interest. If no layer index is specified, the default layer is taken into consideration. Note that this methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*drawDiagonals*

Specifies whether or not the diagonals of the ellipses and of the rectangles are to be drawn.

*element*

The coded element to draw.

*selection*

The selection of coded elements to draw.

*elementIndex*

The index in the selection of the coded element to draw. If no element index is specified, all the elements of the selection are take into consideration.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used.

The features EDrawableFeature_FeretBox and EDrawableFeature_WeightedGravityCenter are only at one's disposal when drawing selections.

# ECodedImage2::DrawHole

Draw the designated hole.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawHole(
 EDrawAdapter* graphicContext,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawHole(
 EDrawAdapter* graphicContext,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void DrawHole(
 HDC graphicContext,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawHole(
 HDC graphicContext,
 const ERGBColor& color,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawHole(
 HDC graphicContext,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawHole(
 HDC graphicContext,
 const ERGBColor& color,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*graphicContext*

Graphic context on which to draw.

*objectIndex*

Index of the parent object of the hole to draw.

*holeIndex*

Index of the hole to draw.

*zoomX*

Horizontal zooming factor. By default, no scaling is used.

*zoomY*

Vertical zooming factor. If set to **0**, the default value, the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, the default layer will be taken into consideration.

*color*

The color in which to draw the overlay.

### Remarks

Drawing is done in the device context associated to the desired window.

This methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

# ECodedImage2::DrawHoleFeature

Draw a given feature of the designated hole.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawHoleFeature(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
void DrawHoleFeature(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
void DrawHoleFeature(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
```

```
void DrawHoleFeature(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawHoleFeature(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawHoleFeature(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
```

## Parameters

*graphicContext*

Graphic context on which to draw.

*feature*

The feature of interest.

*objectIndex*

Index of the parent object of the hole to draw.

*holeIndex*

Index of the hole to draw.

*zoomX*

Horizontal zooming factor. By default, no scaling is used.

*zoomY*

Vertical zooming factor. If set to **0**, the default value, the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*drawDiagonals*

Specifies whether or not the diagonals of the ellipses and of the rectangles are to be drawn.

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, the default layer will be taken into consideration.

*color*

The color in which to draw the overlay.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used.

This methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

Trying to draw the features EDrawableFeature_FeretBox and EDrawableFeature_WeightedGravityCenter will result in an exception, as they only make sense for EObjectSelection.

# ECodedImage2::DrawHoleFeatureWithCurrentPen

Draw a given feature of the designated hole.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawHoleFeatureWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawHoleFeatureWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
```

## Parameters

*graphicContext*

Graphic context on which to draw.

*feature*

The feature of interest.

*objectIndex*

Index of the parent object of the hole to draw.

*holeIndex*

Index of the hole to draw.

*zoomX*

Horizontal zooming factor. By default, no scaling is used.

*zoomY*

Vertical zooming factor. If set to **0**, the default value, the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*drawDiagonals*

Specifies whether or not the diagonals of the ellipses and of the rectangles are to be drawn.

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, the default layer will be taken into consideration.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used.

This methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

Trying to draw the features EDrawableFeature_FeretBox and EDrawableFeature_WeightedGravityCenter will result in an exception, as they only make sense for EObjectSelection.

# ECodedImage2::DrawHoleWithCurrentPen

Draw the designated hole.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawHoleWithCurrentPen(
 HDC graphicContext,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void DrawHoleWithCurrentPen(
 HDC graphicContext,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*graphicContext*

Graphic context on which to draw.

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, the default layer will be taken into consideration.

*objectIndex*

Index of the parent object of the hole to draw.

*holeIndex*

Index of the hole to draw.

*zoomX*

Horizontal zooming factor. By default, no scaling is used.

*zoomY*

Vertical zooming factor. If set to **0**, the default value, the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

## Remarks

Drawing is done in the device context associated to the desired window. The current pen is used.

This methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

# ECodedImage2::DrawObject

Draw the designated object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawObject(
 EDrawAdapter* graphicContext,
 OEV_UINT32 objectIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawObject(
 EDrawAdapter* graphicContext,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawObject(
 HDC graphicContext,
 OEV_UINT32 objectIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void DrawObject(
 HDC graphicContext,
 const ERGBColor& color,
 OEV_UINT32 objectIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawObject(
 HDC graphicContext,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawObject(
 HDC graphicContext,
 const ERGBColor& color,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*graphicContext*

Graphic context on which to draw.

*objectIndex*

Index of the object to draw.

*zoomX*

Horizontal zooming factor. By default, no scaling is used.

*zoomY*

Vertical zooming factor. If set to **0**, the default value, the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, the default layer will be taken into consideration.

*color*

The color in which to draw the overlay.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used.

This methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

# ECodedImage2::DrawObjectFeature

Draw a given feature of the designated object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawObjectFeature(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 objectIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
```

```
void DrawObjectFeature(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawObjectFeature(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 objectIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void DrawObjectFeature(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 objectIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
```

```
void DrawObjectFeature(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
void DrawObjectFeature(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
```

## Parameters

*graphicContext*

Graphic context on which to draw.

*feature*

The feature of interest.

*objectIndex*

Index of the object to draw.

*zoomX*

Horizontal zooming factor. By default, no scaling is used.

*zoomY*

Vertical zooming factor. If set to **0**, the default value, the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*drawDiagonals*

Specifies whether or not the diagonals of the ellipses and of the rectangles are to be drawn.

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, the default layer will be taken into consideration.

*color*

The color in which to draw the overlay.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used.

This methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

Trying to draw the features EDrawableFeature_FeretBox and EDrawableFeature_WeightedGravityCenter will result in an exception, as they only make sense for EObjectSelection.

# ECodedImage2::DrawObjectFeatureWithCurrentPen

Draw a given feature of the designated object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawObjectFeatureWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 objectIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
```

```
void DrawObjectFeatureWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawableFeature feature,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
```

## Parameters

*graphicContext*

Graphic context on which to draw.

*feature*

The feature of interest.

*objectIndex*

Index of the object to draw.

*zoomX*

Horizontal zooming factor. By default, no scaling is used.

*zoomY*

Vertical zooming factor. If set to **0**, the default value, the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*drawDiagonals*

Specifies whether or not the diagonals of the ellipses and of the rectangles are to be drawn.

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, the default layer will be taken into consideration.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used.

This methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

Trying to draw the features EDrawableFeature_FeretBox and EDrawableFeature_WeightedGravityCenter will result in an exception, as they only make sense for EObjectSelection.

# ECodedImage2::DrawObjectWithCurrentPen

Draw the designated object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawObjectWithCurrentPen(
 HDC graphicContext,
 OEV_UINT32 objectIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
void DrawObjectWithCurrentPen(
 HDC graphicContext,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*graphicContext*

Graphic context on which to draw.

*objectIndex*

Index of the object to draw.

*zoomX*

Horizontal zooming factor. By default, no scaling is used.

*zoomY*

Vertical zooming factor. If set to **0**, the default value, the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, the default layer will be taken into consideration.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used.

This methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

# ECodedImage2::DrawWithCurrentPen

Draw the designated coded element, all the objects from a layer, or all the coded elements from a given selection.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawWithCurrentPen(
 HDC graphicContext,
 ECodedElement& element,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void DrawWithCurrentPen(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawWithCurrentPen(
 HDC graphicContext,
 OEV_UINT32 layerIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawWithCurrentPen(
 HDC graphicContext,
 EObjectSelection& selection,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawWithCurrentPen(
 HDC graphicContext,
 EObjectSelection& selection,
 OEV_UINT32 elementIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*graphicContext*

Graphic context on which to draw.

*element*

The coded element to draw.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*layerIndex*

Index of the layer of interest. If no layer index is specified, the default layer is taken into consideration. Note that this methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

*selection*

The selection of coded elements to draw.

*elementIndex*

The index in the selection of the coded element to draw. If no element index is specified, all the elements of the selection are take into consideration.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used.

# ECodedImage2::ECodedImage2

Constructs a coded image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ECodedImage2(
 const ECodedImage2& other
 )
void ECodedImage2(
 )
```

### Parameters

*other*

-

# ECodedImage2::FindObject

Finds an object in the coded image using its coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EObject* FindObject(
 int x,
 int y
 )

EObject* FindObject(
 OEV_UINT32 layerIndex,
 int x,
 int y
 )
```

### Parameters

*x*

The X-coordinate of the object.

*y*

The Y-coordinate of the object.

*layerIndex*

The index of the layer of interest.

### Remarks

If no layer index is specified, all the layers of the coded image are scanned until an object is found at these coordinates.

# ECodedImage2::GetObj

Random access to an object in a given layer.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EObject& GetObj(
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex
 )

const EObject& GetObj(
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex
 )

EObject& GetObj(
 OEV_UINT32 objectIndex
 )

const EObject& GetObj(
 OEV_UINT32 objectIndex
 )
```

### Parameters

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, the default layer will be taken into consideration.

*objectIndex*

The index of the object in the layer.

### Remarks

This methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

# ECodedImage2::GetObjCount

Returns the number of objects in a given layer.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetObjCount(
 OEV_UINT32 layerIndex
 )

OEV_UINT32 GetObjCount(
 )
```

### Parameters

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, the default layer will be taken into consideration.

### Remarks

This methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

# ECodedImage2::GetParentObject

Returns a reference to the object that contains a given hole.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EObject& GetParentObject(
 EHole& hole
 )
```

### Parameters

*hole*

The hole of interest.

# ECodedImage2::GetHeight

Returns the height of the coded image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetHeight() const
```

## Remarks

If the continuous mode is not activated, this height corresponds to the height of the source image. If the continuous mode is activated, this value equals to the highest row index, for all the runs of all the objects in the coded image, augmented by the number of rows index that are below zero.

# ECodedImage2::GetLayerCount

Returns the number of layers that are encoded.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetLayerCount() const
```

# ECodedImage2::RenderMask

Creates a Flexible Mask from a specified layer of the encoded image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RenderMask(
 EROIBW8& result
 )
void RenderMask(
 EROIBW8& result,
 OEV_UINT32 layerIndex,
 int offsetX,
 int offsetY
 )
void RenderMask(
 EROIBW8& result,
 OEV_UINT32 layerIndex
 )
void RenderMask(
 EROIBW8& result,
 int offsetX,
 int offsetY
 )
```

## Parameters

*result*

The image in which the generated mask will be stored.

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, the default layer will serve as a source for the mask generation.

*offsetX*

The X-offset that must be applied to bring the zero X-coordinate in the coded image on the first column of the result image (defaults to zero).

*offsetY*

The Y-offset that must be applied to bring the zero Y-coordinate in the coded image on the first row of the result image (defaults to zero).

## Remarks

The size of the result image will not be changed: It must be properly sized beforehand.

This methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

## ECodedImage2::GetStartY

Returns the lowest row index, for all the runs of all the objects in the coded image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetStartY() const
```

### Remarks

The returned value will always be zero if the continuous mode is not activated.

## ECodedImage2::GetWidth

Returns the width of the coded image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetWidth() const
```

### Remarks

This width corresponds in any case to the width of the source image.

# 4.49. EColorLookup Class

Describes a color lookup table, that is used to speed-up complex conversions between color systems.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AdjustGainOffset | Sets a color transformation such that a gain and offset is applied separately to each color component of a target color system. |
| Calibrate | Sets a color transformation to recalibrate. |
| ConvertFromRgb | Sets a color transformation from the EColorSystem_Rgb representation to another system, as defined by EColorSystem. |
| ConvertToRgb | Sets a color transformation from any color system, as defined by EColorSystem, to the EColorSystem_Rgb representation. |
| EColorLookup | Constructs a color lookup table. |
| GetColorSystemIn | Input color system. |
| GetColorSystemOut | Output color system. |
| GetIndexBits | Number of bits used for indexing the lookup table. |
| GetInterpolation | Interpolation mode. |
| SetIndexBits | Number of bits used for indexing the lookup table. |

| | |
|---|---|
| SetInterpolation | Interpolation mode. |
| Transform | Transforms a quantized color image/pixel to another quantized color image/pixel, using the previously initialized color lookup table. |
| WhiteBalance | Initializes a color lookup table that can be used for color adjustment, including a global gain, gamma pre-compensation [cancellation] and white balancing. |

# olorLookup::AdjustGainOffset

Sets a color transformation such that a gain and offset is applied separately to each color component of a target color system.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AdjustGainOffset(
 Euresys::Open_eVision_2_10::EColorSystem colorSystem,
 float gain0,
 float offset0,
 float gain1,
 float offset1,
 float gain2,
 float offset2
 )
```

## Parameters

*colorSystem*

Target color system, as defined by EColorSystem.

*gain0*

Gain to be applied to color component 0.

*offset0*

Offset to be applied to color component 0.

*gain1*

Gain to be applied to color component 1.

*offset1*

Offset to be applied to color component 1.

*gain2*

Gain to be applied to color component 2.

*offset2*

Offset to be applied to color component 2.

### Remarks

The input and output color systems are both EColorSystem_Rgb. To apply some transform to a color image, you initialize a color lookup once for all and use it at will in a transformation operation such as EColorLookup::Transform.

**Note.** The offsets are specified as unquantized values.

# EColorLookup::Calibrate

Sets a color transformation to recalibrate.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void Calibrate(
 EC24 Color0,
 float x0,
 float y0,
 float z0,
 EC24 Color1,
 float x1,
 float y1,
 float z1,
 EC24 Color2,
 float x2,
 float y2,
 float z2
 )

void Calibrate(
 EC24 Color0,
 float x0,
 float y0,
 float z0,
 EC24 Color1,
 float x1,
 float y1,
 float z1,
 EC24 Color2,
 float x2,
 float y2,
 float z2,
 EC24 Color3,
 float x3,
 float y3,
 float z3
 )

void Calibrate(
 EC24 color,
 float x,
 float y,
 float z
 )
```

## Parameters

*Color0*

Measured quantized values of a pixel of color 0.

*x0*

CIE XYZ tri-stimulus unquantized values corresponding to color 0.

*y0*

CIE XYZ tri-stimulus unquantized values corresponding to color 0.

*z0*

CIE XYZ tri-stimulus unquantized values corresponding to color 0.

*Color1*

Measured quantized values of a pixel of color 1.

*x1*

CIE XYZ tri-stimulus unquantized values corresponding to color 1.

*y1*

CIE XYZ tri-stimulus unquantized values corresponding to color 1.

*z1*

CIE XYZ tri-stimulus unquantized values corresponding to color 1.

*Color2*

Measured quantized values of a pixel of color 2.

*x2*

CIE XYZ tri-stimulus unquantized values corresponding to color 2.

*y2*

CIE XYZ tri-stimulus unquantized values corresponding to color 2.

*z2*

CIE XYZ tri-stimulus unquantized values corresponding to color 2.

*Color3*

Measured quantized values of a pixel of color 3.

*x3*

CIE XYZ tri-stimulus unquantized values corresponding to color 3.

*y3*

CIE XYZ tri-stimulus unquantized values corresponding to color 3.

*z3*

CIE XYZ tri-stimulus unquantized values corresponding to color 3.

*color*

-

*x*

-

*y*

-

*z*

-

### Remarks

The first prototype uses 3 reference colors. The second uses 4 reference colors. To apply some transform to a color image, you initialize a color lookup once for all and use it at will in a transformation operation such as EColorLookup::Transform.

# EColorLookup::GetColorSystemIn

Input color system.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EColorSystem GetColorSystemIn()
```

### Remarks

The **EColorLookup** objects keep track of the color system transformation, for consistency. When applying a transformation, the source image color system (usually EColorSystem_Rgb) must match the *input color system*; the destination image will be automatically be typed with the *output color system*. In case of a mismatch, an error message is issued. These two values are set by the lookup table initialization functions. An uninitialized lookup table has both color systems set to EColorSystem_NoColor.

# EColorLookup::GetColorSystemOut

Output color system.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EColorSystem GetColorSystemOut()
```

### Remarks

The **EColorLookup** objects keep track of the color system transformation, for consistency. When applying a transformation, the source image color system (usually EColorSystem_Rgb) must match the *input color system*; the destination image will be automatically be typed with the *output color system*. In case of a mismatch, an error message is issued. These two values are set by the lookup table initialization functions. An uninitialized lookup table has both color systems set to EColorSystem_NoColor.

# EColorLookup::ConvertFromRgb

Sets a color transformation from the Rgb representation to another system, as defined by EColorSystem.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ConvertFromRgb(
 Euresys::Open_eVision_2_10::EColorSystem colorSystem
 )
```

### Parameters

*colorSystem*

Color system, as defined by EColorSystem.

### Remarks

The input and output color systems are respectively EColorSystem_Rgb and **colorSystem**. To apply some transform to a color image, you initialize a color lookup once for all and use it at will in a transformation operation such as EColorLookup::Transform.

# EColorLookup::ConvertToRgb

Sets a color transformation from any color system, as defined by EColorSystem, to the Rgb representation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ConvertToRgb(
 Euresys::Open_eVision_2_10::EColorSystem colorSystem
 )
```

### Parameters

*colorSystem*

Color system, as defined by EColorSystem.

### Remarks

The input and output color systems are respectively **colorSystem** and EColorSystem_Rgb. To apply some transform to a color image, you initialize a color lookup once for all and use it at will in a transformation operation such as EColorLookup::Transform.

# EColorLookup::EColorLookup

Constructs a color lookup table.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EColorLookup(
 const EColorLookup& other
 )
void EColorLookup(
 )
```

### Parameters

*other*

-

# EColorLookup::GetIndexBits

# EColorLookup::SetIndexBits

Number of bits used for indexing the lookup table.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetIndexBits()
void SetIndexBits(OEV_UINT32 un32IndexBits)
```

### Remarks

Before filling in a lookup table, it is necessary to decide how many table entries it requires. The EColorLookup::IndexBits property indicates how many (high-order) bits of the input components are used. The relation between EColorLookup::IndexBits, the number of table entries and the corresponding table size are given below:  The larger the number of entries, the more accuracy is obtained. After EColorLookup::IndexBits has been changed, the lookup table needs to be recomputed.

**Note.** Be aware that each time a color lookup table is filled, all the entries are recomputed. When EColorLookup::IndexBits equals **6**, this may take a very long time. Such large lookup tables should be computed once only. Different combinations of EColorLookup::IndexBits and **Interpolation** provide a trade-off between accuracy and speed for the table pre-computation and table use.

# EColorLookup::GetInterpolation

# EColorLookup::SetInterpolation

Interpolation mode.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetInterpolation()
void SetInterpolation(BOOL bInterpolation)
```

### Remarks

When applying a lookup table to transform pixel values, tri-linear interpolation can be used: * when interpolation is not used, the table is looked up at the entry closest to the pixel value. This gives an accuracy equal to the value of the **IndexBits** property. On the other hand, table lookup is very fast; * when interpolation is used, the table is looked up at eight neighboring entries and an adequate average is computed. This gives full accuracy (8 bits) if the transformation is smooth enough. On the other hand, table lookup is slower.

**Note.** The interpolation mode may be modified at any time without the need to reinitialize the lookup table.

# EColorLookup::Transform

Transforms a quantized color image/pixel to another quantized color image/pixel, using the previously initialized color lookup table.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Transform(
 EC24 sourceImageColor,
 EC24& destinationImageColor
 )
void Transform(
 EROIC24* sourceImage,
 EROIC24* destinationImage
 )
```

### Parameters

*sourceImageColor*

Input color image.

*destinationImageColor*

Output color image.

*sourceImage*

Input color image.

*destinationImage*

Output color image.

# EColorLookup::WhiteBalance

Initializes a color lookup table that can be used for color adjustment, including a global gain, gamma pre-compensation [cancellation] and white balancing.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void WhiteBalance(
 float gain,
 float gamma,
 float balanceRed,
 float balanceGreen,
 float balanceBlue
 )
```

## Parameters

*gain*

Global gain to be applied to all three color components. By default, the image intensity remains unchanged.

*gamma*

Gamma exponent. Setting this parameter will cancel the gamma pre-compensation feature of the camera, or apply it. By default, the no gamma pre-compensation is assumed (linear response). The gamma exponent can be chosen among the predefined values EasyColor::CompensateNtscGamma/EasyColor::CompensatePalGamma/EasyColor::CompensateSmpteGamma (pre-compensation) or EasyColor::NtscGamma/EasyColor::PalGamma/EasyColor::SmpteGamma (pre-compensation cancellation), or be user-defined.

*balanceRed*

Color values to be used for white balance. These parameters should be set to the measured average values of a white (or gray) pixels area, allowing the white balance to adjust pre-component gains appropriately. Use function EasyImage::PixelAverage to obtain them. By default, no white balancing is performed.

*balanceGreen*

Color values to be used for white balance. These parameters should be set to the measured average values of a white (or gray) pixels area, allowing the white balance to adjust pre-component gains appropriately. Use function EasyImage::PixelAverage to obtain them. By default, no white balancing is performed.

*balanceBlue*

Color values to be used for white balance. These parameters should be set to the measured average values of a white (or gray) pixels area, allowing the white balance to adjust pre-component gains appropriately. Use function EasyImage::PixelAverage to obtain them. By default, no white balancing is performed.

### Remarks

To apply some transform to a color image, you initialize the color lookup once for all and use it at will with EColorLookup::Transform or EasyColor::TransformBayer operation.

# 4.50. EColorRangeThresholdSegmenter Class

Segments an image using a double threshold on a color image.

### Remarks

This segmenter is applicable to EROIC24 RGB color images. It produces coded images with two layers: The White layer (usually, with index 1) contains the unmasked pixels that belong to the cube of the RGB space that spans the low threshold point to the high threshold point; and the Black layer (usually, with index 0) contains the remaining unmasked pixels.

**Base Class:** ETwoLayersImageSegmenter

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

## Methods

| | |
|---|---|
| GetHighThreshold | Value of the high threshold. |
| GetLowThreshold | Value of the low threshold. |
| SetHighThreshold | Value of the high threshold. |

| | |
|---|---|
| SetLowThreshold | Value of the low threshold. |

# EColorRangeThresholdSegmenter::GetHighThreshold

# EColorRangeThresholdSegmenter::SetHighThreshold

Value of the high threshold.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
EC24 GetHighThreshold() const
void SetHighThreshold(EC24 threshold)
```

# EColorRangeThresholdSegmenter::GetLowThreshold

# EColorRangeThresholdSegmenter::SetLowThreshold

Value of the low threshold.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
EC24 GetLowThreshold() const
```

```
void SetLowThreshold(EC24 threshold)
```

# 4.51. EColorSingleThresholdSegmenter Class

Segments an image using a single threshold on a color image.

### Remarks

This segmenter is applicable to EROIC24 RGB color images. It produces coded images with two layers: The White layer (usually, with index 1) contains the unmasked pixels that belong to the cube of the RGB space defined by the threshold point and the white point **(255,255,255)**; and the Black layer (usually, with index 0) contains the remaining unmasked pixels.

**Base Class:** ETwoLayersImageSegmenter

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

## Methods

| | |
|---|---|
| GetThreshold | Value of the threshold. |
| SetThreshold | Value of the threshold. |

# EColorSingleThresholdSegmenter::GetThreshold

# EColorSingleThresholdSegmenter::SetThreshold

Value of the threshold.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
EC24 GetThreshold() const
void SetThreshold(EC24 threshold)
```

# 4.52. EColorVector Class

-

**Base Class:** EVector
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddElement | Appends (adds at the tail) an element to the vector. |
| EColorVector | - |
| GetElement | Returns the vector element at the given index. |
| GetRawDataPtr | Pointer to the vector data. |
| operator[] | Gives access to the vector element at the given index. |
| operator= | Copies all the data from another EColorVector object into the current EColorVector object |

| | |
|---|---|
| SetElement | Modifies the vector element at the given index by the given value. |

# ColorVector::AddElement

Appends (adds at the tail) an element to the vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddElement(
 EColor element
 )
```

## Parameters

*element*

The element to be added.

# EColorVector::EColorVector

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EColorVector(
 )
```

```
void EColorVector(
 OEV_UINT32 un32MaxElements
 )

void EColorVector(
 const EColorVector& other
 )
```

### Parameters

*un32MaxElements*

-

*other*

-

# EColorVector::GetElement

Returns the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EColor GetElement(
 OEV_INT32 index
 )
```

### Parameters

*index*

Index, between **0** and EColorVector (excluded) of the element to be accessed.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# EColorVector::operator[]

Gives access to the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EColor& operator[](
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index, between **0** and EColorVector (excluded) of the element to be accessed.

# EColorVector::operator=

Copies all the data from another EColorVector object into the current EColorVector object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EColorVector& operator=(
 const EColorVector& other
 )
```

### Parameters

*other*

EColorVector object to be copied

# EColorVector::GetRawDataPtr

Pointer to the vector data.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void* GetRawDataPtr() const
```

# EColorVector::SetElement

Modifies the vector element at the given index by the given value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetElement(
 OEV_INT32 index,
 EColor value
 )
```

## Parameters

*index*

Index, between **0** and EColorVector (excluded), of the element to be modified.

*value*

The new value for the element.

## Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# 4.53. EConverter Class

Conversion functions between bit depth formats of various 3D classes.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| Convert | Converts EDepthMap or EZMap between various formats. |

## Converter::Convert

Converts EDepthMap or EZMap between various formats.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Convert(
 const EDepthMap8& DepthMapIn,
 EDepthMap16& DepthMapOut,
 Euresys::Open_eVision_2_10::Easy3D::EMapConversionMode ConversionMode
 )
void Convert(
 const EDepthMap8& DepthMapIn,
 EDepthMap32f& DepthMapOut
 )
```

```
void Convert(
 const EDepthMap16& DepthMapIn,
 EDepthMap8& DepthMapOut,
 Euresys::Open_eVision_2_10::Easy3D::EMapConversionMode ConversionMode
 )

void Convert(
 const EDepthMap16& DepthMapIn,
 EDepthMap32f& DepthMapOut
 )

void Convert(
 const EDepthMap32f& DepthMapIn,
 EDepthMap8& DepthMapOut
 )

void Convert(
 const EDepthMap32f& DepthMapIn,
 EDepthMap16& DepthMapOut
 )

void Convert(
 const EZMap8& ZMapIn,
 EZMap16& ZMapOut,
 Euresys::Open_eVision_2_10::Easy3D::EMapConversionMode ConversionMode
 )

void Convert(
 const EZMap8& ZMapIn,
 EZMap32f& ZMapOut
 )

void Convert(
 const EZMap16& ZMapIn,
 EZMap8& ZMapOut,
 Euresys::Open_eVision_2_10::Easy3D::EMapConversionMode ConversionMode
 )

void Convert(
 const EZMap16& ZMapIn,
 EZMap32f& ZMapOut
 )

void Convert(
 const EZMap32f& ZMapIn,
 EZMap8& ZMapOut
 )
```

```
void Convert(
 const EZMap32f& MapIn,
 EZMap16& MapOut
 )
```

### Parameters

*DepthMapIn*

The input DepthMap (8, 16 or 32 bits)

*DepthMapOut*

The output DepthMap (8, 16 or 32 bits)

*ConversionMode*

The conversion mode from EMapConversionMode defines how to transform the pixel value from a format (8, 16 or 32 bits) to another.
EMapConversionMode_MaxDynamic maximizes the used range.
EMapConversionMode_Shift converts by bit shifting.
By default, the mode EMapConversionMode_MaxDynamic is selected.

*ZMapIn*

The input ZMap (8, 16 or 32 bits)

*ZMapOut*

The output ZMap (8, 16 or 32 bits)

*MapIn*

-

*MapOut*

-

### Remarks

Conversion from or to 32bits only supports EMapConversionMode_MaxDynamic mode. The undefined values are converted to the new map undefined value. For 8 and 16 bits images, the minimum defined value is 1, so the conversion of 32 bits images to 8 or 16 bits images adapts the dynamic range between 1 and the maximum value. For conversion to 32 bits float image, the used dynamic range is between 0 and FLOATMAX.

# 4.54. EDecimator Class

Decimation of a point cloud/Zmap/Depthmap.

**Derived Class(es):** ERandomDecimator

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| Decimate | Decimates a EPointCloud. |
| EDecimator | Creates an EDecimator object. |
| Load | Loads the decimator configuration. The given ESerializer must have been created for reading. |
| Save | Saves the decimator configuration. The given ESerializer must have been created for writing. |

D

# ecimator::Decimate

Decimates a EPointCloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Decimate(
 const EPointCloud& cloudIn,
 EPointCloud& cloudOut
 )
```

### Parameters

*cloudIn*

The input point cloud.

*cloudOut*

The output point cloud.

# EDecimator::EDecimator

Creates an EDecimator object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EDecimator(
 )
void EDecimator(
 const EDecimator& other
 )
```

### Parameters

*other*

-

# EDecimator::Load

Loads the decimator configuration. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

-

## EDecimator::Save

Saves the decimator configuration. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

-

# 4.55. EDepthMap Class

Represents a generic DepthMap type interface.

**Derived Class(es):** EDepthMap8 EDepthMap16 EDepthMap32f

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| AddMetadata | Adds a metadata key (name) and value. Overwrites if exists. |

| | |
|---|---|
| Clear | Clears the depth map: replaces all pixels with undefined value |
| ClearMetadata | Deletes all metadata. |
| ConvertCoordinatesMapToPixel | Converts 3D Map coordinates to Pixel coordinates. |
| ConvertCoordinatesPixelToMap | Converts Pixel coordinates to 3D Map coordinates. |
| DeleteMetadata | Deletes an existing metadata.<br>Throws an exception if it does not exist. |
| Draw | Draws a DepthMap in a device context. |
| DrawImage | Displays the internal image buffer |
| GetAxisSystemType | Manage the axis coordinate system. |
| GetBufferPtr | Retrieves the pointer of the pixel buffer. |
| GetCheckedBufferPtr | Retrieves the pointer of the pixel buffer. |
| GetHeight | Access depth map Height. |

| | |
|---|---|
| GetMetadata | Returns string value of the given metadata.<br>Throws an exception if it does not exist. |
| GetRowPitch | Returns the buffer row pitch. |
| GetType | Returns the image type. |
| GetWidth | Access depth map Width. |
| GetZResolution | Access the Z Resolution (depth units per grey scale value). |
| GetZValue | Gets the Z value of a pixel. |
| IsVoid | Tests if the EDepthMap object has a size of zero. |
| Load | Restores the EDepthMap stored in the given Open eVision file. |
| LoadImage | Restores the EDepthMap image stored in the given image file. |
| LoadImageAndMetadata | Loads the image and the metadata from a file in JSON format. |
| LoadMetadata | Loads the metadata from a file in JSON format. |

| | |
|---|---|
| ModifyMetadata | Changes the value of an existing metadata.<br>Throws an exception if the metadata does not exist. |
| Save | Saves the EDepthMap object to the given Open eVision file. |
| SaveImage | Saves the EDepthMap image to the given image file. |
| SaveImageAndMetadata | Saves the image and the metadata to a JSON format file. |
| SaveJpeg | Saves the EDepthMap object to the given image file, in JPEG format. |
| SaveJpeg2K | Saves the EDepthMap object to the given imagefile, in JPEG 2000 format. |
| SaveMetadata | Saves the metadata to a file in JSON format |
| Serialize | Serializes the object with all its attributes. |
| SerializeImage | Serializes the image associated to EDepthMap. |
| SetAxisSystemType | Manage the axis coordinate system. |
| SetBufferPtr | Sets the pointer to an externally allocated buffer. |

| | |
|---|---|
| SetHeight | Access depth map Height. |
| SetSize | Sets the width and height of a DepthMap. |
| SetWidth | Access depth map Width. |
| SetZResolution | Access the Z Resolution (depth units per grey scale value). |

# DepthMap::AddMetadata

Adds a metadata key (name) and value. Overwrites if exists.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void AddMetadata(
 const std::string& Key,
 const std::string& value
 )
```

## Parameters

*Key*

The name of the metadata. Names are unique.

*value*

The value for the given metadata.

# EDepthMap::GetAxisSystemType

# EDepthMap::SetAxisSystemType

Manage the axis coordinate system.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
Euresys::Open_eVision_2_10::Easy3D::EAxisSystemType GetAxisSystemType() const
void SetAxisSystemType(Euresys::Open_eVision_2_10::Easy3D::EAxisSystemType baseType)
```

# EDepthMap::Clear

Clears the depth map: replaces all pixels with undefined value

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Clear(
 )
```

# EDepthMap::ClearMetadata

Deletes all metadata.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ClearMetadata(
 )
```

# EDepthMap::ConvertCoordinatesMapToPixel

Converts 3D Map coordinates to Pixel coordinates.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool ConvertCoordinatesMapToPixel(
 float x3D,
 float y3D,
 int& xBuffer,
 int& yBuffer
 )
```

## Parameters

*x3D*

The Map X coordinate.

*y3D*

The Map Y coordinate.

*xBuffer*

The returned Pixel X coordinate.

*yBuffer*

The returned Pixel Y coordinate.

# EDepthMap::ConvertCoordinatesPixelToMap

Converts Pixel coordinates to 3D Map coordinates.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ConvertCoordinatesPixelToMap(
 int xBuffer,
 int yBuffer,
 float& x3D,
 float& y3D
 )
```

### Parameters

*xBuffer*

The pixel X coordinate.

*yBuffer*

The pixel Y coordinate.

*x3D*

The returned Map X coordinate.

*y3D*

The returned Map Y coordinate.

# EDepthMap::DeleteMetadata

Deletes an existing metadata.
Throws an exception if it does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void DeleteMetadata(
 const std::string& Key
 )
```

### Parameters

*Key*

The name of an exising metadata.

# EDepthMap::Draw

Draws a DepthMap in a device context.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

```
void Draw(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

### Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the depthmap is displayed in 1:1 scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Pan offset (in pixels) in the horizontal direction. By default, no panning is applied.

*panY*

Pan offset (in pixels) in the vertical direction. By default, no panning is applied.

*colorUndefinedPixel*

An optional parameter to choose the drawing color of undefined pixels.

*c24Vector*

When supplied, this parameter allows using a LUT that maps from Depth to C24 when drawing (false colors).

*bw8Vector*

When supplied, this parameter allows using a LUT that maps from Depth to BW8 when drawing.

### Remarks

A DepthMap can be drawn (its pixels rendered) using a device context.
The horizontal and vertical zooming factors can be different but must be in the **1/16..16** range.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EDepthMap::DrawImage

Displays the internal image buffer

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void DrawImage(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

```
void DrawImage(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the image is displayed in 1:1 scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Pan offset (in pixels) in the horizontal direction. By default, no panning is applied.

*panY*

Pan offset (in pixels) in the vertical direction. By default, no panning is applied.

*colorUndefinedPixel*

An optional parameter to choose the drawing color of undefined pixels.

*c24Vector*

When supplied, this parameter allows using a LUT that maps from Depth to C24 when drawing (false colors).

*bw8Vector*

When supplied, this parameter allows using a LUT that maps from Depth to BW8 when drawing.

## Remarks

An image can be drawn (its pixels rendered) using a device context.
The horizontal and vertical zooming factors can be different but must be in the **1/16..16** range.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EDepthMap::GetBufferPtr

Retrieves the pointer of the pixel buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void* GetBufferPtr(
 )
void* GetBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
const void* GetBufferPtr(
 )
const void* GetBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

## Parameters

*x*

Column of the pixel of which we want the address.

*y*

Row of the pixel of which we want the address.

## Remarks

This function does not check the value of the parameters.
Use carefully.

# EDepthMap::GetCheckedBufferPtr

Retrieves the pointer of the pixel buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void* GetCheckedBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
const void* GetCheckedBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel of which we want the address.

*y*

Row of the pixel of which we want the address.

### Remarks

This function checks the value of the parameters.

# EDepthMap::GetMetadata

Returns string value of the given metadata.
Throws an exception if it does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
std::string GetMetadata(
 const std::string& Key
 )
```

### Parameters

*Key*

The name of an exising metadata.

# EDepthMap::GetZValue

Gets the Z value of a pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetZValue(
 const OEV_INT32 x,
 const OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel.

*y*

Row of the pixel.

# EDepthMap::GetHeight

# EDepthMap::SetHeight

Access depth map Height.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetHeight() const
void SetHeight(OEV_INT32 height)
```

# EDepthMap::IsVoid

Tests if the EDepthMap object has a size of zero.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool IsVoid(
 )
```

## Remarks

Returns **TRUE** if the depthmap size is zero.

# EDepthMap::Load

Restores the EDepthMap stored in the given Open eVision file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 const std::string& path
 )
```

### Parameters

*path*

Full path of the file.

### Remarks

When loading, the depth map is resized if needed.
This function restores the depth map attributes.

# EDepthMap::LoadImage

Restores the EDepthMap image stored in the given image file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void LoadImage(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

### Remarks

When loading, the depth map is resized if need be.
This function does not restore the depth map attributes, only the image associated with the EDepthMap is updated.

# EDepthMap::LoadImageAndMetadata

Loads the image and the metadata from a file in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void LoadImageAndMetadata(
 const std::string& pathImage,
 const std::string& pathMetadata
 )
```

### Parameters

*pathImage*

Full path to the image file.

*pathMetadata*

Full path to the metadata file.

# EDepthMap::LoadMetadata

Loads the metadata from a file in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
void LoadMetadata(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

# EDepthMap::ModifyMetadata

Changes the value of an existing metadata.
Throws an exception if the metadata does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void ModifyMetadata(
 const std::string& Key,
 const std::string& value
 )
```

### Parameters

*Key*

The name of an exising metadata.

*value*

The value for the given metadata.

# EDepthMap::GetRowPitch

Returns the buffer row pitch.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetRowPitch() const
```

# EDepthMap::Save

Saves the EDepthMap object to the given Open eVision file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 const std::string& path
 )
```

### Parameters

*path*

The full path to the destination file.

### Remarks

This function saves the EDepthMap in a Open eVision file.
This function stores the depth map attributes.

# EDepthMap::SaveImage

Saves the EDepthMap image to the given image file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SaveImage(
 const std::string& path,
 Euresys::Open_eVision_2_10::EImageFileType type
 )
```

### Parameters

*path*

The full path to the destination file.

*type*

File format, as defined by EImageFileType.
If not specified, the file format is determined from the file extension.

### Remarks

This function saves the image associated to EDepthMap in a standard image file and thus does not store depth map attributes.

# EDepthMap::SaveImageAndMetadata

Saves the image and the metadata to a JSON format file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SaveImageAndMetadata(
 const std::string& pathImage,
 const std::string& pathMetadata,
 Euresys::Open_eVision_2_10::EImageFileType type
 )
```

### Parameters

*pathImage*

The full path to the destination image file.

*pathMetadata*

The full path to the destination metadata file.

*type*

File format, as defined by EImageFileType.
If not specified, the file format is determined from the file extension.

# EDepthMap::SaveJpeg

Saves the EDepthMap object to the given image file, in JPEG format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SaveJpeg(
 const std::string& path,
 int quality
 )
```

## Parameters

*path*

The full path of the destination file.

*quality*

JPEG quality, between 0 and 100 (100 is best quality). The default value is **75**.

# EDepthMap::SaveJpeg2K

Saves the EDepthMap object to the given imagefile, in JPEG 2000 format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SaveJpeg2K(
 const std::string& path,
 int quality
 )
```

### Parameters

*path*

The full path of the destination file.

*quality*

JPEG 2000 quality, between 1 and 512.
The default value is **16**.

# EDepthMap::SaveMetadata

Saves the metadata to a file in JSON format

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SaveMetadata(
 const std::string& path
 )
```

### Parameters

*path*

The full path to the destination file.

# EDepthMap::Serialize

Serializes the object with all its attributes.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or writen to.

# EDepthMap::SerializeImage

Serializes the image associated to EDepthMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SerializeImage(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or writen to.

# EDepthMap::SetBufferPtr

Sets the pointer to an externally allocated buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SetBufferPtr(
 OEV_INT32 width,
 OEV_INT32 height,
 void* imagePointer,
 OEV_INT32 bitsPerRow
 )
```

### Parameters

*width*

The width of the supplied buffer, in pixels.

*height*

The height of the supplied buffer, in pixels.

*imagePointer*

The pointer (aligned on 4 bytes) to the buffer, which must be large enough to hold the image.

*bitsPerRow*

The total number of bits contained in a row, padding included. Using the value **0** (default) means that this size is computed from the buffer width and the pixel size plus a padding with the smallest possible value that leads to a multiple of 4 bytes (32 bits), which is the minimum padding accepted by EDepthMap::SetBufferPtr.

# EDepthMap::SetSize

Sets the width and height of a DepthMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SetSize(
 OEV_INT32 width,
 OEV_INT32 height
 )

void SetSize(
 const EDepthMap& other
 )
```

### Parameters

*width*

The new requested DepthMap width.

*height*

The new requested DepthMap height.

*other*

The other DepthMap whose dimensions have to be used for the current object.

### Remarks

Open eVision will allocate a new buffer (deallocate the old buffer) if the supplied width and height are different from the existing ones.
If an external buffer has been specified by means of **SetImagePtr**, it will be kept only if the size does not change.
Creating a new buffer and setting its size creates a 4-byte aligned buffer, by default.

# EDepthMap::GetType

Returns the image type.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
Euresys::Open_eVision_2_10::EImageType GetType() const
```

# EDepthMap::GetWidth

# EDepthMap::SetWidth

Access depth map Width.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetWidth() const
void SetWidth(OEV_INT32 width)
```

## EDepthMap::GetZResolution

## EDepthMap::SetZResolution

Access the Z Resolution (depth units per grey scale value).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetZResolution() const
void SetZResolution(float resolution)
```

# 4.56. EDepthMap16 Class

Represents a EDepthMap with an 16-bit pixel internal representation.

**Base Class:** EDepthMap
**Derived Class(es):** EGrabberDepthMap16
**Namespace:** Euresys::Open_eVision_2_10::Easy3D

### Methods

| | |
|---|---|
| AddMetadata | Adds a metadata key (name) and value. Overwrites if exists. |

| | |
|---|---|
| AsEImage | Casts the EDepthMap16 to an EImageBW16 to use with the Open eVision 2D tools. |
| Clear | Clears the depth map: replaces all pixels with undefined value |
| ClearMetadata | Deletes all metadata. |
| ConvertCoordinatesMapToPixel | Converts 3D Map coordinates to Pixel coordinates. |
| ConvertCoordinatesPixelToMap | Converts Pixel coordinates to 3D Map coordinates. |
| CopyMetadataTo | Copies all metadatas to another EDepthMap16. |
| DeleteMetadata | Deletes an existing metadata.<br>Throws an exception if it does not exist. |
| Draw | Draws a DepthMap in a device context. |
| DrawImage | Displays the internal image buffer |
| EDepthMap16 | Creates a 16 bits EDepthMap. |
| FillUndefinedPixels | Fills undefined pixels, used to fill the "holes" in the depth map. |

| | |
|---|---|
| GetAxisSystemType | Manage the axis coordinate system. |
| GetBufferPtr | Retrieves the pointer of the pixel buffer. |
| GetCheckedBufferPtr | Retrieves the pointer of the pixel buffer. |
| GetHeight | Access depth map Height. |
| GetMetadata | Returns string value of the given metadata.<br>Throws an exception if it does not exist. |
| GetPixel | Gets the value of a pixel. |
| GetRowPitch | Returns the buffer row pitch. |
| GetType | Returns the image type. |
| GetUndefinedValue | Returns the Undefined value. That value is used to mark pixels with no valid depth value. |
| GetWidth | Access depth map Width. |
| GetZResolution | Access the Z Resolution (depth units per grey scale value). |

| | |
|---|---|
| GetZValue | Gets Z value of a pixel. |
| IsVoid | Tests if the EDepthMap16 object has a size of zero. |
| Load | Restores the EDepthMap16 stored in the given Open eVision file. |
| LoadImage | Restores the EDepthMap16 image stored in the given image file. |
| LoadImageAndMetadata | Loads the image and the metadata from a file in JSON format. |
| LoadMetadata | Loads the metadata from a file in JSON format. |
| ModifyMetadata | Changes the value of an existing metadata.<br>Throws an exception if the metadata does not exist. |
| operator= | Assignment operator. |
| Save | Saves the EDepthMap16 object to the given Open eVision file. |
| SaveImage | Saves the EDepthMap16 image to the given image file. |
| SaveImageAndMetadata | Saves the image and the metadata to a JSON format file. |

| | |
|---|---|
| SaveJpeg | Saves the EDepthMap16 object to the given image file, in JPEG format. |
| SaveJpeg2K | Saves the EDepthMap16 object to the given imagefile, in JPEG 2000 format. |
| SaveMetadata | Saves the metadata to a file in JSON format |
| Serialize | Serializes the object with all its attributes. |
| SerializeImage | Serializes the image associated to EDepthMap16. |
| SetAxisSystemType | Manage the axis coordinate system. |
| SetBufferPtr | Sets the pointer to an externally allocated buffer. |
| SetHeight | Access depth map Height. |
| SetPixel | Sets the value of a pixel. |
| SetSize | Sets the width and height of a DepthMap. |
| SetWidth | Access depth map Width. |

| | |
|---|---|
| SetZResolution | Access the Z Resolution (depth units per grey scale value). |

# DepthMap16::AddMetadata

Adds a metadata key (name) and value. Overwrites if exists.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void AddMetadata(
 const std::string& Key,
 const std::string& value
 )
```

## Parameters

*Key*

The name of the metadata. Names are unique.

*value*

The value for the given metadata.

# EDepthMap16::AsEImage

Casts the EDepthMap16 to an EImageBW16 to use with the Open eVision 2D tools.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
EImageBW16& AsEImage(
 )
```

# EDepthMap16::GetAxisSystemType

# EDepthMap16::SetAxisSystemType

Manage the axis coordinate system.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
Euresys::Open_eVision_2_10::Easy3D::EAxisSystemType GetAxisSystemType() const
void SetAxisSystemType(Euresys::Open_eVision_2_10::Easy3D::EAxisSystemType baseType)
```

# EDepthMap16::Clear

Clears the depth map: replaces all pixels with undefined value

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Clear(
 )
```

# EDepthMap16::ClearMetadata

Deletes all metadata.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ClearMetadata(
 )
```

# EDepthMap16::ConvertCoordinatesMapToPixel

Converts 3D Map coordinates to Pixel coordinates.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool ConvertCoordinatesMapToPixel(
 float x3D,
 float y3D,
 int& xBuffer,
 int& yBuffer
 )
```

## Parameters

*x3D*

The Map X coordinate.

*y3D*

The Map Y coordinate.

*xBuffer*

The returned Pixel X coordinate.

*yBuffer*

The returned Pixel Y coordinate.

# EDepthMap16::ConvertCoordinatesPixelToMap

Converts Pixel coordinates to 3D Map coordinates.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ConvertCoordinatesPixelToMap(
 int xBuffer,
 int yBuffer,
 float& x3D,
 float& y3D
 )
```

## Parameters

*xBuffer*

The pixel X coordinate.

*yBuffer*

The pixel Y coordinate.

*x3D*

The returned Map X coordinate.

*y3D*

The returned Map Y coordinate.

# EDepthMap16::CopyMetadataTo

Copies all metadatas to another EDepthMap16.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void CopyMetadataTo(
 EDepthMap16& other
 )
```

### Parameters

*other*

The destination EDepthMap16.

# EDepthMap16::DeleteMetadata

Deletes an existing metadata.
Throws an exception if it does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void DeleteMetadata(
 const std::string& Key
 )
```

### Parameters

*Key*

The name of an exising metadata.

# EDepthMap16::Draw

Draws a DepthMap in a device context.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void Draw(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

```
void Draw(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
void Draw(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the depthmap is displayed in 1:1 scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Pan offset (in pixels) in the horizontal direction. By default, no panning is applied.

*panY*

Pan offset (in pixels) in the vertical direction. By default, no panning is applied.

*colorUndefinedPixel*

An optional parameter to choose the drawing color of undefined pixels.

*c24Vector*

When supplied, this parameter allows using a LUT that maps from Depth to C24 when drawing (false colors).

*bw8Vector*

When supplied, this parameter allows using a LUT that maps from Depth to BW8 when drawing.

### Remarks

A DepthMap can be drawn (its pixels rendered) using a device context.
The horizontal and vertical zooming factors can be different but must be in the **1/16..16** range.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EDepthMap16::DrawImage

Displays the internal image buffer

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void DrawImage(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

```
void DrawImage(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the image is displayed in 1:1 scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Pan offset (in pixels) in the horizontal direction. By default, no panning is applied.

*panY*

Pan offset (in pixels) in the vertical direction. By default, no panning is applied.

*colorUndefinedPixel*

An optional parameter to choose the drawing color of undefined pixels.

*c24Vector*

When supplied, this parameter allows using a LUT that maps from Depth to C24 when drawing (false colors).

*bw8Vector*

When supplied, this parameter allows using a LUT that maps from Depth to BW8 when drawing.

### Remarks

An image can be drawn (its pixels rendered) using a device context.
The horizontal and vertical zooming factors can be different but must be in the **1/16..16** range.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EDepthMap16::EDepthMap16

Creates a 16 bits EDepthMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EDepthMap16(
 )
void EDepthMap16(
 OEV_INT32 width,
 OEV_INT32 height
 )
void EDepthMap16(
 const EDepthMap16& other
 )
```

### Parameters

*width*

The width of the new depth map.

*height*

The height of the new depth map.

*other*

Another depth map.

# EDepthMap16::FillUndefinedPixels

Fills undefined pixels, used to fill the "holes" in the depth map.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void FillUndefinedPixels(
 EDepthMap16& outMap,
 Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsDirection direction,
 Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsMethod method
 )
```

### Parameters

*outMap*

The destination depth map.

*direction*

Direction in which the undefined pixels are filled in a depthmap from EFillUndefinedPixelsDirection.

*method*

Which values used to fill the undefined pixels in a depthmap from EFillUndefinedPixelsMethod.

# EDepthMap16::GetBufferPtr

Retrieves the pointer of the pixel buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void* GetBufferPtr(
 )
```

```
void* GetBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )

const void* GetBufferPtr(
 )

const void* GetBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel of which we want the address.

*y*

Row of the pixel of which we want the address.

### Remarks

This function does not check the value of the parameters.
Use carefully.

# EDepthMap16::GetCheckedBufferPtr

Retrieves the pointer of the pixel buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void* GetCheckedBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

```
const void* GetCheckedBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel of which we want the address.

*y*

Row of the pixel of which we want the address.

### Remarks

This function checks the value of the parameters.

# EDepthMap16::GetMetadata

Returns string value of the given metadata.
Throws an exception if it does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
std::string GetMetadata(
 const std::string& Key
 )
```

### Parameters

*Key*

The name of an exising metadata.

# EDepthMap16::GetPixel

Gets the value of a pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EDepth16 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

## Parameters

*x*

Column of the pixel.

*y*

Row of the pixel.

# EDepthMap16::GetZValue

Gets Z value of a pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetZValue(
 const OEV_INT32 x,
 const OEV_INT32 y
 )
```

## Parameters

*x*

Column of the pixel.

*y*

Row of the pixel.

# EDepthMap16::GetHeight

# EDepthMap16::SetHeight

Access depth map Height.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetHeight() const
void SetHeight(OEV_INT32 height)
```

# EDepthMap16::IsVoid

Tests if the EDepthMap16 object has a size of zero.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool IsVoid(
 )
```

## Remarks

Returns **TRUE** if the depthmap size is zero.

# EDepthMap16::Load

Restores the EDepthMap16 stored in the given Open eVision file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 const std::string& path
 )
```

### Parameters

*path*

Full path of the file.

### Remarks

When loading, the depth map is resized if needed.
This function restores the depth map attributes.

# EDepthMap16::LoadImage

Restores the EDepthMap16 image stored in the given image file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void LoadImage(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

### Remarks

When loading, the depth map is resized if need be.
This function does not restore the depth map attributes, only the image associated with the EDepthMap16 is updated.

# EDepthMap16::LoadImageAndMetadata

Loads the image and the metadata from a file in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void LoadImageAndMetadata(
 const std::string& pathImage,
 const std::string& pathMetadata
 )
```

### Parameters

*pathImage*

Full path to the image file.

*pathMetadata*

Full path to the metadata file.

# EDepthMap16::LoadMetadata

Loads the metadata from a file in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
void LoadMetadata(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

# EDepthMap16::ModifyMetadata

Changes the value of an existing metadata.
Throws an exception if the metadata does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void ModifyMetadata(
 const std::string& Key,
 const std::string& value
 )
```

### Parameters

*Key*

The name of an exising metadata.

*value*

The value for the given metadata.

# EDepthMap16::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EDepthMap16& operator=(
 const EDepthMap16& other
 )
```

### Parameters

*other*

The source EDepthMap16.

# EDepthMap16::GetRowPitch

Returns the buffer row pitch.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetRowPitch() const
```

# EDepthMap16::Save

Saves the EDepthMap16 object to the given Open eVision file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 const std::string& path
 )
```

### Parameters

*path*

The full path to the destination file.

### Remarks

This function saves the EDepthMap16 in a Open eVision file.
This function stores the depth map attributes.

# EDepthMap16::SaveImage

Saves the EDepthMap16 image to the given image file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveImage(
 const std::string& path,
 Euresys::Open_eVision_2_10::EImageFileType type
 )
```

### Parameters

*path*

The full path to the destination file.

*type*

File format, as defined by EImageFileType.
If not specified, the file format is determined from the file extension.

### Remarks

This function saves the image associated to EDepthMap16 in a standard image file and thus does not store depth map attributes.

# EDepthMap16::SaveImageAndMetadata

Saves the image and the metadata to a JSON format file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveImageAndMetadata(
 const std::string& pathImage,
 const std::string& pathMetadata,
 Euresys::Open_eVision_2_10::EImageFileType type
 )
```

### Parameters

*pathImage*

The full path to the destination image file.

*pathMetadata*

The full path to the destination metadata file.

*type*

File format, as defined by EImageFileType.
If not specified, the file format is determined from the file extension.

# EDepthMap16::SaveJpeg

Saves the EDepthMap16 object to the given image file, in JPEG format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveJpeg(
 const std::string& path,
 int quality
 )
```

### Parameters

*path*

The full path of the destination file.

*quality*

JPEG quality, between 0 and 100 (100 is best quality). The default value is **75**.

# EDepthMap16::SaveJpeg2K

Saves the EDepthMap16 object to the given imagefile, in JPEG 2000 format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveJpeg2K(
 const std::string& path,
 int quality
 )
```

### Parameters

*path*

The full path of the destination file.

*quality*

JPEG 2000 quality, between 1 and 512.
The default value is **16**.

# EDepthMap16::SaveMetadata

Saves the metadata to a file in JSON format

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveMetadata(
 const std::string& path
 )
```

### Parameters

*path*

The full path to the destination file.

# EDepthMap16::Serialize

Serializes the object with all its attributes.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or writen to.

# EDepthMap16::SerializeImage

Serializes the image associated to EDepthMap16.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SerializeImage(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or writen to.

# EDepthMap16::SetBufferPtr

Sets the pointer to an externally allocated buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetBufferPtr(
 OEV_INT32 width,
 OEV_INT32 height,
 void* imagePointer,
 OEV_INT32 bitsPerRow
 )
```

### Parameters

*width*

The width of the supplied buffer, in pixels.

*height*

The height of the supplied buffer, in pixels.

*imagePointer*

The pointer (aligned on 4 bytes) to the buffer, which must be large enough to hold the image.

*bitsPerRow*

The total number of bits contained in a row, padding included. Using the value **0** (default) means that this size is computed from the buffer width and the pixel size plus a padding with the smallest possible value that leads to a multiple of 4 bytes (32 bits), which is the minimum padding accepted by EDepthMap16::SetBufferPtr.

# EDepthMap16::SetPixel

Sets the value of a pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetPixel(
 EDepth16 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

Value of the pixel.

*x*

Column of the pixel.

*y*

Row of the pixel.

# EDepthMap16::SetSize

Sets the width and height of a DepthMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetSize(
 OEV_INT32 width,
 OEV_INT32 height
 )

void SetSize(
 const EDepthMap& other
 )
```

### Parameters

*width*

The new requested DepthMap width.

*height*

The new requested DepthMap height.

*other*

The other DepthMap whose dimensions have to be used for the current object.

### Remarks

Open eVision will allocate a new buffer (deallocate the old buffer) if the supplied width and height are different from the existing ones.
If an external buffer has been specified by means of **SetImagePtr**, it will be kept only if the size does not change.
Creating a new buffer and setting its size creates a 4-byte aligned buffer, by default.

# EDepthMap16::GetType

Returns the image type.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
Euresys::Open_eVision_2_10::EImageType GetType() const
```

# EDepthMap16::GetUndefinedValue

Returns the Undefined value. That value is used to mark pixels with no valid depth value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
EDepth16 GetUndefinedValue() const
```

# EDepthMap16::GetWidth

# EDepthMap16::SetWidth

Access depth map Width.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetWidth() const
void SetWidth(OEV_INT32 width)
```

# EDepthMap16::GetZResolution

# EDepthMap16::SetZResolution

Access the Z Resolution (depth units per grey scale value).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetZResolution() const
void SetZResolution(float resolution)
```

# 4.57. EDepthMap32f Class

Represents a EDepthMap with an 32-bit pixel internal representation.

**Base Class:** EDepthMap

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| AddMetadata | Adds a metadata key (name) and value. Overwrites if exists. |
| AsEImage | Casts the EDepthMap32f to an EImageBW32f to use with the Open eVision 2D tools. |
| Clear | Clears the depth map: replaces all pixels with undefined value |
| ClearMetadata | Deletes all metadata. |
| ConvertCoordinatesMapToPixel | Converts 3D Map coordinates to Pixel coordinates. |
| ConvertCoordinatesPixelToMap | Converts Pixel coordinates to 3D Map coordinates. |
| CopyMetadataTo | Copies all metadatas to another EDepthMap32f. |

| | |
|---|---|
| DeleteMetadata | Deletes an existing metadata.<br>Throws an exception if it does not exist. |
| Draw | Draws a DepthMap in a device context. |
| DrawImage | Displays the internal image buffer |
| EDepthMap32f | Creates a 32 bits EDepthMap. |
| FillUndefinedPixels | Fills undefined pixels, used to fill the "holes" in the depth map. |
| GetAxisSystemType | Manage the axis coordinate system. |
| GetBufferPtr | Retrieves the pointer of the pixel buffer. |
| GetCheckedBufferPtr | Retrieves the pointer of the pixel buffer. |
| GetHeight | Access depth map Height. |
| GetMetadata | Returns string value of the given metadata.<br>Throws an exception if it does not exist. |
| GetPixel | Gets the value of a pixel. |

| | |
|---|---|
| GetRowPitch | Returns the buffer row pitch. |
| GetType | Returns the image type. |
| GetUndefinedValue | Returns the Undefined value. That value is used to mark pixels with no valid depth value. |
| GetWidth | Access depth map Width. |
| GetZResolution | Access the Z Resolution (depth units per grey scale value). |
| GetZValue | Gets Z value of a pixel. |
| IsVoid | Tests if the EDepthMap32f object has a size of zero. |
| Load | Restores the EDepthMap32f stored in the given Open eVision file. |
| LoadImage | Restores the EDepthMap32f image stored in the given image file. |
| LoadImageAndMetadata | Loads the image and the metadata from a file in JSON format. |
| LoadMetadata | Loads the metadata from a file in JSON format. |

| | |
|---|---|
| ModifyMetadata | Changes the value of an existing metadata.<br>Throws an exception if the metadata does not exist. |
| operator= | Assignment operator. |
| Save | Saves the EDepthMap32f object to the given Open eVision file. |
| SaveImage | Saves the EDepthMap32f image to the given image file. |
| SaveImageAndMetadata | Saves the image and the metadata to a JSON format file. |
| SaveJpeg | Saves the EDepthMap32f object to the given image file, in JPEG format. |
| SaveJpeg2K | Saves the EDepthMap32f object to the given imagefile, in JPEG 2000 format. |
| SaveMetadata | Saves the metadata to a file in JSON format |
| Serialize | Serializes the object with all its attributes. |
| SerializeImage | Serializes the image associated to EDepthMap32f. |
| SetAxisSystemType | Manage the axis coordinate system. |

| | |
|---|---|
| SetBufferPtr | Sets the pointer to an externally allocated buffer. |
| SetHeight | Access depth map Height. |
| SetPixel | Sets the value of a pixel. |
| SetSize | Sets the width and height of a DepthMap. |
| SetWidth | Access depth map Width. |
| SetZResolution | Access the Z Resolution (depth units per grey scale value). |

# DepthMap32f::AddMetadata

Adds a metadata key (name) and value. Overwrites if exists.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void AddMetadata(
 const std::string& Key,
 const std::string& value
 )
```

## Parameters

*Key*

The name of the metadata. Names are unique.

*value*

The value for the given metadata.

# EDepthMap32f::AsEImage

Casts the EDepthMap32f to an EImageBW32f to use with the Open eVision 2D tools.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EImageBW32f& AsEImage(
 )
```

# EDepthMap32f::GetAxisSystemType

# EDepthMap32f::SetAxisSystemType

Manage the axis coordinate system.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
Euresys::Open_eVision_2_10::Easy3D::EAxisSystemType GetAxisSystemType() const
void SetAxisSystemType(Euresys::Open_eVision_2_10::Easy3D::EAxisSystemType baseType)
```

# EDepthMap32f::Clear

Clears the depth map: replaces all pixels with undefined value

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Clear(
 )
```

# EDepthMap32f::ClearMetadata

Deletes all metadata.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ClearMetadata(
 )
```

# EDepthMap32f::ConvertCoordinatesMapToPixel

Converts 3D Map coordinates to Pixel coordinates.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool ConvertCoordinatesMapToPixel(
 float x3D,
 float y3D,
 int& xBuffer,
 int& yBuffer
 )
```

### Parameters

*x3D*

The Map X coordinate.

*y3D*

The Map Y coordinate.

*xBuffer*

The returned Pixel X coordinate.

*yBuffer*

The returned Pixel Y coordinate.

# EDepthMap32f::ConvertCoordinatesPixelToMap

Converts Pixel coordinates to 3D Map coordinates.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ConvertCoordinatesPixelToMap(
 int xBuffer,
 int yBuffer,
 float& x3D,
 float& y3D
 )
```

### Parameters

*xBuffer*

The pixel X coordinate.

*yBuffer*

The pixel Y coordinate.

*x3D*

The returned Map X coordinate.

*y3D*

The returned Map Y coordinate.

# EDepthMap32f::CopyMetadataTo

Copies all metadatas to another EDepthMap32f.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void CopyMetadataTo(
 EDepthMap32f& other
 )
```

## Parameters

*other*

The destination EDepthMap32f.

# EDepthMap32f::DeleteMetadata

Deletes an existing metadata.
Throws an exception if it does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void DeleteMetadata(
 const std::string& Key
 )
```

### Parameters

*Key*

The name of an exising metadata.

# EDepthMap32f::Draw

Draws a DepthMap in a device context.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

```
void Draw(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

### Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the depthmap is displayed in 1:1 scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Pan offset (in pixels) in the horizontal direction. By default, no panning is applied.

*panY*

Pan offset (in pixels) in the vertical direction. By default, no panning is applied.

*colorUndefinedPixel*

An optional parameter to choose the drawing color of undefined pixels.

*c24Vector*

When supplied, this parameter allows using a LUT that maps from Depth to C24 when drawing (false colors).

*bw8Vector*

When supplied, this parameter allows using a LUT that maps from Depth to BW8 when drawing.

### Remarks

A DepthMap can be drawn (its pixels rendered) using a device context.
The horizontal and vertical zooming factors can be different but must be in the **1/16..16** range.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EDepthMap32f::DrawImage

Displays the internal image buffer

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void DrawImage(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

```
void DrawImage(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the image is displayed in 1:1 scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Pan offset (in pixels) in the horizontal direction. By default, no panning is applied.

*panY*

Pan offset (in pixels) in the vertical direction. By default, no panning is applied.

*colorUndefinedPixel*

An optional parameter to choose the drawing color of undefined pixels.

*c24Vector*

When supplied, this parameter allows using a LUT that maps from Depth to C24 when drawing (false colors).

*bw8Vector*

When supplied, this parameter allows using a LUT that maps from Depth to BW8 when drawing.

### Remarks

An image can be drawn (its pixels rendered) using a device context.
The horizontal and vertical zooming factors can be different but must be in the **1/16..16** range.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EDepthMap32f::EDepthMap32f

Creates a 32 bits EDepthMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EDepthMap32f(
 )
void EDepthMap32f(
 OEV_INT32 width,
 OEV_INT32 height
 )
void EDepthMap32f(
 const EDepthMap32f& other
 )
```

### Parameters

*width*

The width of the new depth map.

*height*

The height of the new depth map.

*other*

Another depth map.

# EDepthMap32f::FillUndefinedPixels

Fills undefined pixels, used to fill the "holes" in the depth map.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void FillUndefinedPixels(
 EDepthMap32f& outMap,
 Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsDirection direction,
 Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsMethod method
 )
```

### Parameters

*outMap*

The destination depth map

*direction*

Direction in which the undefined pixels are filled in a depthmap from EFillUndefinedPixelsDirection.

*method*

Which values used to fill the undefined pixels in a depthmap from EFillUndefinedPixelsMethod.

# EDepthMap32f::GetBufferPtr

Retrieves the pointer of the pixel buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void* GetBufferPtr(
 )

void* GetBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )

const void* GetBufferPtr(
 )
```

```
const void* GetBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel of which we want the address.

*y*

Row of the pixel of which we want the address.

### Remarks

This function does not check the value of the parameters.
Use carefully.

# EDepthMap32f::GetCheckedBufferPtr

Retrieves the pointer of the pixel buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void* GetCheckedBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
const void* GetCheckedBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel of which we want the address.

*y*

Row of the pixel of which we want the address.

### Remarks

This function checks the value of the parameters.

# EDepthMap32f::GetMetadata

Returns string value of the given metadata.
Throws an exception if it does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
std::string GetMetadata(
 const std::string& Key
 )
```

### Parameters

*Key*

The name of an exising metadata.

# EDepthMap32f::GetPixel

Gets the value of a pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EDepth32f GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel.

*y*

Row of the pixel.

# EDepthMap32f::GetZValue

Gets Z value of a pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetZValue(
 const OEV_INT32 x,
 const OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel.

*y*

Row of the pixel.

# EDepthMap32f::GetHeight

# EDepthMap32f::SetHeight

Access depth map Height.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetHeight() const
void SetHeight(OEV_INT32 height)
```

# EDepthMap32f::IsVoid

Tests if the EDepthMap32f object has a size of zero.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool IsVoid(
 )
```

## Remarks

Returns **TRUE** if the depthmap size is zero.

# EDepthMap32f::Load

Restores the EDepthMap32f stored in the given Open eVision file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 const std::string& path
 )
```

### Parameters

*path*

Full path of the file.

### Remarks

When loading, the depth map is resized if needed.
This function restores the depth map attributes.

# EDepthMap32f::LoadImage

Restores the EDepthMap32f image stored in the given image file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void LoadImage(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

### Remarks

When loading, the depth map is resized if need be.
This function does not restore the depth map attributes, only the image associated with the EDepthMap32f is updated.

# EDepthMap32f::LoadImageAndMetadata

Loads the image and the metadata from a file in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void LoadImageAndMetadata(
 const std::string& pathImage,
 const std::string& pathMetadata
 )
```

### Parameters

*pathImage*

Full path to the image file.

*pathMetadata*

Full path to the metadata file.

# EDepthMap32f::LoadMetadata

Loads the metadata from a file in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void LoadMetadata(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

# EDepthMap32f::ModifyMetadata

Changes the value of an existing metadata.
Throws an exception if the metadata does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ModifyMetadata(
 const std::string& Key,
 const std::string& value
 )
```

### Parameters

*Key*

The name of an exising metadata.

*value*

The value for the given metadata.

# EDepthMap32f::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EDepthMap32f& operator=(
 const EDepthMap32f& other
 )
```

### Parameters

*other*

The source EDepthMap32f.

# EDepthMap32f::GetRowPitch

Returns the buffer row pitch.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetRowPitch() const
```

# EDepthMap32f::Save

Saves the EDepthMap32f object to the given Open eVision file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 const std::string& path
 )
```

## Parameters

*path*

The full path to the destination file.

## Remarks

This function saves the EDepthMap32f in a Open eVision file.
This function stores the depth map attributes.

# EDepthMap32f::SaveImage

Saves the EDepthMap32f image to the given image file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveImage(
 const std::string& path,
 Euresys::Open_eVision_2_10::EImageFileType type
 )
```

## Parameters

*path*

The full path to the destination file.

*type*

File format, as defined by EImageFileType.
If not specified, the file format is determined from the file extension.

## Remarks

This function saves the image associated to EDepthMap32f in a standard image file and thus does not store depth map attributes.

# EDepthMap32f::SaveImageAndMetadata

Saves the image and the metadata to a JSON format file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
void SaveImageAndMetadata(
 const std::string& pathImage,
 const std::string& pathMetadata,
 Euresys::Open_eVision_2_10::EImageFileType type
 )
```

### Parameters

*pathImage*

The full path to the destination image file.

*pathMetadata*

The full path to the destination metadata file.

*type*

File format, as defined by EImageFileType.
If not specified, the file format is determined from the file extension.

# EDepthMap32f::SaveJpeg

Saves the EDepthMap32f object to the given image file, in JPEG format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SaveJpeg(
 const std::string& path,
 int quality
 )
```

### Parameters

*path*

The full path of the destination file.

*quality*

JPEG quality, between 0 and 100 (100 is best quality). The default value is **75**.

# EDepthMap32f::SaveJpeg2K

Saves the EDepthMap32f object to the given imagefile, in JPEG 2000 format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveJpeg2K(
 const std::string& path,
 int quality
 )
```

## Parameters

*path*

The full path of the destination file.

*quality*

JPEG 2000 quality, between 1 and 512.
The default value is **16**.

# EDepthMap32f::SaveMetadata

Saves the metadata to a file in JSON format

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveMetadata(
 const std::string& path
 )
```

## Parameters

*path*

The full path to the destination file.

# EDepthMap32f::Serialize

Serializes the object with all its attributes.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or writen to.

# EDepthMap32f::SerializeImage

Serializes the image associated to EDepthMap32f.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SerializeImage(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or writen to.

# EDepthMap32f::SetBufferPtr

Sets the pointer to an externally allocated buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetBufferPtr(
 OEV_INT32 width,
 OEV_INT32 height,
 void* imagePointer,
 OEV_INT32 bitsPerRow
 )
```

### Parameters

*width*

The width of the supplied buffer, in pixels.

*height*

The height of the supplied buffer, in pixels.

*imagePointer*

The pointer (aligned on 4 bytes) to the buffer, which must be large enough to hold the image.

*bitsPerRow*

The total number of bits contained in a row, padding included. Using the value **0** (default) means that this size is computed from the buffer width and the pixel size plus a padding with the smallest possible value that leads to a multiple of 4 bytes (32 bits), which is the minimum padding accepted by EDepthMap32f::SetBufferPtr.

# EDepthMap32f::SetPixel

Sets the value of a pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetPixel(
 EDepth32f value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

Value of the pixel.

*x*

Column of the pixel.

*y*

Row of the pixel.

# EDepthMap32f::SetSize

Sets the width and height of a DepthMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetSize(
 OEV_INT32 width,
 OEV_INT32 height
 )
void SetSize(
 const EDepthMap& other
 )
```

### Parameters

*width*

The new requested DepthMap width.

*height*

The new requested DepthMap height.

*other*

The other DepthMap whose dimensions have to be used for the current object.

### Remarks

Open eVision will allocate a new buffer (deallocate the old buffer) if the supplied width and height are different from the existing ones.
If an external buffer has been specified by means of **SetImagePtr**, it will be kept only if the size does not change.
Creating a new buffer and setting its size creates a 4-byte aligned buffer, by default.

# EDepthMap32f::GetType

Returns the image type.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
Euresys::Open_eVision_2_10::EImageType GetType() const
```

# EDepthMap32f::GetUndefinedValue

Returns the Undefined value. That value is used to mark pixels with no valid depth value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
EDepth32f GetUndefinedValue() const
```

# EDepthMap32f::GetWidth

# EDepthMap32f::SetWidth

Access depth map Width.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetWidth() const
void SetWidth(OEV_INT32 width)
```

# EDepthMap32f::GetZResolution

# EDepthMap32f::SetZResolution

Access the Z Resolution (depth units per grey scale value).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetZResolution() const
void SetZResolution(float resolution)
```

# 4.58. EDepthMap8 Class

Represents a EDepthMap with an internal 8-bit pixel representation.

**Base Class:** EDepthMap
**Derived Class(es):** EGrabberDepthMap8
**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| AddMetadata | Adds a metadata key (name) and value. Overwrites if exists. |
| AsEImage | Casts the EDepthMap8 to an EImageBW8 to use with the Open eVision 2D tools. |
| Clear | Clears the depth map: replaces all pixels with undefined value |
| ClearMetadata | Deletes all metadata. |
| ConvertCoordinatesMapToPixel | Converts 3D Map coordinates to Pixel coordinates. |
| ConvertCoordinatesPixelToMap | Converts Pixel coordinates to 3D Map coordinates. |
| CopyMetadataTo | Copies all metadatas to another EDepthMap8. |

| | |
|---|---|
| DeleteMetadata | Deletes an existing metadata.<br>Throws an exception if it does not exist. |
| Draw | Draws a DepthMap in a device context. |
| DrawImage | Displays the internal image buffer |
| EDepthMap8 | Creates a 8 bits EDepthMap. |
| FillUndefinedPixels | Fills undefined pixels, used to fill the "holes" in the depth map. |
| GetAxisSystemType | Manage the axis coordinate system. |
| GetBufferPtr | Retrieves the pointer of the pixel buffer. |
| GetCheckedBufferPtr | Retrieves the pointer of the pixel buffer. |
| GetHeight | Access depth map Height. |
| GetMetadata | Returns string value of the given metadata.<br>Throws an exception if it does not exist. |
| GetPixel | Gets the value of a pixel. |

| | |
|---|---|
| GetRowPitch | Returns the buffer row pitch. |
| GetType | Returns the image type. |
| GetUndefinedValue | Returns the Undefined value. That value is used to mark pixels with no valid depth value. |
| GetWidth | Access depth map Width. |
| GetZResolution | Access the Z Resolution (depth units per grey scale value). |
| GetZValue | Gets Z value of a pixel. |
| IsVoid | Tests if the EDepthMap8 object has a size of zero. |
| Load | Restores the EDepthMap8 stored in the given Open eVision file. |
| LoadImage | Restores the EDepthMap8 image stored in the given image file. |
| LoadImageAndMetadata | Loads the image and the metadata from a file in JSON format. |
| LoadMetadata | Loads the metadata from a file in JSON format. |

| | |
|---|---|
| ModifyMetadata | Changes the value of an existing metadata.<br>Throws an exception if the metadata does not exist. |
| operator= | Assignment operator. |
| Save | Saves the EDepthMap8 object to the given Open eVision file. |
| SaveImage | Saves the EDepthMap8 image to the given image file. |
| SaveImageAndMetadata | Saves the image and the metadata to a JSON format file. |
| SaveJpeg | Saves the EDepthMap8 object to the given image file, in JPEG format. |
| SaveJpeg2K | Saves the EDepthMap8 object to the given imagefile, in JPEG 2000 format. |
| SaveMetadata | Saves the metadata to a file in JSON format |
| Serialize | Serializes the object with all its attributes. |
| SerializeImage | Serializes the image associated to EDepthMap8. |
| SetAxisSystemType | Manage the axis coordinate system. |

| | |
|---|---|
| SetBufferPtr | Sets the pointer to an externally allocated buffer. |
| SetHeight | Access depth map Height. |
| SetPixel | Sets the value of a pixel. |
| SetSize | Sets the width and height of a DepthMap. |
| SetWidth | Access depth map Width. |
| SetZResolution | Access the Z Resolution (depth units per grey scale value). |

# DepthMap8::AddMetadata

Adds a metadata key (name) and value. Overwrites if exists.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void AddMetadata(
 const std::string& Key,
 const std::string& value
 )
```

## Parameters

*Key*

The name of the metadata. Names are unique.

*value*

The value for the given metadata.

# EDepthMap8::AsEImage

Casts the EDepthMap8 to an EImageBW8 to use with the Open eVision 2D tools.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EImageBW8& AsEImage(
 )
```

# EDepthMap8::GetAxisSystemType

# EDepthMap8::SetAxisSystemType

Manage the axis coordinate system.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
Euresys::Open_eVision_2_10::Easy3D::EAxisSystemType GetAxisSystemType() const
void SetAxisSystemType(Euresys::Open_eVision_2_10::Easy3D::EAxisSystemType baseType)
```

# EDepthMap8::Clear

Clears the depth map: replaces all pixels with undefined value

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Clear(
 )
```

# EDepthMap8::ClearMetadata

Deletes all metadata.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ClearMetadata(
 )
```

# EDepthMap8::ConvertCoordinatesMapToPixel

Converts 3D Map coordinates to Pixel coordinates.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool ConvertCoordinatesMapToPixel(
 float x3D,
 float y3D,
 int& xBuffer,
 int& yBuffer
 )
```

### Parameters

*x3D*

The Map X coordinate.

*y3D*

The Map Y coordinate.

*xBuffer*

The returned Pixel X coordinate.

*yBuffer*

The returned Pixel Y coordinate.

# EDepthMap8::ConvertCoordinatesPixelToMap

Converts Pixel coordinates to 3D Map coordinates.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ConvertCoordinatesPixelToMap(
 int xBuffer,
 int yBuffer,
 float& x3D,
 float& y3D
 )
```

### Parameters

*xBuffer*

The pixel X coordinate.

*yBuffer*

The pixel Y coordinate.

*x3D*

The returned Map X coordinate.

*y3D*

The returned Map Y coordinate.

# EDepthMap8::CopyMetadataTo

Copies all metadatas to another EDepthMap8.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void CopyMetadataTo(
 EDepthMap8& other
 )
```

## Parameters

*other*

The destination EDepthMap8.

# EDepthMap8::DeleteMetadata

Deletes an existing metadata.
Throws an exception if it does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void DeleteMetadata(
 const std::string& Key
 )
```

### Parameters

*Key*

The name of an exising metadata.

# EDepthMap8::Draw

Draws a DepthMap in a device context.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

```
void Draw(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

### Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the depthmap is displayed in 1:1 scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Pan offset (in pixels) in the horizontal direction. By default, no panning is applied.

*panY*

Pan offset (in pixels) in the vertical direction. By default, no panning is applied.

*colorUndefinedPixel*

An optional parameter to choose the drawing color of undefined pixels.

*c24Vector*

When supplied, this parameter allows using a LUT that maps from Depth to C24 when drawing (false colors).

*bw8Vector*

When supplied, this parameter allows using a LUT that maps from Depth to BW8 when drawing.

### Remarks

A DepthMap can be drawn (its pixels rendered) using a device context.
The horizontal and vertical zooming factors can be different but must be in the **1/16..16** range.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EDepthMap8::DrawImage

Displays the internal image buffer

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void DrawImage(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

```
void DrawImage(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the image is displayed in 1:1 scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Pan offset (in pixels) in the horizontal direction. By default, no panning is applied.

*panY*

Pan offset (in pixels) in the vertical direction. By default, no panning is applied.

*colorUndefinedPixel*

An optional parameter to choose the drawing color of undefined pixels.

*c24Vector*

When supplied, this parameter allows using a LUT that maps from Depth to C24 when drawing (false colors).

*bw8Vector*

When supplied, this parameter allows using a LUT that maps from Depth to BW8 when drawing.

### Remarks

An image can be drawn (its pixels rendered) using a device context.
The horizontal and vertical zooming factors can be different but must be in the **1/16..16** range.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EDepthMap8::EDepthMap8

Creates a 8 bits EDepthMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EDepthMap8(
 )
void EDepthMap8(
 OEV_INT32 width,
 OEV_INT32 height
 )
void EDepthMap8(
 const EDepthMap8& other
 )
```

### Parameters

*width*

The width of the new depth map.

*height*

The height of the new depth map.

*other*

Another depth map.

# EDepthMap8::FillUndefinedPixels

Fills undefined pixels, used to fill the "holes" in the depth map.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void FillUndefinedPixels(
 EDepthMap8& outMap,
 Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsDirection direction,
 Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsMethod method
 )
```

### Parameters

*outMap*

The destination depth map.

*direction*

Direction in which the undefined pixels are filled in a depthmap from EFillUndefinedPixelsDirection.

*method*

Which values used to fill the undefined pixels in a depthmap from EFillUndefinedPixelsMethod.

# EDepthMap8::GetBufferPtr

Retrieves the pointer of the pixel buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void* GetBufferPtr(
 )

void* GetBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )

const void* GetBufferPtr(
 )
```

```
const void* GetBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel of which we want the address.

*y*

Row of the pixel of which we want the address.

### Remarks

This function does not check the value of the parameters.
Use carefully.

# EDepthMap8::GetCheckedBufferPtr

Retrieves the pointer of the pixel buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void* GetCheckedBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
const void* GetCheckedBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel of which we want the address.

*y*

Row of the pixel of which we want the address.

### Remarks

This function checks the value of the parameters.

# EDepthMap8::GetMetadata

Returns string value of the given metadata.
Throws an exception if it does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
std::string GetMetadata(
 const std::string& Key
 )
```

### Parameters

*Key*

The name of an exising metadata.

# EDepthMap8::GetPixel

Gets the value of a pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EDepth8 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel.

*y*

Row of the pixel.

# EDepthMap8::GetZValue

Gets Z value of a pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetZValue(
 const OEV_INT32 x,
 const OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel.

*y*

Row of the pixel.

# EDepthMap8::GetHeight

# EDepthMap8::SetHeight

Access depth map Height.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetHeight() const
void SetHeight(OEV_INT32 height)
```

# EDepthMap8::IsVoid

Tests if the EDepthMap8 object has a size of zero.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool IsVoid(
 )
```

## Remarks

Returns **TRUE** if the depthmap size is zero.

# EDepthMap8::Load

Restores the EDepthMap8 stored in the given Open eVision file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 const std::string& path
 )
```

### Parameters

*path*

Full path of the file.

### Remarks

When loading, the depth map is resized if needed.
This function restores the depth map attributes.

# EDepthMap8::LoadImage

Restores the EDepthMap8 image stored in the given image file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void LoadImage(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

### Remarks

When loading, the depth map is resized if need be.
This function does not restore the depth map attributes, only the image associated with the EDepthMap8 is updated.

# EDepthMap8::LoadImageAndMetadata

Loads the image and the metadata from a file in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void LoadImageAndMetadata(
 const std::string& pathImage,
 const std::string& pathMetadata
 )
```

### Parameters

*pathImage*

Full path to the image file.

*pathMetadata*

Full path to the metadata file.

# EDepthMap8::LoadMetadata

Loads the metadata from a file in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void LoadMetadata(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

# EDepthMap8::ModifyMetadata

Changes the value of an existing metadata.
Throws an exception if the metadata does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ModifyMetadata(
 const std::string& Key,
 const std::string& value
 )
```

### Parameters

*Key*

The name of an exising metadata.

*value*

The value for the given metadata.

# EDepthMap8::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EDepthMap8& operator=(
 const EDepthMap8& other
 )
```

### Parameters

*other*

The source EDepthMap8.

# EDepthMap8::GetRowPitch

Returns the buffer row pitch.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetRowPitch() const
```

# EDepthMap8::Save

Saves the EDepthMap8 object to the given Open eVision file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 const std::string& path
 )
```

## Parameters

*path*

The full path to the destination file.

## Remarks

This function saves the EDepthMap8 in a Open eVision file.
This function stores the depth map attributes.

# EDepthMap8::SaveImage

Saves the EDepthMap8 image to the given image file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveImage(
 const std::string& path,
 Euresys::Open_eVision_2_10::EImageFileType type
 )
```

## Parameters

*path*

The full path to the destination file.

*type*

File format, as defined by EImageFileType.
If not specified, the file format is determined from the file extension.

## Remarks

This function saves the image associated to EDepthMap8 in a standard image file and thus does not store depth map attributes.

# EDepthMap8::SaveImageAndMetadata

Saves the image and the metadata to a JSON format file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
void SaveImageAndMetadata(
 const std::string& pathImage,
 const std::string& pathMetadata,
 Euresys::Open_eVision_2_10::EImageFileType type
 )
```

### Parameters

*pathImage*

The full path to the destination image file.

*pathMetadata*

The full path to the destination metadata file.

*type*

File format, as defined by EImageFileType.
If not specified, the file format is determined from the file extension.

# EDepthMap8::SaveJpeg

Saves the EDepthMap8 object to the given image file, in JPEG format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SaveJpeg(
 const std::string& path,
 int quality
 )
```

### Parameters

*path*

The full path of the destination file.

*quality*

JPEG quality, between 0 and 100 (100 is best quality). The default value is **75**.

# EDepthMap8::SaveJpeg2K

Saves the EDepthMap8 object to the given imagefile, in JPEG 2000 format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveJpeg2K(
 const std::string& path,
 int quality
 )
```

## Parameters

*path*

The full path of the destination file.

*quality*

JPEG 2000 quality, between 1 and 512.
The default value is **16**.

# EDepthMap8::SaveMetadata

Saves the metadata to a file in JSON format

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveMetadata(
 const std::string& path
 )
```

## Parameters

*path*

The full path to the destination file.

# EDepthMap8::Serialize

Serializes the object with all its attributes.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or writen to.

# EDepthMap8::SerializeImage

Serializes the image associated to EDepthMap8.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SerializeImage(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or writen to.

# EDepthMap8::SetBufferPtr

Sets the pointer to an externally allocated buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetBufferPtr(
 OEV_INT32 width,
 OEV_INT32 height,
 void* imagePointer,
 OEV_INT32 bitsPerRow
 )
```

### Parameters

*width*

The width of the supplied buffer, in pixels.

*height*

The height of the supplied buffer, in pixels.

*imagePointer*

The pointer (aligned on 4 bytes) to the buffer, which must be large enough to hold the image.

*bitsPerRow*

The total number of bits contained in a row, padding included. Using the value **0** (default) means that this size is computed from the buffer width and the pixel size plus a padding with the smallest possible value that leads to a multiple of 4 bytes (32 bits), which is the minimum padding accepted by EDepthMap8::SetBufferPtr.

# EDepthMap8::SetPixel

Sets the value of a pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetPixel(
 EDepth8 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

Value of the pixel.

*x*

Column of the pixel.

*y*

Row of the pixel.

# EDepthMap8::SetSize

Sets the width and height of a DepthMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetSize(
 OEV_INT32 width,
 OEV_INT32 height
 )
void SetSize(
 const EDepthMap& other
 )
```

### Parameters

*width*

The new requested DepthMap width.

*height*

The new requested DepthMap height.

*other*

The other DepthMap whose dimensions have to be used for the current object.

### Remarks

Open eVision will allocate a new buffer (deallocate the old buffer) if the supplied width and height are different from the existing ones.
If an external buffer has been specified by means of **SetImagePtr**, it will be kept only if the size does not change.
Creating a new buffer and setting its size creates a 4-byte aligned buffer, by default.

# EDepthMap8::GetType

Returns the image type.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
Euresys::Open_eVision_2_10::EImageType GetType() const
```

# EDepthMap8::GetUndefinedValue

Returns the Undefined value. That value is used to mark pixels with no valid depth value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
EDepth8 GetUndefinedValue() const
```

# EDepthMap8::GetWidth

# EDepthMap8::SetWidth

Access depth map Width.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetWidth() const
void SetWidth(OEV_INT32 width)
```

# EDepthMap8::GetZResolution

# EDepthMap8::SetZResolution

Access the Z Resolution (depth units per grey scale value).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetZResolution() const
void SetZResolution(float resolution)
```

# 4.59. EDepthMapToMeshConverter Class

Performs the conversion from a EDepthMap to a EMesh, using the given calibration model.
A Depth Map is a grayscale image acquired by a laser triangulation system.
The calibration model defines how to transform a pixel from the Depth Map to a world space position.
The resulting 3D representation contains a EMesh representing the surface in the world space.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| Convert | Using the ECalibrationModel, the method 'Convert' performs the conversion from a EDepthMap to a EMesh. |
| EDepthMapToMeshConverter | Creates an EDepthMapToMeshConverter object. |
| GetCalibrationModel | Access the ECalibrationModel used for conversion. |
| Load | Loads the converter configuration. The given ESerializer must have been created for reading. |
| operator= | Assignment operator. |
| Save | Saves the converter configuration. The given ESerializer must have been created for writing. |
| SetCalibrationModel | Access the ECalibrationModel used for conversion. |

# EDepthMapToMeshConverter::GetCalibrationModel

# EDepthMapToMeshConverter::SetCalibrationModel

Access the ECalibrationModel used for conversion.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const ECalibrationModel& GetCalibrationModel() const
void SetCalibrationModel(const ECalibrationModel& model)
```

# EDepthMapToMeshConverter::Convert

Using the ECalibrationModel, the method 'Convert' performs the conversion from a EDepthMap to a EMesh.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Convert(
 const EDepthMap8& srcDepthMap,
 EMesh& obj
 )
void Convert(
 const EDepthMap16& srcDepthMap,
 EMesh& obj
 )
```

```
void Convert(
 const EDepthMap8& srcDepthMap,
 ERegion& region,
 EMesh& obj
 )

void Convert(
 const EDepthMap16& srcDepthMap,
 ERegion& region,
 EMesh& obj
 )
```

### Parameters

*srcDepthMap*

The Depth Map to convert.

*obj*

The destination mesh.

*region*

The region of interest.

# EDepthMapToMeshConverter::EDepthMapToMeshConverter

Creates an EDepthMapToMeshConverter object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EDepthMapToMeshConverter(
 )

void EDepthMapToMeshConverter(
 const EDepthMapToMeshConverter& other
 )
```

### Parameters

*other*

Another EDepthMapToMeshConverter.

# EDepthMapToMeshConverter::Load

Loads the converter configuration. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 ESerializer* serializer
 )
```

## Parameters

*serializer*

The serializer.

# EDepthMapToMeshConverter::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EDepthMapToMeshConverter& operator=(
 const EDepthMapToMeshConverter& other
 )
```

## Parameters

*other*

-

# EDepthMapToMeshConverter::Save

Saves the converter configuration. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# 4.60. EDepthMapToPointCloudConverter Class

Performs the conversion from a EDepthMap to a EPointCloud, using the given calibration model.
A Depth Map is a grayscale image acquired by a laser triangulation system.
The calibration model defines how to transform a pixel from the Depth Map to a world space position.
The resulting EPointCloud contains a point per defined pixel of the Depth Map.
Undefined pixels are discarded.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| Convert | Applies the ECalibrationModel to the Depth Map pixels and fill the EPointCloud with world positions. |

| | |
|---|---|
| EDepthMapToPointCloudConverter | Create a EDepthMapToPointCloudConverter. |
| GetCalibrationModel | Access the ECalibrationModel used for conversion. |
| Load | Loads the converter configuration. The given ESerializer must have been created for reading. |
| operator= | Assignment operator. |
| Save | Saves the converter configuration. The given ESerializer must have been created for writing. |
| SetCalibrationModel | Access the ECalibrationModel used for conversion. |

# EDepthMapToPointCloudConverter::GetCalibrationModel

# EDepthMapToPointCloudConverter::SetCalibrationModel

Access the ECalibrationModel used for conversion.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
const ECalibrationModel& GetCalibrationModel() const
```

```
void SetCalibrationModel(const ECalibrationModel& model)
```

# EDepthMapToPointCloudConverter::Convert

Applies the ECalibrationModel to the Depth Map pixels and fill the EPointCloud with world positions.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Convert(
 const EDepthMap8& srcDepthMap,
 EPointCloud& pc
 )

void Convert(
 const EDepthMap16& srcDepthMap,
 EPointCloud& pc
 )

void Convert(
 const EDepthMap32f& srcDepthMap,
 EPointCloud& pc
 )

void Convert(
 const EDepthMap8& srcDepthMap,
 ERegion& region,
 EPointCloud& pc
 )

void Convert(
 const EDepthMap16& srcDepthMap,
 ERegion& region,
 EPointCloud& pc
 )
```

```
void Convert(
 const EDepthMap32f& srcDepthMap,
 ERegion& region,
 EPointCloud& pc
 )
```

### Parameters

*srcDepthMap*

The Depth Map to convert.

*pc*

The destination Point Cloud.

*region*

The region of interest, only pixels inside the given region are converted and added to the Point Cloud.

# EDepthMapToPointCloudConverter::EDepthMapToPointCloudConverter

Create a EDepthMapToPointCloudConverter.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EDepthMapToPointCloudConverter(
 )
void EDepthMapToPointCloudConverter(
 const EDepthMapToPointCloudConverter& other
 )
```

### Parameters

*other*

Another EDepthMapToPointCloudConverter.

# EDepthMapToPointCloudConverter::Load

Loads the converter configuration. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# EDepthMapToPointCloudConverter::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EDepthMapToPointCloudConverter& operator=(
 const EDepthMapToPointCloudConverter& other
 )
```

### Parameters

*other*

Another EDepthMapToPointCloudConverter.

# EDepthMapToPointCloudConverter::Save

Saves the converter configuration. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# 4.61. EEllipseRegion Class

Manages a complete context for an ERegion shaped like an ellipse.

**Base Class:** ERegion

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Drag | Moves the specified handle to a new position and updates all placement parameters of the region. |
| EEllipseRegion | Constructs an EEllipseRegion context. |

| | |
|---|---|
| GetAngle | Angle of the region |
| GetCenter | Center of the region |
| GetHorizontalRadius | Horizontal radius of the region |
| GetVerticalRadius | Vertical radius of the region |
| HitTest | Detects if the cursor is placed over one of the dragging handles. |
| Load | Loads the EEllipseRegion. The given ESerializer must have been created for reading. |
| operator!= | Checks if this EEllipseRegion instance is not strictly equal to another |
| operator= | Assignment operator. |
| operator== | Checks if this EEllipseRegion instance is strictly equal to another |
| Rotate | Creates a new EEllipseRegion by rotating the EEllipseRegion. |
| Save | Saves the EEllipseRegion. The given ESerializer must have been created for writing. |

| | |
|---|---|
| Scale | Creates a new EEllipseRegion by scaling the EEllipseRegion. |
| SetAngle | Angle of the region |
| SetCenter | Center of the region |
| SetHorizontalRadius | Horizontal radius of the region |
| SetVerticalRadius | Vertical radius of the region |
| Translate | Creates a new EEllipseRegion by translating the EEllipseRegion. |

# EEllipseRegion::GetAngle

# EEllipseRegion::SetAngle

Angle of the region

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetAngle() const
void SetAngle(float angle)
```

# EEllipseRegion::GetCenter

# EEllipseRegion::SetCenter

Center of the region

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetCenter() const
void SetCenter(const EPoint& center)
```

# EEllipseRegion::Drag

Moves the specified handle to a new position and updates all placement parameters of the region.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 int x,
 int y,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*x*

x-coordinate of the mouse cursor.

*y*

y-coordinate of the mouse cursor.

*zoomX*

Horizontal zoom factor. By default, true scale is used.

*zoomY*

Vertical zoom factor. By default, true scale is used.

*panX*

Horizontal pan offset. By default, no pan is added.

*panY*

Vertical pan offset. By default, no pan is added.

### Remarks

If zooming and/or panning were used when drawing the region, the same values must be used with EEllipseRegion::HitTest and EEllipseRegion::Drag.

# EEllipseRegion::EEllipseRegion

Constructs an EEllipseRegion context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EEllipseRegion(
 )
void EEllipseRegion(
 float centerX,
 float centerY,
 float radius1,
 float radius2,
 float angle
 )
```

```
void EEllipseRegion(
 const EPoint& center,
 float radius1,
 float radius2,
 float angle
 )

void EEllipseRegion(
 const EPoint& center,
 const EPoint& axisEnd1,
 const EPoint& axisEnd2
 )

void EEllipseRegion(
 const EPoint& pt1,
 const EPoint& pt2,
 const EPoint& pt3,
 const EPoint& pt4,
 const EPoint& pt5
 )

void EEllipseRegion(
 const EEllipseRegion& other
 )
```

### Parameters

*centerX*

The abscissa of the center of the EEllipseRegion.

*centerY*

The ordinate of the center of the EEllipseRegion.

*radius1*

The abscissa radius of the non rotated EEllipseRegion.

*radius2*

The ordinate radius of the non rotated EEllipseRegion.

*angle*

The angle of the rotated EEllipseRegion.

*center*

The center of the EEllipseRegion.

*axisEnd1*

The point corresponding to the end of the abscissa axis of the non rotated EEllipseRegion.

*axisEnd2*

The point corresponding to the end of the ordinate axis of the non rotated EEllipseRegion.

*pt1*

One of the five points defining the EEllipseRegion.

*pt2*

One of the five points defining the EEllipseRegion.

*pt3*

One of the five points defining the EEllipseRegion.

*pt4*

One of the five points defining the EEllipseRegion.

*pt5*

One of the five points defining the EEllipseRegion.

*other*

EEllipseRegion context to copy.

### Remarks

When defining an EEllipseRegion, the two resulting radius values must not be **0** else an EError_Parameter3OutOfRange or EError is thrown.
When defining an EEllipseRegion, the two resulting radius valued must be small enough so that the region fit in memory.
When defining an EEllipseRegion with two axis ends, the two axis ends and the center must not all lie on the same straight line else an EError is thrown.
When defining an EEllipseRegion with five points, no more than two of the points should lie on the same straight line else an EError is thrown.

# EEllipseRegion::HitTest

Detects if the cursor is placed over one of the dragging handles.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::EEditionMode HitTest(
 int x,
 int y,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

### Parameters

*x*

x-coordinate of the mouse cursor.

*y*

y-coordinate of the mouse cursor.

*zoomX*

Horizontal zoom factor. By default, true scale is used.

*zoomY*

Vertical zoom factor. By default, true scale is used.

*panX*

Horizontal pan offset. By default, no pan is added.

*panY*

Vertical pan offset. By default, no pan is added.

### Remarks

Returns a handle identifier, as defined by EEditionMode.
If zooming and/or panning were used when drawing the region, the same values must be used with EEllipseRegion::HitTest and EEllipseRegion::Drag.

# EEllipseRegion::GetHorizontalRadius

# EEllipseRegion::SetHorizontalRadius

Horizontal radius of the region

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetHorizontalRadius() const
void SetHorizontalRadius(float radius)
```

# EEllipseRegion::Load

Loads the EEllipseRegion. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# EEllipseRegion::operator!=

Checks if this EEllipseRegion instance is not strictly equal to another

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
bool operator!=(
 const EEllipseRegion& other
 )
```

### Parameters

*other*

Reference to the other EEllipseRegion instance

# EEllipseRegion::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EEllipseRegion& operator=(
 const EEllipseRegion& other
 )
```

### Parameters

*other*

Reference to the EEllipseRegion used for the assignment

# EEllipseRegion::operator==

Checks if this EEllipseRegion instance is strictly equal to another

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool operator==(
 const EEllipseRegion& other
 )
```

### Parameters

*other*

Reference to the other EEllipseRegion instance

# EEllipseRegion::Rotate

Creates a new EEllipseRegion by rotating the EEllipseRegion.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EEllipseRegion Rotate(
 float angle
 )
```

### Parameters

*angle*

Rotation angle

# EEllipseRegion::Save

Saves the EEllipseRegion. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# EEllipseRegion::Scale

Creates a new EEllipseRegion by scaling the EEllipseRegion.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EEllipseRegion Scale(
 float scale
 )

EEllipseRegion Scale(
 float scaleX,
 float scaleY
 )
```

### Parameters

*scale*

Isotropic scale

*scaleX*

Horizontal scale

*scaleY*

Vertical scale

# EEllipseRegion::Translate

Creates a new EEllipseRegion by translating the EEllipseRegion.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EEllipseRegion Translate(
 float dx,
 float dy
 )
```

### Parameters

*dx*

Horizontal translation in pixel value

*dy*

Vertical translation in pixel value

# EEllipseRegion::GetVerticalRadius

# EEllipseRegion::SetVerticalRadius

Vertical radius of the region

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetVerticalRadius() const

void SetVerticalRadius(float radius)
```

# 4.62. EErrorStatistics Class

-

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| CopyTo | - |
| EErrorStatistics | Creates a EErrorStatistics object. |
| GetMax | Gets/Sets the maximum error (calculated on the valid samples). |
| GetMean | Gets/Sets the mean error (calculated on the valid samples). |
| GetMin | Gets/Sets the minimum error (calculated on the valid samples). |
| GetNumOfErrors | Gets/Sets the number of errors (samples not found) which is the number of samples on which no error could be calculated. |
| GetNumOfValidSamples | Gets/Sets the number of valid samples which is the number of samples on which the error is calculated. |

| | |
|---|---|
| GetStdDev | Gets/Sets the standard deviation error (calculated on the valid samples). |
| Load | |
| | Loads a EErrorStatistics. The given ESerializer must have been created for reading. |
| operator= | Assignment operator. |
| Save | Saves a EErrorStatistics. The given ESerializer must have been created for writing. |
| SetMax | Gets/Sets the maximum error (calculated on the valid samples). |
| SetMean | Gets/Sets the mean error (calculated on the valid samples). |
| SetMin | Gets/Sets the minimum error (calculated on the valid samples). |
| SetNumOfErrors | Gets/Sets the number of errors (samples not found) which is the number of samples on which no error could be calculated. |
| SetNumOfValidSamples | Gets/Sets the number of valid samples which is the number of samples on which the error is calculated. |
| SetStdDev | Gets/Sets the standard deviation error (calculated on the valid samples). |

# EErrorStatistics::CopyTo

-

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void CopyTo(
 EErrorStatistics& other
 )
```

### Parameters

*other*

-

# EErrorStatistics::EErrorStatistics

Creates a EErrorStatistics object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EErrorStatistics(
 )

void EErrorStatistics(
 const EErrorStatistics& other
 )
```

### Parameters

*other*

Reference used for the initialization.

# EErrorStatistics::Load

Loads a EErrorStatistics. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

-

# EErrorStatistics::GetMax

# EErrorStatistics::SetMax

Gets/Sets the maximum error (calculated on the valid samples).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetMax()
void SetMax(float max)
```

# EErrorStatistics::GetMean

# EErrorStatistics::SetMean

Gets/Sets the mean error (calculated on the valid samples).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetMean()
void SetMean(float mean)
```

# EErrorStatistics::GetMin

# EErrorStatistics::SetMin

Gets/Sets the minimum error (calculated on the valid samples).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetMin()
void SetMin(float min)
```

# EErrorStatistics::GetNumOfErrors

# EErrorStatistics::SetNumOfErrors

Gets/Sets the number of errors (samples not found) which is the number of samples on which no error could be calculated.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
int GetNumOfErrors()
void SetNumOfErrors(int numOfErrors)
```

# EErrorStatistics::GetNumOfValidSamples

# EErrorStatistics::SetNumOfValidSamples

Gets/Sets the number of valid samples which is the number of samples on which the error is calculated.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
int GetNumOfValidSamples()
void SetNumOfValidSamples(int numOfValidSamples)
```

# EErrorStatistics::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EErrorStatistics& operator=(
 const EErrorStatistics& other
 )
```

### Parameters

*other*

-

# EErrorStatistics::Save

Saves a EErrorStatistics. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

-

# EErrorStatistics::GetStdDev

# EErrorStatistics::SetStdDev

Gets/Sets the standard deviation error (calculated on the valid samples).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetStdDev()
void SetStdDev(float stdDev)
```

# 4.63. EException Class

Holds the exception information, that is the code and the description of the error that has thrown the exception.

### Remarks

Each time an Open eVision error occurs, an exception is thrown. Exceptions feature an error code and a description. To catch a potentially arising exception, the function call is included in a try-catch block.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EException | - |
| GetError | - |

| | |
|---|---|
| operator= | - |
| SetError | - |
| What | Returns the description of the error that has thrown the exception. |

E

# xception::EException

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EException(
 )
void EException(
 Euresys::Open_eVision_2_10::EError error
 )
void EException(
 const EException& other
 )
void EException(
 const std::string& message
 )
```

## Parameters

*error*

-

*other*

-

*message*

-

# EException::GetError

# EException::SetError

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EError GetError() const
void SetError(Euresys::Open_eVision_2_10::EError error)
```

# EException::operator=

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EException& operator=(
 const EException& other
 )
```

## Parameters

*other*

-

# EException::What

Returns the description of the error that has thrown the exception.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::string What(
 )
```

# 4.64. EExplicitGeometricCalibrationModel Class

EExplicitGeometricCalibrationModel is used to calibrate a depth map from a minimal set of explicit geometric values. All model parameters are given to the EExplicitGeometricCalibrationModel constructor.

**Base Class:** ECalibrationModel

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| EExplicitGeometricCalibrationModel | Constructs a EExplicitGeometricCalibrationModel. EExplicitGeometricCalibrationModel is used to calibrate a depth map from a minimal set of explicit geometric values. |
| GetCameraAngle | Camera view angle. |

| | |
|---|---|
| GetCameraHeight | The height from the camera optical center to the object reference plane (assuming the reference plane is projected in the center line of the image), in mm. |
| GetFocalLength | Camera focal length. |
| GetLaserPlaneAngle | Laser plane angle. |
| GetMotionIncrement | Motion increment (Distance between each line of the depth map). |
| GetRoiBottomLine | The ROI bottom line used in laser line extraction. |
| GetRoiLeftColumn | The ROI left column used in laser line extraction. |
| GetSensorHeight | Camera sensor height. |
| GetSensorWidth | Camera sensor width. |
| GetSensorXResolution | Camera X resolution. |
| GetSensorYResolution | Camera Y resolution. |
| GetType | Returns the type of calibration model, see ECalibrationType. |

| | |
|---|---|
| IsInitialized | Returns true if the model has been correctly initialized. |
| Load | Loads the EExplicitGeometricCalibrationModel. The given ESerializer must have been created for reading. |
| operator= | Assignment operator. |
| operator== | Comparison operator. |
| Save | Saves the EExplicitGeometricCalibrationModel. The given ESerializer must have been created for writing. |

# EExplicitGeometricCalibrationModel::GetCameraAngle

Camera view angle.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetCameraAngle() const
```

# EExplicitGeometricCalibrationModel::GetCameraHeight

The height from the camera optical center to the object reference plane (assuming the reference plane is projected in the center line of the image), in mm.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetCameraHeight() const
```

# EExplicitGeometricCalibrationModel::EExplicitGeometricCalibrationModel

Constructs a EExplicitGeometricCalibrationModel.
EExplicitGeometricCalibrationModel is used to calibrate a depth map from a minimal set of explicit geometric values.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EExplicitGeometricCalibrationModel(
 )

void EExplicitGeometricCalibrationModel(
 float sensorWidth,
 float sensorHeight,
 int sensorXResolution,
 int sensorYResolution,
 int roiLeftColumn,
 int roiBottomLine,
 float focalLength,
 float cameraAngle,
 float cameraHeight,
 float laserPlaneAngle,
 float motionIncrement
 )

void EExplicitGeometricCalibrationModel(
 const EExplicitGeometricCalibrationModel& other
 )
```

### Parameters

*sensorWidth*

The camera sensor width, in mm.

*sensorHeight*

The camera sensor height, in mm.

*sensorXResolution*

The camera X resolution (pixel count in width).

*sensorYResolution*

The camera Y resolution (pixel count in height).

*roiLeftColumn*

The ROI left column used in laser line extraction.
Between the left (0) and the right (width) of the image.

*roiBottomLine*

The ROI bottom line used in laser line extraction.
Between the top (0) and the bottom (height) of the image. That's the depth map values origin.

*focalLength*

The camera optics focal length, in mm.

*cameraAngle*

The camera angle from the vertical axis.
Looking down camera is angle 0 and positive in counter clockwise direction. Valid values are between 0 (vertical orientation) and lower that 90° (horizontal orientation).

*cameraHeight*

The height from the camera optical center to the object reference plane (assuming the reference plane is projected in the center line of the image), in mm.

*laserPlaneAngle*

The laser plane angle from the vertical axis.
A perfect vertical laser orientation is angle 0 and negative in clockwise direction. Valid values for laser angle are between -90° (excluded) and +90° (excluded).

*motionIncrement*

The distance in mm between each line of the depth map.
That's the relative motion of the camera/laser setup to the object position.

*other*

Another EExplicitGeometricCalibrationModel.

# EExplicitGeometricCalibrationModel::GetFocalLength

Camera focal length.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetFocalLength() const
```

# EExplicitGeometricCalibrationModel::IsInitialized

Returns true if the model has been correctly initialized.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool IsInitialized(
 )
```

# EExplicitGeometricCalibrationModel::GetLaserPlaneAngle

Laser plane angle.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetLaserPlaneAngle() const
```

# EExplicitGeometricCalibrationModel::Load

Loads the EExplicitGeometricCalibrationModel. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# EExplicitGeometricCalibrationModel::GetMotionIncrement

Motion increment (Distance between each line of the depth map).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetMotionIncrement() const
```

# EExplicitGeometricCalibrationModel::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EExplicitGeometricCalibrationModel& operator=(
 const EExplicitGeometricCalibrationModel& other
 )
```

### Parameters

*other*

Another EExplicitGeometricCalibrationModel.

# EExplicitGeometricCalibrationModel::operator==

Comparison operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool operator==(
 const EExplicitGeometricCalibrationModel& other
 )
```

### Parameters

*other*

The other EExplicitGeometricCalibrationModel.

# EExplicitGeometricCalibrationModel::GetRoiBottomLine

The ROI bottom line used in laser line extraction.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
int GetRoiBottomLine() const
```

# EExplicitGeometricCalibrationModel::GetRoiLeftColumn

The ROI left column used in laser line extraction.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
int GetRoiLeftColumn() const
```

# EExplicitGeometricCalibrationModel::Save

Saves the EExplicitGeometricCalibrationModel. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Save(
 ESerializer* serializer
 )
```

## Parameters

*serializer*

The serializer.

# EExplicitGeometricCalibrationModel::GetSensorHeight

Camera sensor height.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetSensorHeight() const
```

# EExplicitGeometricCalibrationModel::GetSensorWidth

Camera sensor width.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetSensorWidth() const
```

# EExplicitGeometricCalibrationModel::GetSensorXResolution

Camera X resolution.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
int GetSensorXResolution() const
```

# EExplicitGeometricCalibrationModel::GetSensorYResolution

Camera Y resolution.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
int GetSensorYResolution() const
```

# EExplicitGeometricCalibrationModel::GetType

Returns the type of calibration model, see ECalibrationType.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
Euresys::Open_eVision_2_10::Easy3D::ECalibrationType GetType() const
```

# 4.65. EFeaturesAligner Class

Alignment class, used to calculate the best transformation between matching pairs of points.
The object EFeaturesAligner contains a list of points called the 'Model points list'.
The method EFeaturesAligner::Compute takes a second list of points as argument which is called the 'Measured points list' and produces a E3DTransformMatrix as result.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| Compute | Computes the best transformation matrix for a given Measured points list.<br>This will compute the best transformation for the alignment between the Measured points and the Model points. |
| EFeaturesAligner | Creates a EFeaturesAligner object. |
| GetModelPoints | Sets/Gets the model points list.<br>The model point list is a list of points that is used either as source or as destination of the transformation to calculate.<br>The model points list should at least contain 3 unaligned points. |
| GetPolarityTransform | Sets/Gets the polarity of the transformation from EAlignmentPolarity. |
| Load | Loads the EFeaturesAligner object configuration. The given ESerializer must have been created for reading. |

| | |
|---|---|
| operator= | Assignment operator. |
| Save | Saves the EFeaturesAligner object configuration. The given ESerializer must have been created for writing. |
| SetModelPoints | Sets/Gets the model points list.<br>The model point list is a list of points that is used either as source or as destination of the transformation to calculate.<br>The model points list should at least contain 3 unaligned points. |
| SetPolarityTransform | Sets/Gets the polarity of the transformation from EAlignmentPolarity. |

# eaturesAligner::Compute

Computes the best transformation matrix for a given Measured points list.
This will compute the best transformation for the alignment between the Measured points and the Model points.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DTransformMatrix Compute(
 const std::vector<Euresys::Open_eVision_2_10::Easy3D::E3DPoint>& measuredPoints
 )

E3DTransformMatrix Compute(
 const std::vector<Euresys::Open_eVision_2_10::Easy3D::E3DPoint>& measuredPoints,
 EErrorStatistics& errorStatistics
 )
```

### Parameters

*measuredPoints*

The mesured points list; the measured points list should at least contain 3 unaligned points.

*errorStatistics*

Optional parameter passed by reference. This object will contains the error statistics (deviation between model and aligned points).

# EFeaturesAligner::EFeaturesAligner

Creates a EFeaturesAligner object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EFeaturesAligner(
 )
void EFeaturesAligner(
 const EFeaturesAligner& other
 )
```

### Parameters

*other*

Reference to the object used for the initialization.

# EFeaturesAligner::Load

Loads the EFeaturesAligner object configuration. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# EFeaturesAligner::GetModelPoints

# EFeaturesAligner::SetModelPoints

Sets/Gets the model points list.
The model point list is a list of points that is used either as source or as destination of the transformation to calculate.
The model points list should at least contain 3 unaligned points.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
std::vector<Euresys::Open_eVision_2_10::Easy3D::E3DPoint> GetModelPoints() const

void SetModelPoints(const std::vector<Euresys::Open_eVision_2_10::Easy3D::E3DPoint>&
points)
```

# EFeaturesAligner::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EFeaturesAligner& operator=(
 const EFeaturesAligner& other
 )
```

### Parameters

*other*

-

# EFeaturesAligner::GetPolarityTransform

# EFeaturesAligner::SetPolarityTransform

Sets/Gets the polarity of the transformation from EAlignmentPolarity.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
Euresys::Open_eVision_2_10::Easy3D::EAlignmentPolarity GetPolarityTransform() const

void SetPolarityTransform(Euresys::Open_eVision_2_10::Easy3D::EAlignmentPolarity
polarity)
```

### Remarks

If polarity is EAlignmentPolarity_ModelToMeasured, calculates the best transformation from the Model points list to the Measured points list (default).
If the polarity is EAlignmentPolarity_MeasuredToModel, calculates the best transformation from the Measured points list to the Model points list.

# EFeaturesAligner::Save

Saves the EFeaturesAligner object configuration. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# 4.66. EFilePointerSerializer Class

-

**Base Class:** ESerializer

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Close | Closes the file associated with the ESerializer object. |
| GetWriting | Returns **TRUE** if the ESerializer object has been created for writing and **FALSE** otherwise. |

## EFilePointerSerializer::Close

Closes the file associated with the ESerializer object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Close(
 )
```

## EFilePointerSerializer::GetWriting

Returns **TRUE** if the ESerializer object has been created for writing and **FALSE** otherwise.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetWriting() const
```

# 4.67. EFileSerializer Class

-

**Base Class:** ESerializer
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Close | Closes the file associated with the ESerializer object. |
| GetWriting | Returns **TRUE** if the ESerializer object has been created for writing and **FALSE** otherwise. |

F

# ileSerializer::Close

Closes the file associated with the ESerializer object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Close(
 )
```

# EFileSerializer::GetWriting

Returns **TRUE** if the ESerializer object has been created for writing and **FALSE** otherwise.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetWriting() const
```

# 4.68. EFilters Class

Filtering functions used to remove noise on 3D containers.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| RemoveNoise | In a EDepthMap or EZMap, removes noisy pixels. A square filter window is moved over the depthmap. Within this filter window, the average and/or the standard deviation is/are calculated and a rejection criterium defined by the parameter 'method' determines which pixels will be removed. If the rejection criterium determines that a pixel has to be removed or if there is not enough valid pixels in the window filter, as specified by the parameter 'minValidRatio', the pixel is either |

## ilters::RemoveNoise

In a EDepthMap or EZMap, removes noisy pixels. A square filter window is moved over the depthmap. Within this filter window, the average and/or the standard deviation is/are calculated and a rejection criterium defined by the parameter 'method' determines which pixels will be removed.
If the rejection criterium determines that a pixel has to be removed or if there is not enough valid pixels in the window filter, as specified by the parameter 'minValidRatio', the pixel is either simply removed (marked 'invalid') or replaced by the average value (within the filter window), depening on the value of 'replaceByAvg'.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void RemoveNoise(
 const EDepthMap16& sourceDepthMap,
 EDepthMap16& destinationDepthMap,
 Euresys::Open_eVision_2_10::Easy3D::ENoiseRemovalMethod method,
 OEV_INT16 halfKernelSize,
 float threshold,
 float minValidRatio,
 BOOL replaceByAvg
 )

void RemoveNoise(
 const EDepthMap8& sourceDepthMap,
 EDepthMap8& destinationDepthMap,
 Euresys::Open_eVision_2_10::Easy3D::ENoiseRemovalMethod method,
 OEV_INT16 halfKernelSize,
 float threshold,
 float minValidRatio,
 BOOL replaceByAvg
 )

void RemoveNoise(
 const EZMap16& sourceZMap,
 EZMap16& destinationZMap,
 Euresys::Open_eVision_2_10::Easy3D::ENoiseRemovalMethod method,
 OEV_INT16 halfKernelSize,
 float threshold,
 float minValidRatio,
 BOOL replaceByAvg
 )

void RemoveNoise(
 const EZMap8& sourceZMap,
 EZMap8& destinationZMap,
 Euresys::Open_eVision_2_10::Easy3D::ENoiseRemovalMethod method,
 OEV_INT16 halfKernelSize,
 float threshold,
 float minValidRatio,
 BOOL replaceByAvg
 )
```

## Parameters

*sourceDepthMap*

The source depthmap.

*destinationDepthMap*

The destination depthmap. It should have the same dimensions as the source depthmap.

*method*

Noise removal method of type ENoiseRemovalMethod.

*halfKernelSize*

The half-size of the window filter. The filter window size (= kernel size) is halfKernelSize * 2 + 1, should be positive, smaller than (or equal to) the image size, and may not exceed 256.

*threshold*

The threshold used by the rejection criterium.

*minValidRatio*

Required ratio of valid pixels in the filter window to process the calculation. If not enough, marks the pixel for replacement. Setting this ratio to 0.0 means that only one pixel has to be valid. The default value is 0.25 .

*replaceByAvg*

The marked pixels are removed by default; if this parameter is set to true, replaces the marked pixels by the average value, calculated within the filter window.

*sourceZMap*

The source ZMap.

*destinationZMap*

The destination ZMap. It should have the same dimensions as the source ZMap.

# 4.69. EFloatRange Class

Represents a range of floating point values.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EFloatRange | Creates an EFloatRange object. |
| GetCenter | Center of the range. |
| GetLowerBound | Lower bound of the range. |

| | |
|---|---|
| GetSize | Size of the range. |
| GetUpperBound | Upper bound of the range. |
| IsInRange | Checks if a value is inside the range. |
| operator= | Assignment operator |
| operator== | Comparison operator |
| SetBounds | Sets the bounds of the range. |
| SetFromBaseAndAbsoluteTolerance | Sets the bounds of the range from a base value and an absolute tolerance from this base value. |
| SetFromBaseAndRelativeTolerance | Sets the bounds of the range from a base value and a relative tolerance from this base value. |
| Update | Updates the range. If the value passed is outside of the range bounds, they are modified so they include the value. |

# EFloatRange::GetCenter

Center of the range.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCenter() const
```

# EFloatRange::EFloatRange

Creates an EFloatRange object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EFloatRange(
 )
void EFloatRange(
 float min,
 float max
 )
void EFloatRange(
 const EFloatRange& range
 )
```

## Parameters

*min*

Lower bound of the range.

*max*

Upper bound of the range.

*range*

Range to copy.

# EFloatRange::IsInRange

Checks if a value is inside the range.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool IsInRange(
 float value,
 bool lowerBoundInclusive,
 bool upperBoundInclusive
 )
```

### Parameters

*value*

Value to test.

*lowerBoundInclusive*

Indicates if the lower bound should be considered inside the range (true) or outside (false).

*upperBoundInclusive*

Indicates if the upper bound should be considered inside the range (true) or outside (false).

# EFloatRange::GetLowerBound

Lower bound of the range.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetLowerBound() const
```

# EFloatRange::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EFloatRange& operator=(
 const EFloatRange& other
 )
```

## Parameters

*other*

-

# EFloatRange::operator==

Comparison operator

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool operator==(
 const EFloatRange& other
 )
```

## Parameters

*other*

The other object.

# EFloatRange::SetBounds

Sets the bounds of the range.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetBounds(
 float min,
 float max
 )
```

### Parameters

*min*

Lower bound of the range.

*max*

Upper bound of the range.

# EFloatRange::SetFromBaseAndAbsoluteTolerance

Sets the bounds of the range from a base value and an absolute tolerance from this base value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromBaseAndAbsoluteTolerance(
 float baseValue,
 float tolerance
 )
```

### Parameters

*baseValue*

Base value.

*tolerance*

Absolute tolerance around the base value. Must be positive.

### Remarks

The range will be set with a lower bound of (base - tolerance) and an upper bound of (base + tolerance).

# EFloatRange::SetFromBaseAndRelativeTolerance

Sets the bounds of the range from a base value and a relative tolerance from this base value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromBaseAndRelativeTolerance(
 float baseValue,
 float tolerance
 )
```

### Parameters

*baseValue*

Base value.

*tolerance*

Relative tolerance around the base value. Must be positive.

### Remarks

The range will be set with a lower bound of (base - (base * tolerance)) and an upper bound of (base + (base * tolerance)).

# EFloatRange::GetSize

Size of the range.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetSize() const
```

# EFloatRange::Update

Updates the range. If the value passed is outside of the range bounds, they are modified so they include the value.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Update(
 float value
 )
```

## Parameters

*value*

Value to be included in the range.

# EFloatRange::GetUpperBound

Upper bound of the range.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetUpperBound() const
```

# 4.70. EFoundPattern Class

Represents a single instance of the pattern in the search field, as returned by the EasyFind finding process.

### Remarks

EPatternFinder::Find returns a collection of instances of this class. An EFoundPattern object represents one found instance, with all the needed information about it.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Draw | Draws the found pattern, in image coordinates. |
| DrawWithCurrentPen | Draws the found pattern, in image coordinates. |
| EFoundPattern | Constructs a EFoundPattern object. |
| GetAngle | Angle of the found instance, in the current angle unit. |
| GetCenter | Reference point of the instance. |
| GetDrawBoundingBox | Flag indicating if the EFoundPattern::Draw method must draw the bounding box of the **FoundPattern**. |
| GetDrawCenter | Flag indicating if the EFoundPattern::Draw method must draw the center of the **FoundPattern**. |

| | |
|---|---|
| GetDrawFeaturePoints | Flag indicating if the EFoundPattern::Draw method must draw the feature points of the EFoundPattern object. |
| GetQuadrangle | Returns the corners of the bounding box of the found pattern. |
| GetScale | Scaling factor of the found pattern, in units (not percents). |
| GetScore | Matching score of the found pattern, in units (not percents). |
| operator!= | Inequality operator. |
| operator= | Copies all the data from another EFoundPattern object into the current EFoundPattern object |
| operator== | Equality operator. |
| SetDrawBoundingBox | Flag indicating if the EFoundPattern::Draw method must draw the bounding box of the **FoundPattern**. |
| SetDrawCenter | Flag indicating if the EFoundPattern::Draw method must draw the center of the **FoundPattern**. |
| SetDrawFeaturePoints | Flag indicating if the EFoundPattern::Draw method must draw the feature points of the EFoundPattern object. |

# EFoundPattern::GetAngle

Angle of the found instance, in the current angle unit.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetAngle()
```

## Remarks

Read-only. This returned value is always comprised in the range [- a half turn, + a half turn].

# EFoundPattern::GetCenter

Reference point of the instance.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetCenter()
```

## Remarks

By default, this is its center. If the property EPatternFinder::Pivot has been changed in the EPatternFinder, the point returns the pivot in the instance.

# EFoundPattern::Draw

Draws the found pattern, in image coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Horizontal zooming factor.

*zoomY*

Vertical zooming factor. If set to **0**, the horizontal zooming factor will be used for isotropic zooming.

*panX*

Horizontal panning factor.

*panY*

Vertical panning factor.

*color*

The color in which to draw the overlay.

### Remarks

This method draws different features of the EFoundPattern, according to the value of properties EFoundPattern::DrawFeaturePoints, EFoundPattern::DrawCenter, EFoundPattern::DrawBoundingBox. The **zoomX**, **zoomY**, **panX** and **panY** parameters can be used to scale and/or translate the drawing operations.

# EFoundPattern::GetDrawBoundingBox

# EFoundPattern::SetDrawBoundingBox

Flag indicating if the EFoundPattern::Draw method must draw the bounding box of the **FoundPattern**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool GetDrawBoundingBox()
void SetDrawBoundingBox(bool drawBoundingBox)
```

### Remarks

The default value is **TRUE**.

# EFoundPattern::GetDrawCenter

# EFoundPattern::SetDrawCenter

Flag indicating if the EFoundPattern::Draw method must draw the center of the **FoundPattern**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool GetDrawCenter()
void SetDrawCenter(bool drawCenter)
```

### Remarks

The default value is **TRUE**.

# EFoundPattern::GetDrawFeaturePoints

# EFoundPattern::SetDrawFeaturePoints

Flag indicating if the EFoundPattern::Draw method must draw the feature points of the EFoundPattern object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool GetDrawFeaturePoints()
void SetDrawFeaturePoints(bool drawFeaturePoints)
```

### Remarks

The default value is **FALSE**.

# EFoundPattern::DrawWithCurrentPen

Draws the found pattern, in image coordinates.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawWithCurrentPen(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Horizontal zooming factor.

*zoomY*

Vertical zooming factor. If set to **0**, the horizontal zooming factor will be used for isotropic zooming.

*panX*

Horizontal panning factor.

*panY*

Vertical panning factor.

### Remarks

This method draws different features of the EFoundPattern, according to the value of properties EFoundPattern::DrawFeaturePoints, EFoundPattern::DrawCenter, EFoundPattern::DrawBoundingBox. The **zoomX**, **zoomY**, **panX** and **panY** parameters can be used to scale and/or translate the drawing operations.

# EFoundPattern::EFoundPattern

Constructs a EFoundPattern object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EFoundPattern(
 )
void EFoundPattern(
 const EFoundPattern& other
 )
```

### Parameters

*other*

EFoundPattern object to be copied

# EFoundPattern::operator!=

Inequality operator.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool operator!=(
 const EFoundPattern& other
 )
```

### Parameters

*other*

Instance to compare to.

# EFoundPattern::operator=

Copies all the data from another EFoundPattern object into the current EFoundPattern object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EFoundPattern& operator=(
 const EFoundPattern& other
 )
```

### Parameters

*other*

EFoundPattern object to be copied

# EFoundPattern::operator==

Equality operator.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool operator==(
 const EFoundPattern& other
 )
```

### Parameters

*other*

Instance to compare to.

# EFoundPattern::GetQuadrangle

Returns the corners of the bounding box of the found pattern.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EQuadrangle GetQuadrangle() const
```

# EFoundPattern::GetScale

Scaling factor of the found pattern, in units (not percents).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScale()
```

# EFoundPattern::GetScore

Matching score of the found pattern, in units (not percents).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScore()
```

## Remarks

The matching score range is **[-1.0..1.0]**.

# 4.71. EFrame Class

Represents a geometrical frame of reference as well as the parameters needed to transform from/to local and global coordinates. It contains a point and an angle and serves as a base class for geometrical elements.

**Base Class:** EPoint
**Derived Class(es):** ECircle ELine ERectangle EWedge
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| CopyTo | Copies all the data from the current EFrame object into another EFrame object, and returns it. |
| EFrame | Constructs a EFrame object. |
| GetAngle | Orientation of the frame. |
| GetCenterX | Abscissa of the origin point of the frame. |
| GetCenterY | Ordinate of the origin point of the frame. |
| GetScale | Horizontal sensor resolution, in pixels per unit. |
| GlobalToLocal | Transforms a geometrical element from global to local coordinates (world to local). |

| | |
|---|---|
| LocalToGlobal | Transforms a geometrical element from local to global coordinates (local to world). |
| operator= | Copies all the data from another EFrame object into the current EFrame object. |
| SetAngle | Orientation of the frame. |
| SetScale | Horizontal sensor resolution, in pixels per unit. |

# EFrame::GetAngle

# EFrame::SetAngle

Orientation of the frame.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetAngle() const
void SetAngle(float angle)
```

# EFrame::GetCenterX

Abscissa of the origin point of the frame.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCenterX() const
```

# EFrame::GetCenterY

Ordinate of the origin point of the frame.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCenterY() const
```

# EFrame::CopyTo

Copies all the data from the current EFrame object into another EFrame object, and returns it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EFrame* CopyTo(
 EFrame* other
 )
```

## Parameters

*other*

Pointer to the EFrame object in which the current EFrame object data have to be copied.

### Remarks

In case of a **NULL** pointer, a new EFrame object will be created and returned.

# EFrame::EFrame

Constructs a EFrame object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EFrame(
 )
void EFrame(
 float centerX,
 float centerY,
 float angle,
 float scale
 )
void EFrame(
 const EPoint& center,
 float angle,
 float scale
 )
void EFrame(
 const EFrame& frame
 )
```

### Parameters

*centerX*

Abscissa of the origin point of the frame.

*centerY*

Ordinate of the origin point of the frame.

*angle*

Orientation of the frame.

*scale*

Horizontal sensor resolution.

*center*

Coordinates of the origin point of the frame.

*frame*

Pre-existing EFrame object used by the copy constructor.

# EFrame::GlobalToLocal

Transforms a geometrical element from global to local coordinates (world to local).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GlobalToLocal(
 const EPoint& global
 )

EFrame GlobalToLocal(
 const EFrame& global
 )
```

## Parameters

*global*

The element, expressed in global coordinates.

# EFrame::LocalToGlobal

Transforms a geometrical element from local to global coordinates (local to world).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint LocalToGlobal(
 const EPoint& local
 )

EFrame LocalToGlobal(
 const EFrame& local
 )

ELine LocalToGlobal(
 const ELine& local
 )

ECircle LocalToGlobal(
 const ECircle& local
 )
```

### Parameters

*local*

The element, expressed in local coordinates.

# EFrame::operator=

Copies all the data from another EFrame object into the current EFrame object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EFrame& operator=(
 const EFrame& frame
 )
```

### Parameters

*frame*

EFrame object to be copied.

## EFrame::GetScale

## EFrame::SetScale

Horizontal sensor resolution, in pixels per unit.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScale() const
void SetScale(float scale)
```

# 4.72. EFrameShape Class

Manages a complete context for measuring frame shapes.

### Remarks

This class allows the grouping of several gauges or other frames.

**Base Class:** EShape

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Closest | Find the daughter shape that is the closest to this shape. To retrieve the closest shape, use EShape::ClosestShape. |
| CopyTo | Copy operator. |

| | |
|---|---|
| Drag | Moves a handle to a new position and updates the position parameters of the shape. |
| Draw | Draws a graphical representation of a shape, as defined by EDrawingMode. |
| DrawWithCurrentPen | Draws a graphical representation of a shape, as defined by EDrawingMode. |
| EFrameShape | Construct a EFrameShape object. |
| GetAngle | The angle of the frame. |
| GetCenter | Origin point coordinates of the frame. |
| GetCenterX | Gets the origin point abscissa of the frame. |
| GetCenterY | Gets the origin point ordinate of the frame. |
| GetScale | The scale of the frame. |
| GetSizeX | Frame X-axis length. |
| GetSizeY | Frame Y-axis length. |

| | |
|---|---|
| GetType | Type of the frame. |
| HitTest | Checks whether the cursor is positioned over a handle (**TRUE**) or not (**FALSE**). |
| operator= | Assignement operator. |
| Set | Sets the coordinates of the origin point and the orientation of the frame. |
| SetAngle | The angle of the frame. |
| SetCenter | Origin point coordinates of the frame. |
| SetCenterXY | Sets the origin point coordinates of the frame. |
| SetScale | The scale of the frame. |
| SetSize | Sets the frame size. |

# EFrameShape::GetAngle

# EFrameShape::SetAngle

The angle of the frame.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetAngle() const
void SetAngle(float angle)
```

# EFrameShape::GetCenter

# EFrameShape::SetCenter

Origin point coordinates of the frame.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetCenter() const
void SetCenter(const EPoint& point)
```

# EFrameShape::GetCenterX

Gets the origin point abscissa of the frame.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCenterX() const
```

# EFrameShape::GetCenterY

Gets the origin point ordinate of the frame.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCenterY() const
```

# EFrameShape::Closest

Find the daughter shape that is the closest to this shape. To retrieve the closest shape, use EShape::ClosestShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Closest(
 )
```

# EFrameShape::CopyTo

Copy operator.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EFrameShape* CopyTo(
 EFrameShape* other,
 BOOL recursive
 )
```

## Parameters

*other*

EFrameShape object to copy to.

*recursive*

**TRUE** if the daughter shapes have to be copied as well, **FALSE** otherwise.

# EFrameShape::Drag

Moves a handle to a new position and updates the position parameters of the shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 OEV_INT32 cursorX,
 OEV_INT32 cursorY
 )
```

## Parameters

*cursorX*

Current cursor coordinates.

*cursorY*

Current cursor coordinates.

# EFrameShape::Draw

Draws a graphical representation of a shape, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )

void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )

void Draw(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the shape must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughter shapes are to be displayed also.

*color*

The color in which to draw the overlay.

# EFrameShape::DrawWithCurrentPen

Draws a graphical representation of a shape, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

# EFrameShape::EFrameShape

Construct a EFrameShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EFrameShape(
 )
```

```
void EFrameShape(
 const EFrameShape& frameShape
 )
```

### Parameters

*frameShape*

Pre-existing EFrameShape object used by the copy constructor.

### Remarks

With the default constructor, all parameters are initialized to their respective default value. With the copy constructor, the constructed frame shape measurement context is based on a pre-existing EFrameShape object. By default, the daughter shapes are also copied. Use the EFrameShape::CopyTo method to disable explicitly the daughter shapes copy.

# EFrameShape::HitTest

Checks whether the cursor is positioned over a handle (**TRUE**) or not (**FALSE**).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL HitTest(
 BOOL daughter
 )
```

### Parameters

*daughter*

**TRUE** if the daughters shapes handles have to be considered as well.

# EFrameShape::operator=

Assignement operator.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EFrameShape& operator=(
 const EFrameShape& other
 )
```

### Parameters

*other*

EFrameShape object to copy.

### Remarks

By default, the daughter shapes are also copied. Use the EFrameShape::CopyTo method to disable explicitly the daughter shapes copy.

# EFrameShape::GetScale

# EFrameShape::SetScale

The scale of the frame.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScale() const
void SetScale(float scale)
```

# EFrameShape::Set

Sets the coordinates of the origin point and the orientation of the frame.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Set(
 const EPoint& center,
 float angle,
 float scale
 )
```

### Parameters

*center*

Coordinates of the origin point of the frame. The default value is **(0,0)**.

*angle*

Rotation angle of the frame. The default value is **0**.

*scale*

Horizontal sensor resolution, in pixels per unit

# EFrameShape::SetCenterXY

Sets the origin point coordinates of the frame.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void SetCenterXY(
 float centerX,
 float centerY
 )
```

### Parameters

*centerX*

Abscissa of the origin point of the frame. Default value is **0**.

*centerY*

Ordinate of the origin point of the frame. Default value is **0**.

# EFrameShape::SetSize

Sets the frame size.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetSize(
 float sizeX,
 float sizeY
 )
```

### Parameters

*sizeX*

Frame X-axis length. The default value is **100**.

*sizeY*

Frame Y-axis length. By default, both axes have the same length.

### Remarks

By default, both frame axis value are set to **100**, which means 100 pixels when the field of view is not calibrated and 100 "units" in case of a calibrated field of view.

# EFrameShape::GetSizeX

Frame X-axis length.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSizeX()
```

## Remarks

By default, both frame axis values are set to **100**, which means 100 pixels when the field of view is not calibrated, and 100 physical units in case of a calibrated field of view.

# EFrameShape::GetSizeY

Frame Y-axis length.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSizeY()
```

## Remarks

By default, both frame axis values are set to **100**, which means 100 pixels when the field of view is not calibrated, and 100 physical units in case of a calibrated field of view.

# EFrameShape::GetType

Type of the frame.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EShapeType GetType()
```

# 4.73. EGrabberDepthMap16 Class

The EGrabberDepthMap16 class is used to wrap an EGrabber buffer whose pixel type is "Coord3D_C16". It can be used in Open eVision processing as an EDepthMap16 object.

**Base Class:** EDepthMap16

**Namespace:** Euresys::Open_eVision_2_10::EGrabberBridge

## Methods

| | |
|---|---|
| EGrabberDepthMap16 | Constructs an EGrabberDepthMap16. |

# EGrabberDepthMap16::EGrabberDepthMap16

Constructs an EGrabberDepthMap16.

**Namespace:** Euresys::Open_eVision_2_10::EGrabberBridge

```
[C++]
```

```
void EGrabberDepthMap16(
 const BufferInfo& infos
 )

void EGrabberDepthMap16(
 FormatConvert& converter,
 const BufferInfo& infos
 )

void EGrabberDepthMap16(
 const EGrabberDepthMap16&
 )
```

### Parameters

*infos*

Information pertaining to an EGrabber buffer

*converter*

EGrabber format converter

-

### Remarks

The FormatConverter class allows automatic pixel format conversions. If no format converter is provided, the pixel format of the buffer must be "Coord3D_C16"

# 4.74. EGrabberDepthMap8 Class

The EGrabberDepthMap8 class is used to wrap an EGrabber buffer whose pixel type is "Coord3D_C8". It can be used in Open eVision processing as an EDepthMap8 object.

**Base Class:** EDepthMap8

**Namespace:** Euresys::Open_eVision_2_10::EGrabberBridge

## Methods

| | |
|---|---|
| EGrabberDepthMap8 | Constructs an EGrabberDepthMap8. |

# EGrabberDepthMap8::EGrabberDepthMap8

Constructs an EGrabberDepthMap8.

**Namespace:** Euresys::Open_eVision_2_10::EGrabberBridge

```
[C++]
void EGrabberDepthMap8(
 const BufferInfo& infos
 )
void EGrabberDepthMap8(
 FormatConvert& converter,
 const BufferInfo& infos
 )
void EGrabberDepthMap8(
 const EGrabberDepthMap8&
 )
```

## Parameters

*infos*

Information pertaining to an EGrabber buffer

*converter*

EGrabber format converter

-

## Remarks

The FormatConverter class allows automatic pixel format conversions. If no format converter is provided, the pixel format of the buffer must be "Coord3D_C8"

# 4.75. EGrabberImageBW16 Class

The EGrabberImageBW16 class is used to wrap an EGrabber buffer whose pixel type is "Mono16". It can be used in Open eVision processing as an EImageBW16 object.

**Base Class:** EImageBW16

**Namespace:** Euresys::Open_eVision_2_10::EGrabberBridge

## Methods

| | |
|---|---|
| EGrabberImageBW16 | Constructs an EGrabberImageBW16. |

## EGrabberImageBW16::EGrabberImageBW16

Constructs an EGrabberImageBW16.

**Namespace:** Euresys::Open_eVision_2_10::EGrabberBridge

```
[C++]
void EGrabberImageBW16(
 const BufferInfo& infos
 )

void EGrabberImageBW16(
 FormatConvert& converter,
 const BufferInfo& infos
 )

void EGrabberImageBW16(
 const EGrabberImageBW16&
 )
```

### Parameters

*infos*

Information pertaining to an EGrabber buffer

*converter*

EGrabber format converter

-

### Remarks

The FormatConverter class allows automatic pixel format conversions. If no format converter is provided, the pixel format of the buffer must be "Mono16"

# 4.76. EGrabberImageBW8 Class

The EGrabberImageBW8 class is used to wrap an EGrabber buffer whose pixel type is "Mono8". It can be used in Open eVision processing as an EImageBW8 object.

**Base Class:** EImageBW8

**Namespace:** Euresys::Open_eVision_2_10::EGrabberBridge

## Methods

| | |
|---|---|
| EGrabberImageBW8 | Constructs an EGrabberImageBW8. |

# EGrabberImageBW8::EGrabberImageBW8

Constructs an EGrabberImageBW8.

**Namespace:** Euresys::Open_eVision_2_10::EGrabberBridge

```
[C++]
void EGrabberImageBW8(
 const BufferInfo& infos
 )
void EGrabberImageBW8(
 FormatConvert& converter,
 const BufferInfo& infos
 )
void EGrabberImageBW8(
 const EGrabberImageBW8&
 )
```

### Parameters

*infos*

Information pertaining to an EGrabber buffer

*converter*

EGrabber format converter

-

### Remarks

The FormatConverter class allows automatic pixel format conversions. If no format converter is provided, the pixel format of the buffer must be "Mono8"

# 4.77. EGrabberImageC24 Class

The EGrabberImageC24 class is used to wrap an EGrabber buffer whose pixel type is "BGR8". It can be used in Open eVision processing as an EImageC24 object.

**Base Class:** EImageC24

**Namespace:** Euresys::Open_eVision_2_10::EGrabberBridge

## Methods

| | |
|---|---|
| EGrabberImageC24 | Constructs an EGrabberImageC24. |

# EGrabberImageC24::EGrabberImageC24

Constructs an EGrabberImageC24.

**Namespace:** Euresys::Open_eVision_2_10::EGrabberBridge

```
[C++]
void EGrabberImageC24(
 const BufferInfo& infos
 )
void EGrabberImageC24(
 FormatConvert& converter,
 const BufferInfo& infos
 )
void EGrabberImageC24(
 const EGrabberImageC24&
 )
```

### Parameters

*infos*

Information pertaining to an EGrabber buffer

*converter*

EGrabber format converter

-

### Remarks

The FormatConverter class allows automatic pixel format conversions. If no format converter is provided, the pixel format of the buffer must be "BGR8"

# 4.78. EGrayscaleDoubleThresholdSegmenter Class

Segments an image using a double threshold on a grayscale image.

### Remarks

This segmenter is applicable to EROIBW8 and EROIBW16 grayscale images. It produces coded images with three layers: The Black layer (usually, with index 0) contains the unmasked pixels having a gray value strictly below the low threshold value; the White layer (usually, with index 2) contains the unmasked pixels having a gray value above or equal to the high threshold value; and the Neutral layer (usually, with index 1) contains the remaining unmasked pixels.

**Base Class:** EThreeLayersImageSegmenter

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

## Methods

| | |
|---|---|
| GetHighThreshold | Value of the high threshold. |
| GetLowThreshold | Value of the low threshold. |
| SetHighThreshold | Value of the high threshold. |
| SetLowThreshold | Value of the low threshold. |

# EGrayscaleDoubleThresholdSegmenter::GetHighThreshold

# EGrayscaleDoubleThresholdSegmenter::SetHighThreshold

Value of the high threshold.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

[C++]

```
OEV_UINT32 GetHighThreshold() const
void SetHighThreshold(OEV_UINT32 threshold)
```

## EGrayscaleDoubleThresholdSegmenter::GetLowThreshold

## EGrayscaleDoubleThresholdSegmenter::SetLowThreshold

Value of the low threshold.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

[C++]

```
OEV_UINT32 GetLowThreshold() const
void SetLowThreshold(OEV_UINT32 threshold)
```

# 4.79. EGrayscaleSingleThresholdSegmenter Class

Segments an image using a single threshold on a grayscale image.

### Remarks

This segmenter is applicable to EROIBW8 and EROIBW16 grayscale images. It produces coded images with two layers: the black layer (usually, with index 0) contains the unmasked pixels having a gray value strictly below the threshold value; and the white layer (usually, with index 1) contains the remaining unmasked pixels, i.e. unmasked pixels having a gray value greater or equal to the threshold value. The default thresholding method is minimum residue. If another method is required, the EGrayscaleSingleThresholdSegmenter::Mode property must be set prior to encoding.

**Base Class:** ETwoLayersImageSegmenter

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

## Methods

| | |
|---|---|
| GetAbsoluteThreshold | Value of the threshold to be used in the case of absolute thresholding. |
| GetLastThreshold | Retrieves the actual threshold value that was used for the last image that got encoded by the segmenter. |
| GetMode | Threshold selection mode. |
| GetRelativeThreshold | Fraction of the image pixels that belongs to the black layer in the case of relative thresholding. |
| IsFirstApplication | Checks whether the segmenter has already been used to segment an image. |
| SetAbsoluteThreshold | Value of the threshold to be used in the case of absolute thresholding. |
| SetMode | Threshold selection mode. |
| SetRelativeThreshold | Fraction of the image pixels that belongs to the black layer in the case of relative thresholding. |

# EGrayscaleSingleThresholdSegmenter::GetAbsoluteThreshold

# EGrayscaleSingleThresholdSegmenter::SetAbsoluteThreshold

Value of the threshold to be used in the case of absolute thresholding.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
OEV_UINT32 GetAbsoluteThreshold() const
void SetAbsoluteThreshold(OEV_UINT32 threshold)
```

# EGrayscaleSingleThresholdSegmenter::IsFirstApplication

Checks whether the segmenter has already been used to segment an image.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
bool IsFirstApplication(
 )
```

# EGrayscaleSingleThresholdSegmenter::GetLastThreshold

Retrieves the actual threshold value that was used for the last image that got encoded by the segmenter.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
OEV_UINT32 GetLastThreshold() const
```

### Remarks

A call to this method will result in an exception if it is the first time the segmenter is applied. To check whether the segmenter has already been applied, call the EGrayscaleSingleThresholdSegmenter::IsFirstApplication method.

# EGrayscaleSingleThresholdSegmenter::GetMode

# EGrayscaleSingleThresholdSegmenter::SetMode

Threshold selection mode.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
Euresys::Open_eVision_2_10::EGrayscaleSingleThreshold GetMode() const
void SetMode(Euresys::Open_eVision_2_10::EGrayscaleSingleThreshold mode)
```

# EGrayscaleSingleThresholdSegmenter::GetRelativeThreshold

# EGrayscaleSingleThresholdSegmenter::SetRelativeThreshold

Fraction of the image pixels that belongs to the black layer in the case of relative thresholding.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
float GetRelativeThreshold() const
void SetRelativeThreshold(float threshold)
```

# 4.80. EHarrisCornerDetector Class

Manages a complete context for the Harris corner detector.

### Remarks

This implementation of the Harris corner detector operates exclusively on a grayscale BW8 images.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Apply | Apply the Harris corner detector on an image/ROI. |
| EHarrisCornerDetector | Constructs a EHarrisCornerDetector object initialized to its default values. |
| GetDerivationScale | Sets the derivation scale. |
| GetGradientNormalizationEnabled | Sets whether the gradient is normalized before the computation of the cornerness measure. |
| GetIntegrationScale | The integration scale. |

| | |
|---|---|
| GetSubpixelPrecisionEnabled | Sets whether the sub-pixel interpolation is enabled. |
| GetThreshold | Threshold on the cornerness measure for a pixel to be considered as a corner. |
| GetThresholdingMode | Thresholding mode for the cornerness measure. |
| SetDerivationScale | Sets the derivation scale. |
| SetGradientNormalizationEnabled | Sets whether the gradient is normalized before the computation of the cornerness measure. |
| SetIntegrationScale | The integration scale. |
| SetSubpixelPrecisionEnabled | Sets whether the sub-pixel interpolation is enabled. |
| SetThreshold | Threshold on the cornerness measure for a pixel to be considered as a corner. |
| SetThresholdingMode | Thresholding mode for the cornerness measure. |

E

H

# arrisCornerDetector::Apply

Apply the Harris corner detector on an image/ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Apply(
 const EROIBW8& source,
 EHarrisInterestPoints& interestPoints
 )
```

### Parameters

*source*

The source image/ROI.

*interestPoints*

The container in which to store the interest points.

# EHarrisCornerDetector::GetDerivationScale

# EHarrisCornerDetector::SetDerivationScale

Sets the derivation scale.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetDerivationScale() const
void SetDerivationScale(float derivationScale)
```

### Remarks

The derivation scale is the standard deviation of the Gaussian Filter used for the noise reduction during the computation of the gradient. Whenever the integration scale is set through EHarrisCornerDetector::IntegrationScale, the derivation scale is reset to its default value, **0.7 * integrationScale**. This is a recommended value, as suggested by the literature.

# EHarrisCornerDetector::EHarrisCornerDetector

Constructs a EHarrisCornerDetector object initialized to its default values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EHarrisCornerDetector(
 const EHarrisCornerDetector& other
 )
void EHarrisCornerDetector(
 )
```

### Parameters

*other*

-

# EHarrisCornerDetector::GetGradientNormalizationEnabled

# EHarrisCornerDetector::SetGradientNormalizationEnabled

Sets whether the gradient is normalized before the computation of the cornerness measure.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool GetGradientNormalizationEnabled() const
void SetGradientNormalizationEnabled(bool isEnabled)
```

### Remarks

If this flag is enabled, the values of the X-gradient and of the Y-gradient are first divided by their maximum absolute value (in the internal computations). This results in a cornerness measure that is roughly distributed around the value 1. If this flag is disabled, the cornerness measure will be much greater.

# EHarrisCornerDetector::GetIntegrationScale

# EHarrisCornerDetector::SetIntegrationScale

The integration scale.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetIntegrationScale() const
void SetIntegrationScale(float integrationScale)
```

### Remarks

The *integration scale* is the standard deviation of the Gaussian filter that is used for scale analysis.

# EHarrisCornerDetector::GetSubpixelPrecisionEnabled

# EHarrisCornerDetector::SetSubpixelPrecisionEnabled

Sets whether the sub-pixel interpolation is enabled.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool GetSubpixelPrecisionEnabled() const
void SetSubpixelPrecisionEnabled(bool subpixelEnabled)
```

### Remarks

When this flag is enabled, a sub-pixel interpolation is carried on so as to improve the accuracy of the location of the corners, to the expense of a loss of speed.

# EHarrisCornerDetector::GetThreshold

# EHarrisCornerDetector::SetThreshold

Threshold on the cornerness measure for a pixel to be considered as a corner.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetThreshold() const
void SetThreshold(float threshold)
```

### Remarks

If the threshold mode is set to EHarrisThresholdingMode_Absolute, the threshold value is interpreted as an absolute threshold on the cornerness. In this case, the threshold must be a strictly positive real value.

If the threshold mode is set to EHarrisThresholdingMode_Relative, the threshold is expressed as a fraction ranging from 0 to 1 of the maximum value of the cornerness of the source image.

# EHarrisCornerDetector::GetThresholdingMode

# EHarrisCornerDetector::SetThresholdingMode

Thresholding mode for the cornerness measure.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EHarrisThresholdingMode GetThresholdingMode() const
void SetThresholdingMode(Euresys::Open_eVision_2_10::EHarrisThresholdingMode mode)
```

# 4.81. EHarrisInterestPoints Class

Container class for the results of the Harris corner detector.

### Remarks

The EHarrisCornerDetector class stores its results in this container.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Draw | Draws the location of the corner points. |
| DrawCorner | Draws the location of a specific corner point. |
| DrawCornerWithCurrentPen | Draws the location of a specific corner point. |

| | |
|---|---|
| DrawWithCurrentPen | Draws the location of the corner points. |
| EHarrisInterestPoints | Constructs a container for the results of a Harris corner detector. |
| GetCornerness | Returns the cornerness measure of a corner point. |
| GetGradientMagnitude | Returns the magnitude of the gradient at a corner point. |
| GetGradientOrientation | Returns the orientation of the gradient at a corner point. |
| GetGradientX | Returns the gradient along the X-axis of a corner point. |
| GetGradientY | Returns the gradient along the Y-axis of a corner point. |
| GetPoint | Returns the coordinates of a corner point. |
| GetPointCount | The number of corner points in the container. |
| GetX | Returns the abscissa of a corner point. |
| GetY | Returns the ordinate of a corner point. |

# EHarrisInterestPoints::Draw

Draws the location of the corner points.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )

void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )

void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Graphic context on which to draw.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*originX*

Horizontal panning factor. By default, no panning occurs.

*originY*

Vertical panning factor. By default, no panning occurs.

*color*

The color in which to draw the overlay.

### Remarks

Drawing is done in the device context associated to the desired window.

# EHarrisInterestPoints::DrawCorner

Draws the location of a specific corner point.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawCorner(
 HDC graphicContext,
 int index,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
void DrawCorner(
 HDC graphicContext,
 const ERGBColor& color,
 int index,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

```
void DrawCorner(
 EDrawAdapter* graphicContext,
 int index,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Graphic context on which to draw.

*index*

Corner index

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*originX*

Horizontal panning factor. By default, no panning occurs.

*originY*

Vertical panning factor. By default, no panning occurs.

*color*

The color in which to draw the overlay.

### Remarks

Drawing is done in the device context associated to the desired window.

# EHarrisInterestPoints::DrawCornerWithCurrentPen

Draws the location of a specific corner point.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawCornerWithCurrentPen(
 HDC graphicContext,
 int index,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Graphic context on which to draw.

*index*

Corner index

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*originX*

Horizontal panning factor. By default, no panning occurs.

*originY*

Vertical panning factor. By default, no panning occurs.

### Remarks

Drawing is done in the device context associated to the desired window.

# EHarrisInterestPoints::DrawWithCurrentPen

Draws the location of the corner points.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Graphic context on which to draw.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*originX*

Horizontal panning factor. By default, no panning occurs.

*originY*

Vertical panning factor. By default, no panning occurs.

### Remarks

Drawing is done in the device context associated to the desired window.

# EHarrisInterestPoints::EHarrisInterestPoints

Constructs a container for the results of a Harris corner detector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void EHarrisInterestPoints(
 const EHarrisInterestPoints& other
 )

void EHarrisInterestPoints(
 )
```

### Parameters

*other*

-

# EHarrisInterestPoints::GetCornerness

Returns the cornerness measure of a corner point.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCornerness(
 OEV_UINT32 index
 )
```

### Parameters

*index*

The index of the corner point.

# EHarrisInterestPoints::GetGradientMagnitude

Returns the magnitude of the gradient at a corner point.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetGradientMagnitude(
 OEV_UINT32 index
 )
```

### Parameters

*index*

The index of the corner point.

# EHarrisInterestPoints::GetGradientOrientation

Returns the orientation of the gradient at a corner point.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetGradientOrientation(
 OEV_UINT32 index
 )
```

### Parameters

*index*

The index of the corner point.

# EHarrisInterestPoints::GetGradientX

Returns the gradient along the X-axis of a corner point.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetGradientX(
 OEV_UINT32 index
 )
```

### Parameters

*index*

The index of the corner point.

# EHarrisInterestPoints::GetGradientY

Returns the gradient along the Y-axis of a corner point.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetGradientY(
 OEV_UINT32 index
 )
```

### Parameters

*index*

The index of the corner point.

# EHarrisInterestPoints::GetPoint

Returns the coordinates of a corner point.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetPoint(
 OEV_UINT32 index
 )
```

### Parameters

*index*

The index of the corner point.

# EHarrisInterestPoints::GetX

Returns the abscissa of a corner point.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetX(
 OEV_UINT32 index
 )
```

### Parameters

*index*

The index of the corner point.

# EHarrisInterestPoints::GetY

Returns the ordinate of a corner point.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetY(
 OEV_UINT32 index
 )
```

### Parameters

*index*

The index of the corner point.

## EHarrisInterestPoints::GetPointCount

The number of corner points in the container.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetPointCount() const
```

# 4.82. EHDRColorFuser Class

A EHDRColorFuser instance is a tool that flexibly fuses color images using HDR principles.

**Namespace:** Euresys::Open_eVision_2_10

### Methods

EHDRColorFuser

| | |
|---|---|
| | Constructors for EHDRColorFuser objects. The image used must be the lightest (slowest shutter speed). |
| Fuse | Fuses a dark image with the light image passed during object construction. |
| GetFusedImage | Retrieves the resulting image. Returns false if part of the potential image dynamic is lost due to the chosen output image type. |
| operator= | Assignment operator |

H

# DRColorFuser::EHDRColorFuser

Constructors for EHDRColorFuser objects. The image used must be the lightest (slowest shutter speed).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EHDRColorFuser(
 const EROIC24* lightSrc
 )
void EHDRColorFuser(
 const EROIC48* lightSrc
 )
void EHDRColorFuser(
 const EHDRColorFuser& other
 )
```

### Parameters

*lightSrc*

-

*other*

-

# EHDRColorFuser::Fuse

Fuses a dark image with the light image passed during object construction.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Fuse(
 const EROIC24* darkSrc,
 int shutterSpeedFactor
 )
void Fuse(
 const EROIC48* darkSrc,
 int shutterSpeedFactor
 )
```

### Parameters

*darkSrc*

Dark input image (higher shutter speed).

*shutterSpeedFactor*

Shutter speed factor between light and dark image.

# EHDRColorFuser::GetFusedImage

Retrieves the resulting image. Returns false if part of the potential image dynamic is lost due to the chosen output image type.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool GetFusedImage(
 EROIC24* dst
 )
bool GetFusedImage(
 EROIC48* dst
 )
```

### Parameters

*dst*

Output image.

## EHDRColorFuser::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EHDRColorFuser& operator=(
 const EHDRColorFuser& other
 )
```

### Parameters

*other*

-

# 4.83. EHDRFuser Class

A EHDRFuser instance is a tool that flexibly fuses grayscale images using HDR principles.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EHDRFuser | Constructors for EHDRFuser objects. The image used must be the lightest (slowest shutter speed). |
| Fuse | Fuses a dark image with the light image passed during object construction. |
| GetFusedImage | Retrieves the resulting image. Returns false if part of the potential image dynamic is lost due to the chosen output image type. |
| operator= | Assignment operator |

E

H

# DRFuser::EHDRFuser

Constructors for EHDRFuser objects. The image used must be the lightest (slowest shutter speed).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EHDRFuser(
 const EROIBW8* lightSrc
 )

void EHDRFuser(
 const EROIBW16* lightSrc
 )
```

```
void EHDRFuser(
 const EHDRFuser& other
 )
```

### Parameters

*lightSrc*

-

*other*

-

# EHDRFuser::Fuse

Fuses a dark image with the light image passed during object construction.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Fuse(
 const EROIBW8* darkSrc,
 int shutterSpeedFactor
 )

void Fuse(
 const EROIBW16* darkSrc,
 int shutterSpeedFactor
 )
```

### Parameters

*darkSrc*

Dark input image (higher shutter speed).

*shutterSpeedFactor*

Shutter speed factor between light and dark image.

# EHDRFuser::GetFusedImage

Retrieves the resulting image. Returns false if part of the potential image dynamic is lost due to the chosen output image type.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
bool GetFusedImage(
 EROIBW8* dst
 )
bool GetFusedImage(
 EROIBW16* dst
 )
bool GetFusedImage(
 EROIBW32* dst
 )
```

### Parameters

*dst*

Output image.

# EHDRFuser::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EHDRFuser& operator=(
 const EHDRFuser& other
 )
```

### Parameters

*other*

-

# 4.84. EHitAndMissKernel Class

Class that defines a kernel for the morphological hit-and-miss operations.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EHitAndMissKernel | Constructs a EHitAndMissKernel object. |
| GetEndX | Returns the abscissa of the rightmost element of the kernel. |
| GetEndY | Returns the abscissa of the bottommost element of the kernel. |
| GetStartX | Returns the abscissa of the leftmost element of the kernel. |
| GetStartY | Returns the ordinate of the toptmost element of the kernel. |

| | |
|---|---|
| GetValue | Returns the value of an element of the kernel at a given coordinate. |
| SetSize | Modify the size of the kernel. |
| SetValue | Sets the value of an element of the kernel at a given coordinate. |

E

H

# itAndMissKernel::EHitAndMissKernel

Constructs a EHitAndMissKernel object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EHitAndMissKernel(
 const EHitAndMissKernel& other
 )

void EHitAndMissKernel(
 int startX,
 int startY,
 int endX,
 int endY
 )

void EHitAndMissKernel(
 OEV_UINT32 halfSizeX,
 OEV_UINT32 halfSizeY
 )

void EHitAndMissKernel(
 )
```

### Parameters

*other*

-

*startX*

The abscissa of the top leftmost element of the kernel. This value must be less than or equal to zero.

*startY*

The ordinate of the top leftmost element of the kernel. This value must be less than or equal to zero.

*endX*

The abscissa of the bottom rightmost element of the kernel. This value must be greater than or equal to zero.

*endY*

The ordinate of the bottom rightmost element of the kernel. This value must be greater than or equal to zero.

*halfSizeX*

The half of the kernel width minus 1. This value must be greater than zero.

*halfSizeY*

The half of the kernel height minus 1. This value must be greater than zero.

### Remarks

The constructor without argument creates a centered kernel of size 3x3.

All the elements of the kernel are initialized with EHitAndMissValue_DontCare values.

If the object is constructed by specifying the halves of the kernel dimensions, the width (resp. the height) of the kernel is given by "2 * halfSizeX + 1" (resp. "2 * halfSizeY + 1"). Otherwise, the width (resp. the height) of the kernel is given by "endX - startX + 1" (resp. "endY - startY + 1").

# EHitAndMissKernel::GetEndX

Returns the abscissa of the rightmost element of the kernel.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
int GetEndX() const
```

# EHitAndMissKernel::GetEndY

Returns the abscissa of the bottommost element of the kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetEndY() const
```

# EHitAndMissKernel::GetValue

Returns the value of an element of the kernel at a given coordinate.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EHitAndMissValue GetValue(
 int x,
 int y
 )
```

## Parameters

*x*

The abscissa of the element.

*y*

The ordinate of the element.

# EHitAndMissKernel::SetSize

Modify the size of the kernel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetSize(
 int startX,
 int startY,
 int endX,
 int endY
 )
void SetSize(
 OEV_UINT32 halfSizeX,
 OEV_UINT32 halfSizeY
 )
```

## Parameters

*startX*

The abscissa of the top leftmost element of the kernel. This value must be less than or equal to zero.

*startY*

The ordinate of the top leftmost element of the kernel. This value must be less than or equal to zero.

*endX*

The abscissa of the bottom rightmost element of the kernel. This value must be greater than or equal to zero.

*endY*

The ordinate of the bottom rightmost element of the kernel. This value must be greater than or equal to zero.

*halfSizeX*

The half of the kernel width minus 1. This value must be greater than zero.

*halfSizeY*

The half of the kernel height minus 1. This value must be greater than zero.

### Remarks

All the elements of the kernel are initialized with EHitAndMissValue_DontCare values.

If the object is constructed by specifying the halves of the kernel dimensions, the width (resp. the height) of the kernel is given by "2 * halfSizeX + 1" (resp. "2 * halfSizeY + 1"). Otherwise, the width (resp. the height) of the kernel is given by "endX - startX + 1" (resp. "endY - startY + 1").

# EHitAndMissKernel::SetValue

Sets the value of an element of the kernel at a given coordinate.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetValue(
 int x,
 int y,
 Euresys::Open_eVision_2_10::EHitAndMissValue value
 )
```

### Parameters

*x*

The abscissa of the element.

*y*

The ordinate of the element.

*value*

The value of the element.

# EHitAndMissKernel::GetStartX

Returns the abscissa of the leftmost element of the kernel.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
int GetStartX() const
```

## EHitAndMissKernel::GetStartY

Returns the ordinate of the toptmost element of the kernel.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
int GetStartY() const
```

# 4.85. EHole Class

This class represents a hole inside an object (blob) of an encoded image.

### Remarks

This class inherits from the ECodedElement class and provides an additional method to retrieve the parent object of a particular hole.

**Base Class:** ECodedElement

**Namespace:** Euresys::Open_eVision_2_10

### Methods

| | |
|---|---|
| GetParentObjectIndex | Returns the index of the parent object of the hole. |

# EHole::GetParentObjectIndex

Returns the index of the parent object of the hole.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetParentObjectIndex() const
```

# 4.86. EImageBW1 Class

The EImageBW1 class is used to represent rectangular regions of interest inside EBW1 black and white images. See ROIs.

**Base Class:** EROIBW1

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EImageBW1 | Constructs a EImageBW1 image. |
| GetBitIndex | - |
| operator= | Copies a EImageBW1 image. |

# EImageBW1::EImageBW1

Constructs a EImageBW1 image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EImageBW1(
 )
void EImageBW1(
 OEV_INT32 width,
 OEV_INT32 height
 )
void EImageBW1(
 const EImageBW1& other
 )
```

### Parameters

*width*

The width, in pixels.

*height*

The height, in pixels.

*other*

Another EImageBW1 object.

### Remarks

The constructor with no arguments creates a zero-sized image. You can modify the image size by calling EBaseROI::SetSize method. The sizing constructor constructs an image of the given size. See EBaseROI::SetSize for informations about the sizing restrictions. The copy constructor copies all the supplied image properties (content, attributes, sub-ROIs...) into the current object.

# EImageBW1::GetBitIndex

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT64 GetBitIndex(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

-

*y*

-

# EImageBW1::operator=

Copies a EImageBW1 image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageBW1& operator=(
 const EImageBW1& other
 )
```

### Parameters

*other*

Another EImageBW1 object.

### Remarks

This method copies all the supplied image properties (content, attributes, sub-ROIs...) into the current object.

# 4.87. EImageBW16 Class

The EImageBW16 class is used to represent rectangular regions of interest inside EBW16 gray-level images. See ROIs.

**Base Class:** EROIBW16
**Derived Class(es):** EGrabberImageBW16
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EImageBW16 | Constructs a EImageBW16 image. |
| operator= | Copies a EImageBW16 image. |

# mageBW16::EImageBW16

Constructs a EImageBW16 image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EImageBW16(
 )

void EImageBW16(
 OEV_INT32 width,
 OEV_INT32 height
 )
```

```
void EImageBW16(
 const EImageBW16& other
 )
```

### Parameters

*width*

The width, in pixels.

*height*

The height, in pixels.

*other*

Another EImageBW16 object.

### Remarks

The constructor with no arguments creates a zero-sized image. You can modify the image size by calling EBaseROI::SetSize method. The sizing constructor constructs an image of the given size. See EBaseROI::SetSize for informations about the sizing restrictions. The copy constructor copies all the supplied image properties (content, attributes, sub-ROIs...) into the current object.

# EImageBW16::operator=

Copies a EImageBW16 image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageBW16& operator=(
 const EImageBW16& other
 )
```

### Parameters

*other*

Another EImageBW16 object.

### Remarks

This method copies all the supplied image properties (content, attributes, sub-ROIs...) into the current object.

# 4.88. EImageBW32 Class

The EImageBW32 class is used to represent rectangular regions of interest inside EBW32 gray-level images. See ROIs.

**Base Class:** EROIBW32

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EImageBW32 | Constructs a EImageBW32 image. |
| operator= | Copies a EImageBW32 image. |

# mageBW32::EImageBW32

Constructs a EImageBW32 image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EImageBW32(
 )
void EImageBW32(
 OEV_INT32 width,
 OEV_INT32 height
 )
```

```
void EImageBW32(
 const EImageBW32& other
 )
```

### Parameters

*width*

The width, in pixels.

*height*

The height, in pixels.

*other*

Another EImageBW32 object.

### Remarks

The constructor with no arguments creates a zero-sized image. You can modify the image size by calling EBaseROI::SetSize method. The sizing constructor constructs an image of the given size. See EBaseROI::SetSize for informations about the sizing restrictions. The copy constructor copies all the supplied image properties (content, attributes, sub-ROIs...) into the current object.

# EImageBW32::operator=

Copies a EImageBW32 image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageBW32& operator=(
 const EImageBW32& other
 )
```

### Parameters

*other*

Another EImageBW32 object.

### Remarks

This method copies all the supplied image properties (content, attributes, sub-ROIs...) into the current object.

# 4.89. EImageBW8 Class

The EImageBW8 class is used to represent rectangular regions of interest inside EBW8 gray-level images. See ROIs.

**Base Class:** EROIBW8
**Derived Class(es):** EGrabberImageBW8
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EImageBW8 | Constructs a EImageBW8 image. |
| operator= | Copies a EImageBW8 image. |

## mageBW8::EImageBW8

Constructs a EImageBW8 image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EImageBW8(
 )

void EImageBW8(
 OEV_INT32 width,
 OEV_INT32 height
 )
```

```
void EImageBW8(
 const EImageBW8& other
 )
```

### Parameters

*width*

The width, in pixels.

*height*

The height, in pixels.

*other*

Another EImageBW8 object.

### Remarks

The constructor with no arguments creates a zero-sized image. You can modify the image size by calling EBaseROI::SetSize method. The sizing constructor constructs an image of the given size. See EBaseROI::SetSize for informations about the sizing restrictions. The copy constructor copies all the supplied image properties (content, attributes, sub-ROIs...) into the current object.

# EImageBW8::operator=

Copies a EImageBW8 image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageBW8& operator=(
 const EImageBW8& other
 )
```

### Parameters

*other*

Another EImageBW8 object.

### Remarks

This method copies all the supplied image properties (content, attributes, sub-ROIs...) into the current object.

# 4.90. EImageC15 Class

The EImageC15 class is used to represent rectangular regions of interest inside EC15 color images. See ROIs.

**Base Class:** EROIC15

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EImageC15 | Constructs a EImageC15 image. |
| operator= | Copies a EImageC15 image. |

# mageC15::EImageC15

Constructs a EImageC15 image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EImageC15(
 )
void EImageC15(
 OEV_INT32 width,
 OEV_INT32 height
 )
```

```
void EImageC15(
 const EImageC15& other
 )
```

### Parameters

*width*

The width, in pixels.

*height*

The height, in pixels.

*other*

Another EImageC15 object.

### Remarks

The constructor with no arguments creates a zero-sized image. You can modify the image size by calling EBaseROI::SetSize method. The sizing constructor constructs an image of the given size. See EBaseROI::SetSize for informations about the sizing restrictions. The copy constructor copies all the supplied image properties (content, attributes, sub-ROIs...) into the current object.

# EImageC15::operator=

Copies a EImageC15 image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageC15& operator=(
 const EImageC15& other
 )
```

### Parameters

*other*

Another EImageC15 object.

### Remarks

This method copies all the supplied image properties (content, attributes, sub-ROIs...) into the current object.

# 4.91. EImageC16 Class

The EImageC16 class is used to represent rectangular regions of interest inside EC16 color images. See ROIs.

**Base Class:** EROIC16

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EImageC16 | Constructs a EImageC16 image. |
| operator= | Copies a EImageC16 image. |

# mageC16::EImageC16

Constructs a EImageC16 image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EImageC16(
 )
void EImageC16(
 OEV_INT32 width,
 OEV_INT32 height
 )
```

```
void EImageC16(
 const EImageC16& other
 )
```

### Parameters

*width*

The width, in pixels.

*height*

The height, in pixels.

*other*

Another EImageC16 object.

### Remarks

The constructor with no arguments creates a zero-sized image. You can modify the image size by calling EBaseROI::SetSize method. The sizing constructor constructs an image of the given size. See EBaseROI::SetSize for informations about the sizing restrictions. The copy constructor copies all the supplied image properties (content, attributes, sub-ROIs...) into the current object.

# EImageC16::operator=

Copies a EImageC16 image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageC16& operator=(
 const EImageC16& other
 )
```

### Parameters

*other*

Another EImageC16 object.

### Remarks

This method copies all the supplied image properties (content, attributes, sub-ROIs...) into the current object.

# 4.92. EImageC24 Class

The EImageC24 class is used to represent rectangular regions of interest inside EC24 color images. See ROIs.

**Base Class:** EROIC24
**Derived Class(es):** EGrabberImageC24
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EImageC24 | Constructs a EImageC24 image. |
| operator= | Copies a EImageC24 image. |

# mageC24::EImageC24

Constructs a EImageC24 image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EImageC24(
 )

void EImageC24(
 OEV_INT32 width,
 OEV_INT32 height
 )
```

```
void EImageC24(
 const EImageC24& other
 )
```

### Parameters

*width*

The width, in pixels.

*height*

The height, in pixels.

*other*

Another EImageC24 object.

### Remarks

The constructor with no arguments creates a zero-sized image. You can modify the image size by calling EBaseROI::SetSize method. The sizing constructor constructs an image of the given size. See EBaseROI::SetSize for informations about the sizing restrictions. The copy constructor copies all the supplied image properties (content, attributes, sub-ROIs...) into the current object.

# EImageC24::operator=

Copies a EImageC24 image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageC24& operator=(
 const EImageC24& other
 )
```

### Parameters

*other*

Another EImageC24 object.

### Remarks

This method copies all the supplied image properties (content, attributes, sub-ROIs...) into the current object.

# 4.93. EImageC24A Class

The EImageC24A class is used to represent rectangular regions of interest inside EC24A color images. See ROIs.

**Base Class:** EROIC24A

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EImageC24A | Constructs a EImageC24A image. |
| operator= | Copies a EImageC24A image. |

# mageC24A::EImageC24A

Constructs a EImageC24A image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EImageC24A(
 )
void EImageC24A(
 OEV_INT32 width,
 OEV_INT32 height
 )
```

```
void EImageC24A(
 const EImageC24A& other
 )
```

### Parameters

*width*

The width, in pixels.

*height*

The height, in pixels.

*other*

Another EImageC24A object.

### Remarks

The constructor with no arguments creates a zero-sized image. You can modify the image size by calling EBaseROI::SetSize method. The sizing constructor constructs an image of the given size. See EBaseROI::SetSize for informations about the sizing restrictions. The copy constructor copies all the supplied image properties (content, attributes, sub-ROIs...) into the current object.

# EImageC24A::operator=

Copies a EImageC24A image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageC24A& operator=(
 const EImageC24A& other
 )
```

### Parameters

*other*

Another EImageC24A object.

### Remarks

This method copies all the supplied image properties (content, attributes, sub-ROIs...) into the current object.

# 4.94. EImageC48 Class

The EImageC48 class is used to represent rectangular regions of interest inside EC48 color images. See ROIs.

**Base Class:** EROIC48

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EImageC48 | Constructs a EImageC48 image. |
| operator= | Copies a EImageC48 image. |

# mageC48::EImageC48

Constructs a EImageC48 image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EImageC48(
 )
void EImageC48(
 OEV_INT32 width,
 OEV_INT32 height
 )
```

```
void EImageC48(
 const EImageC48& other
 )
```

### Parameters

*width*

The width, in pixels.

*height*

The height, in pixels.

*other*

Another EImageC48 object.

### Remarks

The constructor with no arguments creates a zero-sized image. You can modify the image size by calling EBaseROI::SetSize method. The sizing constructor constructs an image of the given size. See EBaseROI::SetSize for informations about the sizing restrictions. The copy constructor copies all the supplied image properties (content, attributes, sub-ROIs...) into the current object.

# EImageC48::operator=

Copies a EImageC48 image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageC48& operator=(
 const EImageC48& other
 )
```

### Parameters

*other*

Another EImageC48 object.

### Remarks

This method copies all the supplied image properties (content, attributes, sub-ROIs...) into the current object.

# 4.95. EImageEncoder Class

This class is responsible for the encoding of an image into an ECodedImage2 object.

### Remarks

This class is responsible for the extraction of the runs in a source image, the aggregation of the runs into objects, as well as the proper handling of the continuous mode. It also provides methods to configure the image segmentation process.

The segmentation process classifies the pixels of the source image according to their value to create a set of layers. In each layer taken separately, the encoding process then assembles the connected pixels to build the coded elements (blobs).

By default, the segmentation method consists of grayscale single thresholding, with automatic threshold selection (as determined by the minimum residue rule).

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EImageEncoder | Constructs an image encoder. |
| Encode | Encodes an image or an ROI as a coded image. |
| FlushContinuousMode | Resets the continuous mode, emptying the internal memory, after writing the objects that are not yet completed in a coded image. |
| GetBinaryImageSegmenter | Returns a reference to the internal instance of the the segmenter for binary images, in order to set its parameters. |
| GetColorRangeThresholdSegmenter | Returns a reference to the internal instance of the the segmenter for color-range thresholding, in order to set its parameters. |

| | |
|---|---|
| GetColorSingleThresholdSegmenter | Returns a reference to the internal instance of the the segmenter for single color thresholding, in order to set its parameters. |
| GetContinuousModeEnabled | Continuous mode enabling status. |
| GetContinuousModeMaxHeight | Maximum number of rows that are kept in memory in the continuous mode. |
| GetEncodingConnexity | Connexity mode. |
| GetGrayscaleDoubleThresholdSegmenter | Returns a reference to the internal instance of the the segmenter for double color thresholding, in order to set its parameters. |
| GetGrayscaleSingleThresholdSegmenter | Returns a reference to the internal instance of the the segmenter for single grayscale thresholding, in order to set its parameters. |
| GetImageRangeSegmenter | Returns a reference to the internal instance of the the segmenter for pixel-by-pixel, range-based thresholding, in order to set its parameters. |
| GetLabeledImageSegmenter | Returns a reference to the internal instance of the the segmenter thats maps the value of the pixels directly to a layer index, in order to set its parameters. |
| GetReferenceImageSegmenter | Returns a reference to the internal instance of the the segmenter for single pixel-by-pixel thresholding, in order to set its parameters. |

| | |
|---|---|
| GetSegmentationMethod | Segmentation method used during the encoding. |
| ResetContinuousMode | Resets the continuous mode, emptying the internal memory. |
| SetContinuousModeEnabled | Continuous mode enabling status. |
| SetContinuousModeMaxHeight | Maximum number of rows that are kept in memory in the continuous mode. |
| SetEncodingConnexity | Connexity mode. |
| SetSegmentationMethod | Segmentation method used during the encoding. |

# EImageEncoder::GetBinaryImageSegmenter

Returns a reference to the internal instance of the the segmenter for binary images, in order to set its parameters.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EBinaryImageSegmenter& GetBinaryImageSegmenter()
```

# EImageEncoder::GetColorRangeThresholdSegmenter

Returns a reference to the internal instance of the the segmenter for color-range thresholding, in order to set its parameters.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EColorRangeThresholdSegmenter& GetColorRangeThresholdSegmenter()
```

# EImageEncoder::GetColorSingleThresholdSegmenter

Returns a reference to the internal instance of the the segmenter for single color thresholding, in order to set its parameters.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EColorSingleThresholdSegmenter& GetColorSingleThresholdSegmenter()
```

# EImageEncoder::GetContinuousModeEnabled

# EImageEncoder::SetContinuousModeEnabled

Continuous mode enabling status.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool GetContinuousModeEnabled() const
void SetContinuousModeEnabled(bool enabled)
```

### Remarks

In the continuous mode, objects are constructed over a sequence of images: The image encoder encodes only the objects that contain no run touching the last row of the source image. The objects touching the inferior border of the image are not written in the coded image: These objects are indeed expected to continue in the subsequent image chunks. Such objects are kept in memory, and are consumed when analyzing the subsequent images.

# EImageEncoder::GetContinuousModeMaxHeight

# EImageEncoder::SetContinuousModeMaxHeight

Maximum number of rows that are kept in memory in the continuous mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetContinuousModeMaxHeight() const
void SetContinuousModeMaxHeight(OEV_UINT32 maxHeight)
```

### Remarks

This property can be used to put a bound on the size of the internal memory of the image encoder in the continuous mode. If this property is set to zero, then memory can grow arbitrarily (there is no maximum number of rows).

# EImageEncoder::EImageEncoder

Constructs an image encoder.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EImageEncoder(
 const EImageEncoder& other
 )
void EImageEncoder(
 )
```

### Parameters

*other*

-

# EImageEncoder::Encode

Encodes an image or an ROI as a coded image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Encode(
 const EROIBW1& sourceImage,
 ECodedImage2& codedImage
 )
void Encode(
 const EROIBW8& sourceImage,
 ECodedImage2& codedImage
 )
void Encode(
 const EROIBW16& sourceImage,
 ECodedImage2& codedImage
 )
```

```
void Encode(
 const EROIC24& sourceImage,
 ECodedImage2& codedImage
 )

void Encode(
 const EROIBW1& sourceImage,
 const EROIBW8& inputMask,
 ECodedImage2& codedImage
 )

void Encode(
 const EROIBW8& sourceImage,
 const EROIBW8& inputMask,
 ECodedImage2& codedImage
 )

void Encode(
 const EROIBW16& sourceImage,
 const EROIBW8& inputMask,
 ECodedImage2& codedImage
 )

void Encode(
 const EROIC24& sourceImage,
 const EROIBW8& inputMask,
 ECodedImage2& codedImage
 )

void Encode(
 const EROIBW1& sourceImage,
 const ERegion& region,
 ECodedImage2& codedImage
 )

void Encode(
 const EROIBW8& sourceImage,
 const ERegion& region,
 ECodedImage2& codedImage
 )

void Encode(
 const EROIBW16& sourceImage,
 const ERegion& region,
 ECodedImage2& codedImage
 )
```

```
void Encode(
 const EROIC24& sourceImage,
 const ERegion& region,
 ECodedImage2& codedImage
 )
```

### Parameters

*sourceImage*

The input image that is to be encoded.

*codedImage*

The coded image that will hold the result of the encoding process.

*inputMask*

The possible input Flexible Mask that restricts the encoding. The input mask is a grayscale image having the same height and the same width as the source image. Any pixel in the source image that is covered by a value of **0** in the input mask will not get encoded in any layer. Any other pixel value in the input mask causes the pixel to be a candidate for the encoding.

*region*

Region of the input image that is to be encoded.

### Remarks

The previous content of the result coded image is discarded.

# EImageEncoder::GetEncodingConnexity

# EImageEncoder::SetEncodingConnexity

Connexity mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EEncodingConnexity GetEncodingConnexity() const
```

```
void SetEncodingConnexity(Euresys::Open_eVision_2_10::EEncodingConnexity connexity)
```

### Remarks

The connexity mode specifies the conditions that must hold for neighboring pixels to belong to the same object.

# EImageEncoder::FlushContinuousMode

Resets the continuous mode, emptying the internal memory, after writing the objects that are not yet completed in a coded image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void FlushContinuousMode(
 ECodedImage2& codedImage
 )
```

### Parameters

*codedImage*

The coded image in which the not-yet-completed objects are written.

# EImageEncoder::GetGrayscaleDoubleThresholdSegmenter

Returns a reference to the internal instance of the the segmenter for double color thresholding, in order to set its parameters.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EGrayscaleDoubleThresholdSegmenter& GetGrayscaleDoubleThresholdSegmenter()
```

# EImageEncoder::GetGrayscaleSingleThresholdSegmenter

Returns a reference to the internal instance of the the segmenter for single grayscale thresholding, in order to set its parameters.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EGrayscaleSingleThresholdSegmenter& GetGrayscaleSingleThresholdSegmenter()
```

# EImageEncoder::GetImageRangeSegmenter

Returns a reference to the internal instance of the the segmenter for pixel-by-pixel, range-based thresholding, in order to set its parameters.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EImageRangeSegmenter& GetImageRangeSegmenter()
```

# EImageEncoder::GetLabeledImageSegmenter

Returns a reference to the internal instance of the the segmenter thats maps the value of the pixels directly to a layer index, in order to set its parameters.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
ELabeledImageSegmenter& GetLabeledImageSegmenter()
```

# EImageEncoder::GetReferenceImageSegmenter

Returns a reference to the internal instance of the the segmenter for single pixel-by-pixel thresholding, in order to set its parameters.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EReferenceImageSegmenter& GetReferenceImageSegmenter()
```

# EImageEncoder::ResetContinuousMode

Resets the continuous mode, emptying the internal memory.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ResetContinuousMode(
 )
```

## EImageEncoder::GetSegmentationMethod

## EImageEncoder::SetSegmentationMethod

Segmentation method used during the encoding.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::ESegmentationMethod GetSegmentationMethod() const
void SetSegmentationMethod(Euresys::Open_eVision_2_10::ESegmentationMethod method)
```

# 4.96. EImageRangeSegmenter Class

Segments an image using a pixel-by-pixel double threshold given as two images.

### Remarks

This segmenter is applicable to EROIBW8, EROIBW16 and EROIC24 images. It produces coded images with two layers. The low threshold and the high threshold are defined for each pixel individually by means of two reference images of the same type as the source image: the Low Image and the High Image.

For grayscales images, the White layer (usually, with index 1) contains unmasked pixels having a gray value in a range defined by the gray value of the corresponding unmasked pixels in the Low Image and the High Image.

For RGB color images, the White layer (usually, with index 1) contains unmasked pixels having a color inside the cube of the color space defined by the colors of the corresponding unmasked pixels in the Low Image and the High Image.

The Black layer (usually, with index 0) contains the remaining unmasked pixels.

**Base Class:** ETwoLayersImageSegmenter

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

## Methods

| | |
|---|---|
| GetHighImageBW16 | High image for the segmentation of EROIBW16 images. |
| GetHighImageBW8 | High image for the segmentation of EROIBW8 images. |
| GetHighImageC24 | High image for the segmentation of EROIC24 images. |
| GetLowImageBW16 | Low image for the segmentation of EROIBW16 images. |
| GetLowImageBW8 | Low image for the segmentation of EROIBW8 images. |
| GetLowImageC24 | Low image for the segmentation of EROIC24 images. |
| SetBlackLayerEncoded | Black layer encoding status. |

| | |
|---|---|
| SetBlackLayerIndex | Index of the black layer in the destination coded image. |
| SetHighImageBW16 | High image for the segmentation of EROIBW16 images. |
| SetHighImageBW8 | High image for the segmentation of EROIBW8 images. |
| SetHighImageC24 | High image for the segmentation of EROIC24 images. |
| SetLowImageBW16 | Low image for the segmentation of EROIBW16 images. |
| SetLowImageBW8 | Low image for the segmentation of EROIBW8 images. |
| SetLowImageC24 | Low image for the segmentation of EROIC24 images. |
| SetWhiteLayerEncoded | White layer encoding status. |
| SetWhiteLayerIndex | Index of the white layer in the destination coded image. |

# EImageRangeSegmenter::SetBlackLayerEncoded

Black layer encoding status.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

[C++]

```
void SetBlackLayerEncoded(bool encode)
```

# EImageRangeSegmenter::SetBlackLayerIndex

Index of the black layer in the destination coded image.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

[C++]

```
void SetBlackLayerIndex(OEV_UINT32 index)
```

## Remarks

Setting this property automatically switches on the encoding of the black layer.

# EImageRangeSegmenter::GetHighImageBW16

# EImageRangeSegmenter::SetHighImageBW16

High image for the segmentation of EROIBW16 images.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

[C++]

```
const EROIBW16* GetHighImageBW16() const
```

```
void SetHighImageBW16(const EROIBW16* image)
```

## EImageRangeSegmenter::GetHighImageBW8

## EImageRangeSegmenter::SetHighImageBW8

High image for the segmentation of EROIBW8 images.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
const EROIBW8* GetHighImageBW8() const
void SetHighImageBW8(const EROIBW8* image)
```

## EImageRangeSegmenter::GetHighImageC24

## EImageRangeSegmenter::SetHighImageC24

High image for the segmentation of EROIC24 images.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
const EROIC24* GetHighImageC24() const
void SetHighImageC24(const EROIC24* image)
```

# EImageRangeSegmenter::GetLowImageBW16

# EImageRangeSegmenter::SetLowImageBW16

Low image for the segmentation of EROIBW16 images.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
const EROIBW16* GetLowImageBW16() const
void SetLowImageBW16(const EROIBW16* image)
```

# EImageRangeSegmenter::GetLowImageBW8

# EImageRangeSegmenter::SetLowImageBW8

Low image for the segmentation of EROIBW8 images.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
const EROIBW8* GetLowImageBW8() const
void SetLowImageBW8(const EROIBW8* image)
```

# EImageRangeSegmenter::GetLowImageC24

# EImageRangeSegmenter::SetLowImageC24

Low image for the segmentation of EROIC24 images.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
const EROIC24* GetLowImageC24() const
void SetLowImageC24(const EROIC24* image)
```

# EImageRangeSegmenter::SetWhiteLayerEncoded

White layer encoding status.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
void SetWhiteLayerEncoded(bool encode)
```

# EImageRangeSegmenter::SetWhiteLayerIndex

Index of the white layer in the destination coded image.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
void SetWhiteLayerIndex(OEV_UINT32 index)
```

### Remarks

Setting this property automatically switches on the encoding of the white layer.

# 4.97. EImageSegmenter Class

Base class from which all the segmenters derive.

**Derived Class(es):** ETwoLayersImageSegmenter EThreeLayersImageSegmenter ELabeledImageSegmenter
**Namespace:** Euresys::Open_eVision_2_10::Segmenters

# 4.98. EIntegerRange Class

Represents a range of integer values.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EIntegerRange | Creates an EIntegerRange object. |
| GetCenter | Center of the range. |
| GetLowerBound | Lower bound of the range. |

| | |
|---|---|
| GetSize | Size of the range. |
| GetUpperBound | Upper bound of the range. |
| IsInRange | Checks if a value is inside the range. |
| operator= | Assignment operator |
| SetBounds | Sets the bounds of the range. |
| SetFromBaseAndAbsoluteTolerance | Sets the bounds of the range from a base value and an absolute tolerance from this base value. |
| SetFromBaseAndRelativeTolerance | Sets the bounds of the range from a base value and a relative tolerance from this base value. |
| Update | Updates the range. If the value passed is outside of the range bounds, they are modified so they include the value. |

# EIntegerRange::GetCenter

Center of the range.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetCenter() const
```

# EIntegerRange::EIntegerRange

Creates an EIntegerRange object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EIntegerRange(
 )
void EIntegerRange(
 int min,
 int max
 )
void EIntegerRange(
 const EIntegerRange& range
 )
```

## Parameters

*min*

Lower bound of the range.

*max*

Upper bound of the range.

*range*

Range to copy.

# EIntegerRange::IsInRange

Checks if a value is inside the range.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool IsInRange(
 int value,
 bool lowerBoundInclusive,
 bool upperBoundInclusive
 )

bool IsInRange(
 float value,
 bool lowerBoundInclusive,
 bool upperBoundInclusive
 )
```

### Parameters

*value*

Value to test.

*lowerBoundInclusive*

Indicates if the lower bound should be considered inside the range (true) or outside (false).

*upperBoundInclusive*

Indicates if the upper bound should be considered inside the range (true) or outside (false).

# EIntegerRange::GetLowerBound

Lower bound of the range.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
int GetLowerBound() const
```

# EIntegerRange::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EIntegerRange& operator=(
 const EIntegerRange& other
 )
```

## Parameters

*other*

-

# EIntegerRange::SetBounds

Sets the bounds of the range.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetBounds(
 int min,
 int max
 )
```

### Parameters

*min*

Lower bound of the range.

*max*

Upper bound of the range.

# EIntegerRange::SetFromBaseAndAbsoluteTolerance

Sets the bounds of the range from a base value and an absolute tolerance from this base value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromBaseAndAbsoluteTolerance(
 int baseValue,
 int tolerance
 )
```

### Parameters

*baseValue*

Base value.

*tolerance*

Absolute tolerance around the base value. Must be positive.

### Remarks

The range will be set with a lower bound of (base - tolerance) and an upper bound of (base + tolerance).

# EIntegerRange::SetFromBaseAndRelativeTolerance

Sets the bounds of the range from a base value and a relative tolerance from this base value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromBaseAndRelativeTolerance(
 int baseValue,
 float tolerance
 )
```

### Parameters

*baseValue*

Base value.

*tolerance*

Relative tolerance around the base value. Must be positive.

### Remarks

The range will be set with a lower bound of (base - (base * tolerance)) and an upper bound of (base + (base * tolerance)).

# EIntegerRange::GetSize

Size of the range.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetSize() const
```

# EIntegerRange::Update

Updates the range. If the value passed is outside of the range bounds, they are modified so they include the value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Update(
 int value
 )
```

### Parameters

*value*

Value to be included in the range.

# EIntegerRange::GetUpperBound

Upper bound of the range.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetUpperBound() const
```

# 4.99. EKernel Class

Kernel for use in convolution operations.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EKernel | Constructs an EKernel object. |
| GetGain | Global gain. |
| GetKernelData | Returns the convolution coefficient of given indices. |
| GetOffset | Global offset (a constant added to the convolution result). |
| GetOutsideValue | Out-of-limits image value (only influences the result along image edges). |
| GetRawDataPtr | Pointer to the upper left convolution coefficient. |
| GetRectifier | Rectification mode. This property allows specifying how negative convolution result values are handled. |
| GetSizeX | Number of coefficients along a row. |
| GetSizeY | Number of coefficients along a column. |
| SetGain | Global gain. |

| | |
|---|---|
| SetKernelData | Sets the convolution coefficient values at the given indices. |
| SetOffset | Global offset (a constant added to the convolution result). |
| SetOutsideValue | Out-of-limits image value (only influences the result along image edges). |
| SetRectifier | Rectification mode. This property allows specifying how negative convolution result values are handled. |
| SetSize | E- |

K

# ernel::EKernel

Constructs an EKernel object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EKernel(
 const EKernel& other
 )

void EKernel(
 )
```

```
void EKernel(
 OEV_INT16 sizeX,
 OEV_INT16 sizeY,
 float gain,
 OEV_UINT32 offset,
 Euresys::Open_eVision_2_10::EKernelRectifier rectifier,
 OEV_UINT32 outsideValue
 )
void EKernel(
 Euresys::Open_eVision_2_10::EKernelType KernelType
 )
```

### Parameters

*other*

-

*sizeX*

Number of coefficients along a row.

*sizeY*

Number of coefficients along a column.

*gain*

Global gain.

*offset*

Global offset.

*rectifier*

Rectification mode, as defined by EKernelRectifier.

*outsideValue*

Out-of-limits image value.

*KernelType*

Kernel type, as defined by EKernelType.

### Remarks

The default constructor constructs a void kernel. A void kernel has no associated convolution coefficients. The sizing constructor constructs a kernel of given size and global parameters. The third constructor constructs a kernel of a predefined type.

# EKernel::GetGain

# EKernel::SetGain

Global gain.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetGain()
void SetGain(float f32Gain)
```

### Remarks

Before the global gain is applied, the coefficients are normalized so that their sum equals one, unless their sum equals zero (as is the case for a derivation operator). The rectification enables to handle the negative values that may appear after convolution.

# EKernel::GetKernelData

Returns the convolution coefficient of given indices.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetKernelData(
 OEV_INT32 columnIndex,
 OEV_INT32 rowIndex,
 float& coefficientValue
 )
```

### Parameters

*columnIndex*

Column index, from **0** on, increasing rightwards.

*rowIndex*

Row index, from **0** on, increasing downwards.

*coefficientValue*

Reference to the coefficient value.

# EKernel::GetOffset

# EKernel::SetOffset

Global offset (a constant added to the convolution result).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetOffset()
void SetOffset(OEV_UINT32 un32Offset)
```

# EKernel::GetOutsideValue

# EKernel::SetOutsideValue

Out-of-limits image value (only influences the result along image edges).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetOutsideValue()
void SetOutsideValue(OEV_UINT32 un32OutsideValue)
```

# EKernel::GetRawDataPtr

Pointer to the upper left convolution coefficient.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void* GetRawDataPtr()
```

## Remarks

This pointer is actually the base address of a float array containing all coefficients.

# EKernel::GetRectifier

# EKernel::SetRectifier

Rectification mode. This property allows specifying how negative convolution result values are handled.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EKernelRectifier GetRectifier()
```

```
void SetRectifier(Euresys::Open_eVision_2_10::EKernelRectifier rectifier)
```

# EKernel::SetKernelData

Sets the convolution coefficient values at the given indices.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetKernelData(
 OEV_INT32 columnIndex,
 OEV_INT32 rowIndex,
 float coefficientValue
 )

void SetKernelData(
 float coefficientValue00,
 float coefficientValue10,
 float coefficientValue20,
 float coefficientValue01,
 float coefficientValue11,
 float coefficientValue21,
 float coefficientValue02,
 float coefficientValue12,
 float coefficientValue22
 )
```

```
void SetKernelData(
 float coefficientValue00,
 float coefficientValue10,
 float coefficientValue20,
 float coefficientValue30,
 float coefficientValue40,
 float coefficientValue01,
 float coefficientValue11,
 float coefficientValue21,
 float coefficientValue31,
 float coefficientValue41,
 float coefficientValue02,
 float coefficientValue12,
 float coefficientValue22,
 float coefficientValue32,
 float coefficientValue42,
 float coefficientValue03,
 float coefficientValue13,
 float coefficientValue23,
 float coefficientValue33,
 float coefficientValue43,
 float coefficientValue04,
 float coefficientValue14,
 float coefficientValue24,
 float coefficientValue34,
 float coefficientValue44
 )
```

```
void SetKernelData(
 float coefficientValue00,
 float coefficientValue10,
 float coefficientValue20,
 float coefficientValue30,
 float coefficientValue40,
 float coefficientValue50,
 float coefficientValue60,
 float coefficientValue01,
 float coefficientValue11,
 float coefficientValue21,
 float coefficientValue31,
 float coefficientValue41,
 float coefficientValue51,
 float coefficientValue61,
 float coefficientValue02,
 float coefficientValue12,
 float coefficientValue22,
 float coefficientValue32,
 float coefficientValue42,
 float coefficientValue52,
 float coefficientValue62,
 float coefficientValue03,
 float coefficientValue13,
 float coefficientValue23,
 float coefficientValue33,
 float coefficientValue43,
 float coefficientValue53,
 float coefficientValue63,
 float coefficientValue04,
 float coefficientValue14,
 float coefficientValue24,
 float coefficientValue34,
 float coefficientValue44,
 float coefficientValue54,
 float coefficientValue64,
 float coefficientValue05,
 float coefficientValue15,
 float coefficientValue25,
 float coefficientValue35,
 float coefficientValue45,
 float coefficientValue55,
 float coefficientValue65,
 float coefficientValue06,
```

```
  float coefficientValue16,
  float coefficientValue26,
  float coefficientValue36,
  float coefficientValue46,
  float coefficientValue56,
  float coefficientValue66
  )
```

### Parameters

*columnIndex*

Column index, from **0** on, increasing rightwards.

*rowIndex*

Row index, from **0** on, increasing downwards.

*coefficientValue*

New coefficientValue.

*coefficientValue00*

Coefficient value at corresponding column and row indices.

*coefficientValue10*

Coefficient value at corresponding column and row indices.

*coefficientValue20*

Coefficient value at corresponding column and row indices.

*coefficientValue01*

Coefficient value at corresponding column and row indices.

*coefficientValue11*

Coefficient value at corresponding column and row indices.

*coefficientValue21*

Coefficient value at corresponding column and row indices.

*coefficientValue02*

Coefficient value at corresponding column and row indices.

*coefficientValue12*

Coefficient value at corresponding column and row indices.

*coefficientValue22*

Coefficient value at corresponding column and row indices.

*coefficientValue30*

Coefficient value at corresponding column and row indices.

*coefficientValue40*

Coefficient value at corresponding column and row indices.

*coefficientValue31*

Coefficient value at corresponding column and row indices.

*coefficientValue41*

Coefficient value at corresponding column and row indices.

*coefficientValue32*

Coefficient value at corresponding column and row indices.

*coefficientValue42*

Coefficient value at corresponding column and row indices.

*coefficientValue03*

Coefficient value at corresponding column and row indices.

*coefficientValue13*

Coefficient value at corresponding column and row indices.

*coefficientValue23*

Coefficient value at corresponding column and row indices.

*coefficientValue33*

Coefficient value at corresponding column and row indices.

*coefficientValue43*

Coefficient value at corresponding column and row indices.

*coefficientValue04*

Coefficient value at corresponding column and row indices.

*coefficientValue14*

Coefficient value at corresponding column and row indices.

*coefficientValue24*

Coefficient value at corresponding column and row indices.

*coefficientValue34*

Coefficient value at corresponding column and row indices.

*coefficientValue44*

Coefficient value at corresponding column and row indices.

*coefficientValue50*

Coefficient value at corresponding column and row indices.

*coefficientValue60*

Coefficient value at corresponding column and row indices.

*coefficientValue51*

Coefficient value at corresponding column and row indices.

*coefficientValue61*

Coefficient value at corresponding column and row indices.

*coefficientValue52*

Coefficient value at corresponding column and row indices.

*coefficientValue62*

Coefficient value at corresponding column and row indices.

*coefficientValue53*

Coefficient value at corresponding column and row indices.

*coefficientValue63*

Coefficient value at corresponding column and row indices.

*coefficientValue54*

Coefficient value at corresponding column and row indices.

*coefficientValue64*

Coefficient value at corresponding column and row indices.

*coefficientValue05*

Coefficient value at corresponding column and row indices.

*coefficientValue15*

Coefficient value at corresponding column and row indices.

*coefficientValue25*

Coefficient value at corresponding column and row indices.

*coefficientValue35*

Coefficient value at corresponding column and row indices.

*coefficientValue45*

Coefficient value at corresponding column and row indices.

*coefficientValue55*

Coefficient value at corresponding column and row indices.

*coefficientValue65*

Coefficient value at corresponding column and row indices.

*coefficientValue06*

Coefficient value at corresponding column and row indices.

*coefficientValue16*

Coefficient value at corresponding column and row indices.

*coefficientValue26*

Coefficient value at corresponding column and row indices.

*coefficientValue36*

Coefficient value at corresponding column and row indices.

*coefficientValue46*

Coefficient value at corresponding column and row indices.

*coefficientValue56*

Coefficient value at corresponding column and row indices.

*coefficientValue66*

Coefficient value at corresponding column and row indices.

### Remarks

The function can also set the coefficient values for 3x3, 5x5 and 7x7 kernels.

# EKernel::SetSize

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetSize(
 OEV_INT16 n16SizeX,
 OEV_INT16 n16SizeY
 )
```

### Parameters

*n16SizeX*

-

*n16SizeY*

-

# EKernel::GetSizeX

Number of coefficients along a row.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT16 GetSizeX()
```

# EKernel::GetSizeY

Number of coefficients along a column.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT16 GetSizeY()
```

# 4.100. ELabeledImageSegmenter Class

Segments an image by mapping the value of the pixels directly to a layer index.

## Remarks

This segmenter is applicable to EROIBW8 and EROIBW16 grayscale images. It produces coded images with a varying number of layers. The layer with index **N** contains all the unmasked pixels having a gray value equal to **N**.

By default, the segmentation is restricted to the range of layers whose index is between **0** and **255** (inclusive). This default range can be changed through ELabeledImageSegmenter::MinLayer and ELabeledImageSegmenter::MaxLayer.

**Base Class:** EImageSegmenter

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

## Methods

| | |
|---|---|
| GetMaxLayer | High index of the range of layers to be encoded. |
| GetMinLayer | Low index of the range of layers to be encoded. |

| | |
|---|---|
| SetMaxLayer | High index of the range of layers to be encoded. |
| SetMinLayer | Low index of the range of layers to be encoded. |

# ELabeledImageSegmenter::GetMaxLayer

# ELabeledImageSegmenter::SetMaxLayer

High index of the range of layers to be encoded.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
EBW16 GetMaxLayer() const
void SetMaxLayer(EBW16 maxLayer)
```

# ELabeledImageSegmenter::GetMinLayer

# ELabeledImageSegmenter::SetMinLayer

Low index of the range of layers to be encoded.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
EBW16 GetMinLayer() const
void SetMinLayer(EBW16 minLayer)
```

# 4.101. ELandmark Class

The landMark descriptor class.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| ELandmark | Constructs a ELandmark object. |
| GetSensorX | SensorX. |
| GetSensorY | SensorY. |
| GetWorldX | WorldX. |
| GetWorldY | WorldY. |
| operator= | Assignement operator. |
| SetSensorX | SensorX. |

| | |
|---|---|
| SetSensorY | SensorY. |
| SetWorldX | WorldX. |
| SetWorldY | WorldY. |

# andmark::ELandmark

Constructs a ELandmark object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ELandmark(
 const ELandmark& other
 )
void ELandmark(
 )
```

### Parameters

*other*

-

# ELandmark::operator=

Assignement operator.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ELandmark& operator=(
 const ELandmark& other
 )
```

### Parameters

*other*

ELandmark object to copy.

# ELandmark::GetSensorX

# ELandmark::SetSensorX

SensorX.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSensorX() const
void SetSensorX(float val)
```

# ELandmark::GetSensorY

# ELandmark::SetSensorY

SensorY.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSensorY() const
void SetSensorY(float val)
```

# ELandmark::GetWorldX

# ELandmark::SetWorldX

WorldX.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetWorldX() const
void SetWorldX(float val)
```

# ELandmark::GetWorldY

# ELandmark::SetWorldY

WorldY.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetWorldY() const
void SetWorldY(float val)
```

# 4.102. ELaserLineExtractor Class

Manages a laser line extraction context.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| ELaserLineExtractor | Creates an ELaserLineExtractor object. |
| ExtractProfileFromFrame | Extracts a profile from a frame and adds it to the current depth map.<br>Returns true if the depth map is complete and ready for further processing. |
| GetAnalysisMode | Analysis mode. |
| GetAnalysisThreshold | Analysis threshold. Set this value to eliminate noise. |
| GetDepthMap | Returns the current depth map. |
| GetEnableSmoothing | Enables or disables grayscale profile smoothing before extraction. |

| | |
|---|---|
| GetProfile | Returns the last extracted profile. |
| operator= | Assignment operator |
| SetAnalysisMode | Analysis mode. |
| SetAnalysisThreshold | Analysis threshold. Set this value to eliminate noise. |
| SetEnableSmoothing | Enables or disables grayscale profile smoothing before extraction. |
| SetSmoothingParameters | Sets the parameters of the smoothing kernel. |

# ELaserLineExtractor::GetAnalysisMode

# ELaserLineExtractor::SetAnalysisMode

Analysis mode.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
Euresys::Open_eVision_2_10::Easy3D::EMaximumAnalysisMode GetAnalysisMode()
void SetAnalysisMode(Euresys::Open_eVision_2_10::Easy3D::EMaximumAnalysisMode mode)
```

# ELaserLineExtractor::GetAnalysisThreshold

# ELaserLineExtractor::SetAnalysisThreshold

Analysis threshold. Set this value to eliminate noise.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
int GetAnalysisThreshold()
void SetAnalysisThreshold(int threshold)
```

## Remarks

In the center of gravity (COG) analysis mode, this threshold is used to discriminate peaks and should be set accordingly.

# ELaserLineExtractor::GetDepthMap

Returns the current depth map.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const EDepthMap16& GetDepthMap()
```

## Remarks

Should be called only when the previous call to ExtractProfileFromFrame() returned true. Otherwise, the depth map returnes will be incomplete.

# ELaserLineExtractor::ELaserLineExtractor

Creates an ELaserLineExtractor object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ELaserLineExtractor(
 int frameWidth,
 int frameHeight,
 int numFramesPerMap,
 float zResolution
 )
void ELaserLineExtractor(
 const ELaserLineExtractor& other
 )
```

## Parameters

*frameWidth*

Width of the frames from which the profiles will be extracted.

*frameHeight*

Height of the frames from which the profiles will be extracted.

*numFramesPerMap*

Number of frames (and thus profiles) to be used per depth map. Each extracted profile create a line in the depth map.

*zResolution*

Optional parameter for the Z resolution of the extracted profile.
With a value of 0, the resolution will be automatically calculated to maximize the sub-pixel accuracy.

*other*

The object that should be copied

# ELaserLineExtractor::GetEnableSmoothing

# ELaserLineExtractor::SetEnableSmoothing

Enables or disables grayscale profile smoothing before extraction.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool GetEnableSmoothing()
void SetEnableSmoothing(bool enable)
```

# ELaserLineExtractor::ExtractProfileFromFrame

Extracts a profile from a frame and adds it to the current depth map.
Returns true if the depth map is complete and ready for further processing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool ExtractProfileFromFrame(
 const EROIBW8& frame
 )
```

## Parameters

*frame*

Frame from which the profile will be extracted.

# ELaserLineExtractor::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
ELaserLineExtractor& operator=(
 const ELaserLineExtractor& other
 )
```

## Parameters

*other*

The object that should be copied

# ELaserLineExtractor::GetProfile

Returns the last extracted profile.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
std::vector<float> GetProfile()
```

## Remarks

If a point could not be extracted, its value will be set to FLOAT_MAX.

# ELaserLineExtractor::SetSmoothingParameters

Sets the parameters of the smoothing kernel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetSmoothingParameters(
 int param0,
 int param1,
 int param2
 )
```

### Parameters

*param0*

First kernel parameter.

*param1*

Second kernel parameter.

*param2*

Third kernel parameter.

### Remarks

If enabled, the smoothing will be performed using the following formula: f[i] = (f[i-1] * param0) + (f[i] * param1) + (f[i+1] * param2).

# 4.103. ELine Class

Represents a model of a line segment in EasyGauge.

**Base Class:** EFrame

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| CopyTo | Copies all the data of the current ELine object into another ELine object and returns it. |
| ELine | Constructs a ELine object. |
| GetAngleBetweenLines | Computes the angle between two lines. |
| GetDistanceBetweenPointAndLine | Computes the distance between a point and a line. |
| GetEnd | End point coordinates of the ELine object. |
| GetIntersectionOfLines | Computes the intersection between two lines. |
| GetLength | Length of the ELine object. |
| GetOrg | Origin point coordinates of the ELine object. |
| GetPoint | Returns the coordinates of a point along the line. |
| GetProjectionOfPointOnLine | Computes the projection of a point on a line. |
| operator= | Copies all the data from another ELine object into the current ELine object |

| | |
|---|---|
| SetFromOriginAndEnd | Sets the geometric parameters (center coordinates, length, and rotation angle) of a ELine object. |
| SetFromTwoPoints | DEPRECATED (you should use ELine::SetFromOriginAndEnd) : Sets the geometric parameters (center coordinates, length, and rotation angle) of a ELine object. |
| SetLength | Length of the ELine object. |

# ine::CopyTo

Copies all the data of the current ELine object into another ELine object and returns it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ELine* CopyTo(
 ELine* other
 )
```

## Parameters

*other*

Pointer to the ELine object in which the current ELine object data have to be copied.

## Remarks

In case of a **NULL** pointer, a new ELine object will be created and returned.

# ELine::ELine

Constructs a ELine object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ELine(
 )
void ELine(
 const EPoint& center,
 float length,
 float angle
 )
void ELine(
 const EPoint& origin,
 const EPoint& end
 )
void ELine(
 const ELine& other
 )
```

## Parameters

*center*

Center coordinates of the line at its nominal position. The default value is **(0,0)**.

*length*

Nominal length of the line. The default value is **100**.

*angle*

Nominal rotation angle of the line. The default value is **0**.

*origin*

Origin point coordinates of the line.

*end*

End point coordinates of the line.

*other*

Another ELine object to be copied in the new ELine object.

# ELine::GetEnd

End point coordinates of the ELine object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetEnd() const
```

# ELine::GetAngleBetweenLines

Computes the angle between two lines.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetAngleBetweenLines(
 const ELine& line1,
 const ELine& line2
 )
```

## Parameters

*line1*

First line

*line2*

Second line

## Remarks

The angle returned by this function is signed in the trigonometric sense, meaning that angle(1,2) = -angle(2,1).

# ELine::GetDistanceBetweenPointAndLine

Computes the distance between a point and a line.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetDistanceBetweenPointAndLine(
 const EPoint& pt,
 const ELine& line,
 bool limited
 )
```

### Parameters

*pt*

The point.

*line*

The line.

*limited*

Indicates if the line parameter should be considered as an infinite line or as a segment.

# ELine::GetIntersectionOfLines

Computes the intersection between two lines.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 GetIntersectionOfLines(
 const ELine& line1,
 const ELine& line2,
 EPoint& intersection,
 bool limited
 )
```

### Parameters

*line1*

First line.

*line2*

Second line.

*intersection*

Found intersection.

*limited*

Indicates if the line parameters should be considered as infinite lines or as a segments.

### Remarks

The function returns the number of intersections found. It will return -1 if the two lines are overlapping.

# ELine::GetPoint

Returns the coordinates of a point along the line.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetPoint(
 float fraction
 )
```

### Parameters

*fraction*

Point location expressed as a fraction of the line length (range **[-1, +1]**).

# ELine::GetProjectionOfPointOnLine

Computes the projection of a point on a line.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetProjectionOfPointOnLine(
 const EPoint& pt,
 const ELine& line
 )
```

### Parameters

*pt*

The point.

*line*

The line.

# ELine::GetLength

# ELine::SetLength

Length of the ELine object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetLength()

void SetLength(float length)
```

### Remarks

By default, the length of the line is **100**, which means 100 pixels when the field of view is not calibrated, and 100 physical units in case of a calibrated field of view.

# ELine::operator=

Copies all the data from another ELine object into the current ELine object

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
ELine& operator=(
 const ELine& other
 )
```

### Parameters

*other*

ELine object to be copied

# ELine::GetOrg

Origin point coordinates of the ELine object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetOrg() const
```

# ELine::SetFromOriginAndEnd

Sets the geometric parameters (center coordinates, length, and rotation angle) of a ELine object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromOriginAndEnd(
 const EPoint& origin,
 const EPoint& end
 )
```

### Parameters

*origin*

Origin point coordinates of the line.

*end*

End point coordinates of the line.

# ELine::SetFromTwoPoints

DEPRECATED (you should use ELine::SetFromOriginAndEnd) : Sets the geometric parameters (center coordinates, length, and rotation angle) of a ELine object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromTwoPoints(
 const EPoint& origin,
 const EPoint& end
 )
```

### Parameters

*origin*

Origin point coordinates of the line.

*end*

End point coordinates of the line.

# 4.104. ELineGauge Class

Manages a line fitting gauge.

**Base Class:** ELineShape

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddSkipRange | Adds an item to the set of skip ranges and returns the index of the newly added range. |
| CopyTo | Copies all the data of the current ELineGauge object into another ELineGauge object, and returns it. |
| Drag | Moves a handle to a new position and updates the position parameters of the gauge. |
| Draw | Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode. |

| | |
|---|---|
| DrawWithCurrentPen | Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode. |
| ELineGauge | Constructs a line measurement context. |
| GetAverageDistance | Average distance between the sampled points and the fitted model. |
| GetClippingMode | Clipping mode, that allows to choose how the fitted segment length and center are computed. |
| GetFilteringThreshold | Relative filtering threshold, that is the fraction of the average distance between the sampled points and the fitted model above which a point is filtered out. |
| GetHVConstraint | Status of the restriction on the orientation of the point location gauge or model fitting sample paths. |
| GetKnownAngle | Flag indicating whether the slope of the line to be fitted is known or not. |
| GetMeasuredLine | Information pertaining to the fitted line. |
| GetMeasuredPeak | Returns information pertaining to the default derivative peak, along one of the sample paths of the gauge. |

| | |
|---|---|
| GetMeasuredPoint | Returns the coordinates of a sample point, measured along one of the sample paths of the gauge. |
| GetMinAmplitude | Offset added to the **Threshold** when a peak is to be detected. |
| GetMinArea | Minimum area value. |
| GetMinNumFitSamples | Returns the minimum number of samples required for fitting on each side of the shape. |
| GetNumFilteringPasses | Number of filtering passes for a model fitting operation. |
| GetNumMeasuredPoints | Number of edge-crossing points along the sample path of the gauge that was inspected with the last call to ELineGauge::MeasureSample. |
| GetNumSamples | Number of sampled points during the model fitting operation. |
| GetNumSkipRanges | Number of skip ranges in the gauge after a call to ELineGauge::AddSkipRange. |
| GetNumValidSamples | Number of valid sample points remaining after a model fitting operation. |
| GetRectangularSamplingArea | Flag indicating whether the sampling area remains a rectangle when rotated, instead of becoming a parallelogram. |

| | |
|---|---|
| GetSample | Allows to retrieve the sample points found along the line. |
| GetSamplingStep | Approximate distance between sampled points during a model fitting operation. |
| GetSkipRange | Allows to retrieve the start and end values of the skip range corresponding to the given index, if it is valid (i.e. if it has previously been created by a call to the ELineGauge::AddSkipRange method. |
| GetSmoothing | Number of pixels used for the low-pass filtering operation. |
| GetThickness | Number of parallel segments used to extract the data profile. |
| GetThreshold | Threshold level used to delimit significant peaks in the data profile. |
| GetTolerance | Searching area half thickness of the line fitting gauge. |
| GetTransitionChoice | Transition choice. |
| GetTransitionIndex | Index (from **0** on) of the transition to be retained when the transition choice parameter is set to ETransitionChoice_NthFromBegin or ETransitionChoice_NthFromEnd. |
| GetTransitionType | Transition type. |

| | |
|---|---|
| GetType | Shape type. |
| GetValid | Flag indicating if at least one valid transition has been found. |
| HitTest | Checks whether the cursor is positioned over a handle (**TRUE**) or not (**FALSE**). |
| Measure | Triggers the point location or the model fitting operation. |
| MeasureSample | Computes the sample points along the sample path whose index in the list is given by the **pathIndex** parameter. |
| MeasureWithoutFitting | Triggers the point location without line fitting operation. |
| operator= | Compares the instance with another ELineGauge object and returns TRUE if they are identical. |
| Plot | Draws the profile that is crossed by one of the sample paths of the gauge, as defined by EPlotItem. |
| PlotWithCurrentPen | Draws the profile that is crossed by one of the sample paths of the gauge, as defined by EPlotItem. |
| Process | Triggers the process pertaining to a shape or gauge and all the daughter gauges attach to it. |

| | |
|---|---|
| RemoveAllSkipRanges | Removes all the skip ranges previously created by a call to ELineGauge::AddSkipRange. |
| RemoveSkipRange | After a call to ELineGauge::AddSkipRange, removes the skip range with the given index. |
| SetActive | Sets the flag indicating whether the gauge is active or not. |
| SetClippingMode | Clipping mode, that allows to choose how the fitted segment length and center are computed. |
| SetFilteringThreshold | Relative filtering threshold, that is the fraction of the average distance between the sampled points and the fitted model above which a point is filtered out. |
| SetHVConstraint | Status of the restriction on the orientation of the point location gauge or model fitting sample paths. |
| SetKnownAngle | Flag indicating whether the slope of the line to be fitted is known or not. |
| SetLine | Sets the nominal position, length and rotation angle of the line fitting gauge, according to a known ELine object. |
| SetMinAmplitude | Offset added to the **Threshold** when a peak is to be detected. |

| | |
|---|---|
| SetMinArea | Minimum area value. |
| SetMinNumFitSamples | Sets the minimum number of samples required for fitting on each side of the shape. |
| SetNumFilteringPasses | Number of filtering passes for a model fitting operation. |
| SetRectangularSamplingArea | Flag indicating whether the sampling area remains a rectangle when rotated, instead of becoming a parallelogram. |
| SetSamplingStep | Approximate distance between sampled points during a model fitting operation. |
| SetSmoothing | Number of pixels used for the low-pass filtering operation. |
| SetThickness | Number of parallel segments used to extract the data profile. |
| SetThreshold | Threshold level used to delimit significant peaks in the data profile. |
| SetTolerance | Searching area half thickness of the line fitting gauge. |
| SetTransitionChoice | Transition choice. |

| | |
|---|---|
| SetTransitionIndex | Index (from **0** on) of the transition to be retained when the transition choice parameter is set to ETransitionChoice_NthFromBegin or ETransitionChoice_NthFromEnd. |
| SetTransitionType | Transition type. |

# ELineGauge::SetActive

Sets the flag indicating whether the gauge is active or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetActive(BOOL active)
```

## Remarks

When complex gauging is required, several gauges can be grouped together. Applying ELineGauge::Process to the mother gauge or shape triggers the measurement of the whole. Only the active gauges will participate in the process. By default, the gauge is active (**TRUE**).

# ELineGauge::AddSkipRange

Adds an item to the set of skip ranges and returns the index of the newly added range.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 AddSkipRange(
 const OEV_UINT32 start,
 const OEV_UINT32 end
 )
```

### Parameters

*start*

Beginning of the skip range.

*end*

End of the skip range.

### Remarks

The samples indices between start and end (including the boundaries) will be discarded during the measurement process. The ELineGauge::AddSkipRange method allows to define skip ranges in an ELineGauge. This means that, at measure time, samples belonging to these ranges will not be taken into account. A sample may belong to more than one skip range; to be discarded, a sample has to pertain to at least one range. Moreover, the skip ranges are allowed to overlap one another. The range is allowed to be reversed (i.e. end is not required to be greater than start). Also, start and end are not required to reference valid indices at the time of the call (i.e. the range may lie outside of the current return value for ELineGauge::NumSamples).

# ELineGauge::GetAverageDistance

Average distance between the sampled points and the fitted model.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetAverageDistance()
```

### Remarks

Irrelevant in case of a point location operation.

# ELineGauge::GetClippingMode

# ELineGauge::SetClippingMode

Clipping mode, that allows to choose how the fitted segment length and center are computed.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EClippingMode GetClippingMode() const
void SetClippingMode(Euresys::Open_eVision_2_10::EClippingMode clippingMode)
```

## Remarks

By default, the clipping mode is EClippingMode_CenteredNominal, which corresponds to the behavior appearing in Open eVision version 6.4 and before.

# ELineGauge::CopyTo

Copies all the data of the current ELineGauge object into another ELineGauge object, and returns it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ELineGauge* CopyTo(
 ELineGauge* other,
 BOOL recursive
 )
```

## Parameters

*other*

Pointer to the ELineGauge object in which the current ELineGauge object data have to be copied.

*recursive*

**TRUE** if the children gauges have to be copied as well, **FALSE** otherwise.

### Remarks

In case of a **NULL** pointer, a new ELineGauge object will be created and returned.

# ELineGauge::Drag

Moves a handle to a new position and updates the position parameters of the gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Cursor current X coordinate.

*y*

Cursor current Y coordinate.

# ELineGauge::Draw

Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
void Draw(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

*color*

The color in which to draw the overlay.

# ELineGauge::DrawWithCurrentPen

Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

# ELineGauge::ELineGauge

Constructs a line measurement context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ELineGauge(
 )
void ELineGauge(
 const ELineGauge& other
 )
```

### Parameters

*other*

Another ELineGauge object to be copied in the new ELineGauge object.

### Remarks

With the default constructor, all the parameters are initialized to their respective default values. With the copy constructor, the constructed line measurement context is based on a pre-existing ELineGauge object. The gauge children are also copied. Hierarchy copying through a copy constructor is always recursive. To disable this recursion, use instead the ELineGauge::CopyTo method.

# ELineGauge::GetFilteringThreshold

# ELineGauge::SetFilteringThreshold

Relative filtering threshold, that is the fraction of the average distance between the sampled points and the fitted model above which a point is filtered out.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetFilteringThreshold()
void SetFilteringThreshold(float f32FilteringThreshold)
```

### Remarks

Irrelevant in case of a point location operation. During a model fitting operation, the "filtering" process can be invoked to remove outliers, i.e. points that were located significantly far away from the fitted model (so that their position is dubious).

# ELineGauge::GetMeasuredPeak

Returns information pertaining to the default derivative peak, along one of the sample paths of the gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPeak GetMeasuredPeak(
 OEV_UINT32 index
 )
```

### Parameters

*index*

This argument must be left unchanged from its default value, i.e. **~0** (**= 0xFFFFFFFF**).

### Remarks

ELineGauge::GetMeasuredPeak returns the information about the derivative peak that meets the following two requirements: 1. It lies on the sample path inspected with the last call to ELineGauge::MeasureSample, and 1. It is associated with the edge-crossing point along the latter sample path that is selected by the transition choice parameter (cf. ELineGauge::TransitionChoice).

**Note.** For this method to succeed, it is necessary to previously call ELineGauge::MeasureSample.

# ELineGauge::GetMeasuredPoint

Returns the coordinates of a sample point, measured along one of the sample paths of the gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetMeasuredPoint(
 OEV_UINT32 index
 )
```

### Parameters

*index*

This argument must be left unchanged from its default value, i.e. **~0** (**= 0xFFFFFFFF**).

### Remarks

These coordinates pertain to the world space; they are expressed in the reference frame to which the current ELineGauge object belongs. The gauging process uses a list of sample points to find the shape position and size that best fit a given image. These sample points are measured along the sample paths defined by the gauge geometry. ELineGauge::GetMeasuredPoint returns the coordinates of the sample point that meets the following two requirements: 1. It lies on the sample path inspected with the last call to ELineGauge::MeasureSample, and 1. Among all the sample points along the latter sample path, it is the one selected by the ELineGauge::TransitionChoice property.

**Note.** For this method to succeed, it is necessary to previously call ELineGauge::MeasureSample.

# ELineGauge::GetMinNumFitSamples

Returns the minimum number of samples required for fitting on each side of the shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetMinNumFitSamples(
 OEV_INT32& side0,
 OEV_INT32& side1,
 OEV_INT32& side2,
 OEV_INT32& side3
 )
```

### Parameters

*side0*

Minimum number of samples on the top side of the rectangle.

*side1*

Minimum number of samples on the left side of the rectangle.

*side2*

Minimum number of samples on the bottom side of the rectangle.

*side3*

Minimum number of samples on the right side of the rectangle.

### Remarks

Irrelevant in case of a point location operation.

# ELineGauge::GetSample

Allows to retrieve the sample points found along the line.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetSample(
 EPoint& pt,
 OEV_UINT32 index
 )
void GetSample(
 ESamplePoint& pt,
 OEV_UINT32 index
 )
```

## Parameters

*pt*

EPoint structure that will contain the sample position.

*index*

The sample index

## Remarks

The method provides the sample point corresponding to the supplied index. The returned value corresponds to the sample validity.

# ELineGauge::GetSkipRange

Allows to retrieve the start and end values of the skip range corresponding to the given index, if it is valid (i.e. if it has previously been created by a call to the ELineGauge::AddSkipRange method.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetSkipRange(
 const OEV_UINT32 index,
 OEV_UINT32& start,
 OEV_UINT32& end
 )
```

### Parameters

*index*

Index of the skip range.

*start*

Beginning of the skip range.

*end*

End of the skip range.

### Remarks

Start is guaranteed to be smaller or equal to end.

# ELineGauge::HitTest

Checks whether the cursor is positioned over a handle (**TRUE**) or not (**FALSE**).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL HitTest(
 BOOL daughters
 )
```

### Parameters

*daughters*

**TRUE** if the daughters gauges handles have to be considered as well.

# ELineGauge::GetHVConstraint

# ELineGauge::SetHVConstraint

Status of the restriction on the orientation of the point location gauge or model fitting sample paths.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetHVConstraint()
void SetHVConstraint(BOOL HVConstraint)
```

## Remarks

*Sample paths* are the point location gauges placed along the model to be fitted.

# ELineGauge::GetKnownAngle

# ELineGauge::SetKnownAngle

Flag indicating whether the slope of the line to be fitted is known or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetKnownAngle()
void SetKnownAngle(BOOL bKnownAngle)
```

### Remarks

A line model to be fitted may have a well-known slope. It is possible to impose the value of this slope, thus removing one degree of freedom. The line fitting gauge slope is set by means of ELineGauge. The sign of the rotation angle depends whether the field of view is calibrated or not. * When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value. * When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# ELineGauge::SetLine

Sets the nominal position, length and rotation angle of the line fitting gauge, according to a known ELine object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetLine(const ELine& line)
```

# ELineGauge::Measure

Triggers the point location or the model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Measure(
 EROIBW8* sourceImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image.

### Remarks

When this method is called, and if not enough valid points are detected, then the method returns directly, and the measured gauge is set to the nominal parameters.

# ELineGauge::GetMeasuredLine

Information pertaining to the fitted line.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ELine GetMeasuredLine()
```

# ELineGauge::MeasureSample

Computes the sample points along the sample path whose index in the list is given by the **pathIndex** parameter.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void MeasureSample(
 EROIBW8* sourceImage,
 OEV_UINT32 pathIndex
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*pathIndex*

Sample path index.

### Remarks

This method stores its results into a temporary variable inside the ELineGauge object.

# ELineGauge::MeasureWithoutFitting

Triggers the point location without line fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void MeasureWithoutFitting(
 EROIBW8* sourceImage
 )
```

### Parameters

*sourceImage*

Source image.

### Remarks

This method performs the actual measurement for each transition, but does not perform the line fitting. This means that individual samples will be available through the ELineGauge::GetSample method, but the gauge position will not be changed. Please note that the filtering will not be performed in this method, since it relies upon the fitting process. The filtering parameters will thus be unused.

# ELineGauge::GetMinAmplitude

# ELineGauge::SetMinAmplitude

Offset added to the **Threshold** when a peak is to be detected.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetMinAmplitude()
void SetMinAmplitude(OEV_UINT32 un32MinAmplitude)
```

### Remarks

When analyzing a derivative profile, a peak is made up of consecutive pixel values above **Threshold**. To detect weak [strong] transitions, lower [raise] the **Threshold** value. To avoid interference of noise, an additional parameter is provided. The **MinAmplitude** parameter is an offset added to **Threshold** when a peak is to be detected. When the pixel values of the derivative profile do not reach **Threshold + MinAmplitude**, the peak is not taken into account. Anyway, when a peak is taken into account, all the pixels with values above **Threshold** are considered (for more accuracy). Setting the **MinAmplitude** value to **0** merely cancels its effect.

# ELineGauge::GetMinArea

# ELineGauge::SetMinArea

Minimum area value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetMinArea()
void SetMinArea(OEV_UINT32 un32MinArea)
```

### Remarks

A transition is detected if its derivative peak reaches **Threshold** + **MinAmplitude** value, and then declared valid if the area between the peak curve and the horizontal at level **Threshold** reaches the **MinArea** value.

# ELineGauge::GetNumFilteringPasses

# ELineGauge::SetNumFilteringPasses

Number of filtering passes for a model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumFilteringPasses()
void SetNumFilteringPasses(OEV_UINT32 un32NumFilteringPasses)
```

## Remarks

Irrelevant in case of a point location operation. During a filtering pass, the points that are too distant from the model are discarded. During a model fitting operation, the "filtering" process can be invoked to remove outliers, i.e. points that were located significantly far away from the fitted model (so that their position is dubious). By default (the number of filtering passes is 0), the outliers rejection process is disabled.

# ELineGauge::GetNumMeasuredPoints

Number of edge-crossing points along the sample path of the gauge that was inspected with the last call to ELineGauge::MeasureSample.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumMeasuredPoints()
```

## Remarks

**Note.** For this method to succeed, it is necessary to previously call ELineGauge::MeasureSample.

# ELineGauge::GetNumSamples

Number of sampled points during the model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumSamples() const
```

## Remarks

Irrelevant in case of a point location operation. After a model fitting operation, a number of points have been fitted along the model. Among them, some are not reliable because of their **Area** value. Among the remaining ones, some are filtered out (**NumFilteringPasses**, **FilteringThreshold**).

# ELineGauge::GetNumSkipRanges

Number of skip ranges in the gauge after a call to ELineGauge::AddSkipRange.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumSkipRanges() const
```

# ELineGauge::GetNumValidSamples

Number of valid sample points remaining after a model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumValidSamples()
```

### Remarks

Irrelevant in case of a point location operation. After a model fitting operation, a number of points have been fitted along the model. Among them, some are not reliable because of their **Area** value. Among the remaining ones, some are filtered out (**NumFilteringPasses**, **FilteringThreshold**).

# ELineGauge::operator=

Compares the instance with another ELineGauge object and returns TRUE if they are identical.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ELineGauge& operator=(
 const ELineGauge& other
 )
```

### Parameters

*other*

ELineGauge object to be compared

# ELineGauge::Plot

Draws the profile that is crossed by one of the sample paths of the gauge, as defined by EPlotItem.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Plot(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )

void Plot(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )

void Plot(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawItems*

Boolean combination of EPlotItem members, that indicates which items must be displayed.

*width*

Width of the plot.

*height*

Height of the plot.

*originX*

Origin point coordinates of the plot along the X axis.

*originY*

Origin point coordinates of the plot along the Y axis.

*color*

The color in which to draw the overlay.

### Remarks

The sample path that is taken into considered is the one inspected with the last call to ELineGauge::MeasureSample.

**Note.** For this method to succeed, it is necessary to previously call ELineGauge::MeasureSample.

# ELineGauge::PlotWithCurrentPen

Draws the profile that is crossed by one of the sample paths of the gauge, as defined by EPlotItem.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void PlotWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawItems*

Boolean combination of EPlotItem members, that indicates which items must be displayed.

*width*

Width of the plot.

*height*

Height of the plot.

*originX*

Origin point coordinates of the plot along the X axis.

*originY*

Origin point coordinates of the plot along the Y axis.

### Remarks

The sample path that is taken into considered is the one inspected with the last call to ELineGauge::MeasureSample.

**Note.** For this method to succeed, it is necessary to previously call ELineGauge::MeasureSample.

# ELineGauge::Process

Triggers the process pertaining to a shape or gauge and all the daughter gauges attach to it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Process(
 EROIBW8* sourceImage,
 BOOL daughters
 )
```

### Parameters

*sourceImage*

Pointer to the BW8 roi source image.

*daughters*

Flag indicating whether the daughters shapes inherit of the same behavior.

### Remarks

When complex gauging is required, several gauges can be grouped together. Applying **Process** to the mother gauge or shape triggers the measurement of the whole. Only the active gauges will participate in the process.

# ELineGauge::GetRectangularSamplingArea

# ELineGauge::SetRectangularSamplingArea

Flag indicating whether the sampling area remains a rectangle when rotated, instead of becoming a parallelogram.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetRectangularSamplingArea()
void SetRectangularSamplingArea(BOOL bRectangularSamplingArea)
```

### Remarks

By default, this flag is set to **TRUE**: the sampling area always remains a rectangle. Setting this property is only useful when the thickness transition parameter is greater than 1. In fact, when thickness transition parameter is equal to 1, rectangle and parallelogram reduce to the same segment.

# ELineGauge::RemoveAllSkipRanges

Removes all the skip ranges previously created by a call to ELineGauge::AddSkipRange.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveAllSkipRanges(
 )
```

# ELineGauge::RemoveSkipRange

After a call to ELineGauge::AddSkipRange, removes the skip range with the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveSkipRange(
 const OEV_UINT32 index
 )
```

### Parameters

*index*

Index of the skip range to remove, as returned by ELineGauge::AddSkipRange.

# ELineGauge::GetSamplingStep

# ELineGauge::SetSamplingStep

Approximate distance between sampled points during a model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSamplingStep()
void SetSamplingStep(float f32SamplingStep)
```

### Remarks

Irrelevant in case of a point location operation. To fit a model, a series of point location operations are performed along the model. The point location gauges spacing is given by the sampling step. By default, the sampling step is set to **5**, which means 5 pixels when the field of view is not calibrated, and 5 physical units in case of a calibrated field of view. Be aware that if the sampling step is too large, the number of sampled points along the model will not be sufficient enough to accurately locate it.

# ELineGauge::SetMinNumFitSamples

Sets the minimum number of samples required for fitting on each side of the shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetMinNumFitSamples(
 OEV_INT32 side0,
 OEV_INT32 side1,
 OEV_INT32 side2,
 OEV_INT32 side3
 )
```

## Parameters

*side0*

Required number of samples to correctly fit the line. The default value is **2**. It is the only parameter taken into account.

*side1*

Not used.

*side2*

Not used.

*side3*

Not used.

## Remarks

Irrelevant in case of a point location operation. When the ELineGauge::Measure method is called, and if not enough valid points are detected, then the method returns directly, and the measured gauge is set to the nominal parameters.

# ELineGauge::GetSmoothing

# ELineGauge::SetSmoothing

Number of pixels used for the low-pass filtering operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetSmoothing()
void SetSmoothing(OEV_UINT32 smoothing)
```

### Remarks

To reduce the effect of noise, the profile data can be low-pass filtered along the point location gauge direction.

# ELineGauge::GetThickness

# ELineGauge::SetThickness

Number of parallel segments used to extract the data profile.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetThickness()
void SetThickness(OEV_UINT32 thickness)
```

### Remarks

To reduce the effect of noise and/or strengthen a transition, several parallel profiles can be accumulated.

# ELineGauge::GetThreshold

# ELineGauge::SetThreshold

Threshold level used to delimit significant peaks in the data profile.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetThreshold()
void SetThreshold(OEV_UINT32 threshold)
```

## Remarks

When analyzing a derivative profile, a peak is made up of consecutive pixel values above **Threshold**. To detect weak [strong] transitions, lower [raise] the **Threshold** value. To avoid interference of noise, an additional parameter is provided. The **MinAmplitude** parameter is an offset added to **Threshold** when a peak is to be detected. When the pixel values of the derivative profile do not reach **Threshold + MinAmplitude**, the peak is not taken into account. Anyway, when a peak is taken into account, all the pixels with values above **Threshold** are considered (for more accuracy). Setting the **MinAmplitude** value to **0** merely cancels its effect.

# ELineGauge::GetTolerance

# ELineGauge::SetTolerance

Searching area half thickness of the line fitting gauge.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetTolerance()
void SetTolerance(float tolerance)
```

### Remarks

By default, the searching area thickness of the line fitting gauge is **20** (2x10), which means 20 pixels when the field of view is not calibrated, and 20 physical units in case of a calibrated field of view.

# ELineGauge::GetTransitionChoice

# ELineGauge::SetTransitionChoice

Transition choice.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::ETransitionChoice GetTransitionChoice()
void SetTransitionChoice(Euresys::Open_eVision_2_10::ETransitionChoice transitionChoice)
```

### Remarks

Several peaks may be detected along a point location gauge. This parameter helps to select the desired transition. In case of ETransitionChoice_NthFromBegin or ETransitionChoice_NthFromEnd transition choice, set ELineGauge::TransitionIndex to specify the desired transition. By default, the selected transition corresponds to the one with the largest amplitude (ETransitionChoice_LargestAmplitude).

# ELineGauge::GetTransitionIndex

# ELineGauge::SetTransitionIndex

Index (from **0** on) of the transition to be retained when the transition choice parameter is set to NthFromBegin or NthFromEnd.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetTransitionIndex()
void SetTransitionIndex(OEV_UINT32 transitionIndex)
```

## Remarks

Several peaks may be detected along a point location gauge. This parameter helps to select the desired transition. By default, the first transition is retained (the index value is **0**).

# ELineGauge::GetTransitionType

# ELineGauge::SetTransitionType

Transition type.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::ETransitionType GetTransitionType()
void SetTransitionType(Euresys::Open_eVision_2_10::ETransitionType transitionType)
```

### Remarks

The type of a transition tells whether it crosses increasing or decreasing gray-level values. This helps discriminate between nearby edges of an object. By default, the searched transition type is indifferently a black to white or a white to black transition (ETransitionType_BwOrWb).

# ELineGauge::GetType

Shape type.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EShapeType GetType()
```

# ELineGauge::GetValid

Flag indicating if at least one valid transition has been found.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetValid()
```

### Remarks

A **FALSE** value means that no measurement has been performed. A **TRUE** value means that a transition was found along the sample path inspected with the last call to ELineGauge::MeasureSample, and thus a point has been measured.

# 4.105. ELineShape Class

Manages a line shape.

**Base Class:** EShape
**Derived Class(es):** ELineGauge
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Closest | Find the daughter shape that is the closest to this shape. To retrieve the closest shape, use EShape::ClosestShape. |
| CopyTo | Copies all the data of the current ELineShape object into another ELineShape object and returns it. |
| Drag | Moves a handle to a new position and updates the position parameters of the shape. |
| Draw | Draws a graphical representation of a shape, as defined by EDrawingMode. |
| DrawWithCurrentPen | Draws a graphical representation of a shape, as defined by EDrawingMode. |
| GetAngle | Orientation of the shape. |

| | |
|---|---|
| GetCenter | Center point of the frame. |
| GetCenterX | Abscissa of the origin point of the frame. |
| GetCenterY | Ordinate of the origin point of the frame. |
| GetEnd | End point coordinates of the ELineShape object. |
| GetLength | Length of the ELineShape object. |
| GetOrg | Origin point coordinates of the ELineShape object. |
| GetPoint | Returns the coordinates of a point along the line. |
| GetScale | Horizontal sensor resolution, in pixels per unit. |
| GetType | Shape type. |
| HitTest | Checks if there is a handle under the cursor. |
| operator= | Assignment operator |
| SetAngle | Orientation of the shape. |

| | |
|---|---|
| SetCenter | Center point of the frame. |
| SetCenterXY | Sets the center coordinates of a ELineShape object. |
| SetFromOriginAndEnd | Sets the geometric parameters (center coordinates, length, and rotation angle) of a ELineShape object. |
| SetFromTwoPoints | DEPRECATED (you should use ELineShape::SetFromOriginAndEnd) : Sets the geometric parameters (center coordinates, length, and rotation angle) of a ELineShape object. |
| SetLength | Length of the ELineShape object. |
| SetLine | Sets the nominal position, length and rotation angle of the line, according to a known ELine object. |
| SetScale | Horizontal sensor resolution, in pixels per unit. |

# ELineShape::GetAngle

# ELineShape::SetAngle

Orientation of the shape.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetAngle() const
void SetAngle(float f32Angle)
```

# ELineShape::GetCenter

# ELineShape::SetCenter

Center point of the frame.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetCenter() const
void SetCenter(const EPoint& center)
```

# ELineShape::GetCenterX

Abscissa of the origin point of the frame.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetCenterX() const
```

# ELineShape::GetCenterY

Ordinate of the origin point of the frame.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCenterY() const
```

# ELineShape::Closest

Find the daughter shape that is the closest to this shape. To retrieve the closest shape, use EShape::ClosestShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Closest(
 )
```

# ELineShape::CopyTo

Copies all the data of the current ELineShape object into another ELineShape object and returns it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ELineShape* CopyTo(
 ELineShape* dest,
 BOOL bRecursive
 )
```

### Parameters

*dest*

Pointer to the ELineShape object in which the current ELineShape object data have to be copied.

*bRecursive*

**TRUE** if the children shapes have to be copied as well, **FALSE** otherwise.

### Remarks

In case of a **NULL** pointer, a new ELineShape object will be created and returned.

# ELineShape::Drag

Moves a handle to a new position and updates the position parameters of the shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 OEV_INT32 n32CursorX,
 OEV_INT32 n32CursorY
 )
```

### Parameters

*n32CursorX*

Current cursor coordinates.

*n32CursorY*

Current cursor coordinates.

# ELineShape::Draw

Draws a graphical representation of a shape, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
void Draw(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

*color*

-

# ELineShape::DrawWithCurrentPen

Draws a graphical representation of a shape, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

# ELineShape::GetEnd

End point coordinates of the ELineShape object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetEnd()
```

# ELineShape::GetPoint

Returns the coordinates of a point along the line.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetPoint(
 float fraction
 )
```

### Parameters

*fraction*

Point location expressed as a fraction of the line length (range **[-1, +1]**).

# ELineShape::HitTest

Checks if there is a handle under the cursor.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL HitTest(
 BOOL bDaughters
 )
```

### Parameters

*bDaughters*

Indicates if the check must be done in the whole hierarchy or just this object.

# ELineShape::GetLength

# ELineShape::SetLength

Length of the ELineShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetLength()
void SetLength(float length)
```

## Remarks

By default, the length of the line is **100**, which means 100 pixels when the field of view is not calibrated, and 100 physical units in case of a calibrated field of view.

# ELineShape::SetLine

Sets the nominal position, length and rotation angle of the line, according to a known ELine object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetLine(const ELine& line)
```

# ELineShape::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
ELineShape& operator=(
 const ELineShape& other
 )
```

### Parameters

*other*

Reference to the ELineShape object used for the assignment

# ELineShape::GetOrg

Origin point coordinates of the ELineShape object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetOrg()
```

# ELineShape::GetScale

# ELineShape::SetScale

Horizontal sensor resolution, in pixels per unit.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScale() const
void SetScale(float f32Scale)
```

# ELineShape::SetCenterXY

Sets the center coordinates of a ELineShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetCenterXY(
 float centerX,
 float centerY
 )
```

## Parameters

*centerX*

Center coordinates of the ELineShape object.

*centerY*

Center coordinates of the ELineShape object.

# ELineShape::SetFromOriginAndEnd

Sets the geometric parameters (center coordinates, length, and rotation angle) of a ELineShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromOriginAndEnd(
 const EPoint& origin,
 const EPoint& end
 )
```

### Parameters

*origin*

Origin point coordinates of the line.

*end*

End point coordinates of the line.

# ELineShape::SetFromTwoPoints

DEPRECATED (you should use ELineShape::SetFromOriginAndEnd) : Sets the geometric parameters (center coordinates, length, and rotation angle) of a ELineShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromTwoPoints(
 const EPoint& origin,
 const EPoint& end
 )
```

### Parameters

*origin*

Origin point coordinates of the line.

*end*

End point coordinates of the line.

## ELineShape::GetType

Shape type.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EShapeType GetType()
```

# 4.106. EListItem Class

Describes list items. This class pertains to the EasyObject legacy API. Please use ECodedImage2 for all new developments instead.

### Remarks

A list is a sequence of orderly list items. Each list item contains a pointer to a memory zone containing its data, a pointer to the previous list item, and a pointer to the next list item. List itemsA few ECodedImage methods handle EListItem objects, or EListItem pointers. Runs listsECodedImage::GetFirstRunData, ECodedImage::GetFirstRunPtr, ECodedImage::GetLastRunData, ECodedImage::GetLastRunPtr, ECodedImage::GetPreviousRunData, ECodedImage::GetPreviousRunPtr, ECodedImage::GetNextRunData, ECodedImage::GetNextRunPtr These properties and methods allow to traverse the runs lists from the first run to the last, or from one run to its previous or next neighbor. A run can also be directly reached by its index within the list. The first run has index **0**. The last run has index **NumRuns-1**. The ECodedImage::GetRunData and ECodedImage::GetRunDataPtr methods return the run data, or a pointer to the run data. Objects listsECodedImage::GetFirstObjData, ECodedImage::GetLastObjData, ECodedImage::GetPreviousObjData, ECodedImage::GetPreviousObjPtr, ECodedImage::GetNextObjData, ECodedImage::GetNextObjPtr These properties and methods allow to traverse the objects lists from the first object to the last, or from one object to its previous or next neighbor. An object can also be directly reached by its index within the list. The first object has index **0**. The last object has index **NumObjects-1**. The ECodedImage::GetObjectData and ECodedImage::GetObjDataPtr methods return the object data, or a pointer to the object data.

**Namespace:** Euresys::Open_eVision_2_10

# 4.107. EMailBarcode Class

Manages a complete context for a Mail Barcode.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Draw | Draws the bounding box of the mail barcode. |
| EMailBarcode | Constructs an EMailBarcodeReader context. |
| GetChecksumOk | Returns if the checksum was sucessfully validated. |

| | |
|---|---|
| GetComponentStrings | Returns the list of semantic parts included in the mail barcode text. |
| GetOrientation | Returns the orientation of the mail barcode. |
| GetPosition | Returns the position of the mail barcode in the Image/ROI. |
| GetSymbology | Returns the symbology of the mail barcode. |
| GetText | Returns the full decoded text of the mail barcode. |
| operator= | Assignment operator |

# EMailBarcode::GetChecksumOk

Returns if the checksum was sucessfully validated.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
bool GetChecksumOk() const
```

# EMailBarcode::GetComponentStrings

Returns the list of semantic parts included in the mail barcode text.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::vector<Euresys::Open_eVision_2_10::EStringPair> GetComponentStrings() const
```

# EMailBarcode::Draw

Draws the bounding box of the mail barcode.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Draw(
 HDC hDC,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
void Draw(
 EDrawAdapter* adapter,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*hDC*

Handle of the device context on which to draw.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*adapter*

-

# EMailBarcode::EMailBarcode

Constructs an EMailBarcodeReader context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EMailBarcode(
 )
void EMailBarcode(
 const EMailBarcode& other
 )
```

### Parameters

*other*

-

# EMailBarcode::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EMailBarcode& operator=(
 const EMailBarcode& other
 )
```

### Parameters

*other*

-

# EMailBarcode::GetOrientation

Returns the orientation of the mail barcode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EMailBarcodeOrientation GetOrientation() const
```

# EMailBarcode::GetPosition

Returns the position of the mail barcode in the Image/ROI.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
ERectangle GetPosition() const
```

## EMailBarcode::GetSymbology

Returns the symbology of the mail barcode.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::EMailBarcodeSymbologies GetSymbology() const
```

## EMailBarcode::GetText

Returns the full decoded text of the mail barcode.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::string GetText() const
```

# 4.108. EMailBarcodeReader Class

Manages a complete context for a Mail Barcode Reader.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EMailBarcodeReader | Constructs an EMailBarcodeReader context. |
| GetEnableClutteredBarcodes | Enables cluttered barcode (barcode with fused bars) support. |
| GetEnableDottedBarcodes | Enables dotted barcode support. |
| GetExpectedOrientations | Expected barcode orientations for the Mail Barcode detection. |
| GetExpectedSymbologies | Expected symbologies for the Mail Barcode detection. |
| GetValidateChecksum | Configures the reader to return barcodes even if their checksum is incorrect. |
| Load | Loads the settings of the EMailBarcodeReader object, from disk. |
| operator= | Assignment operator |
| Read | Locates and decodes mail barcodes. |
| Save | Saves the current settings of the EMailBarcodeReader object. |
| SetEnableClutteredBarcodes | Enables cluttered barcode (barcode with fused bars) support. |

| | |
|---|---|
| SetEnableDottedBarcodes | Enables dotted barcode support. |
| SetExpectedOrientations | Expected barcode orientations for the Mail Barcode detection. |
| SetExpectedSymbologies | Expected symbologies for the Mail Barcode detection. |
| SetValidateChecksum | Configures the reader to return barcodes even if their checksum is incorrect. |

# MailBarcodeReader::EMailBarcodeReader

Constructs an EMailBarcodeReader context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EMailBarcodeReader(
 )
void EMailBarcodeReader(
 const EMailBarcodeReader& other
 )
```

## Parameters

*other*

-

# EMailBarcodeReader::GetEnableClutteredBarcodes

# EMailBarcodeReader::SetEnableClutteredBarcodes

Enables cluttered barcode (barcode with fused bars) support.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool GetEnableClutteredBarcodes() const
void SetEnableClutteredBarcodes(bool enable)
```

# EMailBarcodeReader::GetEnableDottedBarcodes

# EMailBarcodeReader::SetEnableDottedBarcodes

Enables dotted barcode support.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool GetEnableDottedBarcodes() const
void SetEnableDottedBarcodes(bool enable)
```

# EMailBarcodeReader::GetExpectedOrientations

# EMailBarcodeReader::SetExpectedOrientations

Expected barcode orientations for the Mail Barcode detection.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetExpectedOrientations() const
void SetExpectedOrientations(int orientations)
```

## Remarks

The value is a combination of the members of the EMailBarcodeOrientation enumerate.

# EMailBarcodeReader::GetExpectedSymbologies

# EMailBarcodeReader::SetExpectedSymbologies

Expected symbologies for the Mail Barcode detection.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetExpectedSymbologies() const
void SetExpectedSymbologies(int symbologies)
```

### Remarks

The value is a combination of the members of the EMailBarcodeSymbologies enumerate.

# EMailBarcodeReader::Load

Loads the settings of the EMailBarcodeReader object, from disk.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Load(
 ESerializer* path
 )
```

### Parameters

*path*

A string containing the full path to the file.

# EMailBarcodeReader::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EMailBarcodeReader& operator=(
 const EMailBarcodeReader& other
 )
```

### Parameters

*other*

-

# EMailBarcodeReader::Read

Locates and decodes mail barcodes.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::vector<Euresys::Open_eVision_2_10::EMailBarcode> Read(
 const EROIBW8& roi
 )
```

## Parameters

*roi*

The ROI/Image in which to search for mail barcodes.

## Remarks

This method returns the list of the detected barcodes.

# EMailBarcodeReader::Save

Saves the current settings of the EMailBarcodeReader object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Save(
 ESerializer* path
 )
```

### Parameters

*path*

A string containing the full path to the file.

### Remarks

It is advised to use a file extension that is non-standard (for instance *.mbr).

## EMailBarcodeReader::GetValidateChecksum

## EMailBarcodeReader::SetValidateChecksum

Configures the reader to return barcodes even if their checksum is incorrect.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool GetValidateChecksum() const
void SetValidateChecksum(bool validateChecksum)
```

# 4.109. EMatcher Class

Manages a complete matching context in EasyMatch.

### Remarks

A matching context consists of a learned pattern and of the parameters required to locate one or more instances of the pattern in a search field.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| ClearImage | Releases the pointer to the image object that has been passed to the EMatcher object. |
| CopyLearntPattern | Copies the learnt pattern in the supplied image. If no pattern has been learned, an exception with code EError_NoPatternLearnt will be thrown. |
| CopyTo | Copies all the data of the current EMatcher object into another EMatcher object and returns it. |
| DrawPosition | Draws a graphical representation of a given occurrence of the pattern in the image, using a rectangle and possibly a small line segment in the upper-left corner. |
| DrawPositions | Draws a graphical representation of all occurrences of the pattern in the image, using a rectangle and possibly a small line segment in the upper-left corner. |
| DrawPositionsWithCurrentPen | Draws a graphical representation of all occurrences of the pattern in the image, using a rectangle and possibly a small line segment in the upper-left corner. |
| DrawPositionWithCurrentPen | Draws a graphical representation of a given occurrence of the pattern in the image, using a rectangle and possibly a small line segment in the upper-left corner. |
| EMatcher | Constructs a matching context. |

| | |
|---|---|
| GetAdvancedLearning | Toggle advanced learning. |
| GetAngleStep | Current angle step. |
| GetContrastMode | Contrast mode. |
| GetCorrelationMode | Correlation mode. |
| GetDontCareThreshold | "Don't care" threshold. |
| GetFilteringMode | Filtering mode. |
| GetFinalReduction | Index of the last reduction. |
| GetInitialMinScore | Minimum score applied as a selection criterion in the early stages of the matching process. |
| GetInterpolate | Interpolation mode. |
| GetIsotropicScale | Flag indicating whether isotropic (as opposed to anisotropic) scaling is used. |
| GetMaxAngle | Maximum angle, in the current angle unit. |

| | |
|---|---|
| GetMaxInitialPositions | Maximum number of positions at the first stage of the matching process. |
| GetMaxPositions | Maximum number of positions. |
| GetMaxScale | Maximum scale factor for isotropic scaling. |
| GetMaxScaleX | Maximum horizontal scale factor for anisotropic scaling. |
| GetMaxScaleY | Maximum vertical scale factor for anisotropic scaling. |
| GetMinAngle | Minimum angle, in the current angle unit. |
| GetMinReducedArea | Minimum reduced area parameter. |
| GetMinScale | Minimum scale factor for isotropic scaling. |
| GetMinScaleX | Minimum horizontal scale factor for anisotropic scaling. |
| GetMinScaleY | Minimum vertical scale factor for anisotropic scaling. |
| GetMinScore | Minimum score. |

| | |
|---|---|
| GetNumPositions | Number of good matches found, as defined by EMatcher::MinScore and EMatcher::MaxPositions properties. |
| GetNumReductions | Number of reduction steps used in the matching process. |
| GetPatternHeight | Learnt pattern height. |
| GetPatternLearnt | Returns **TRUE** after a learning operation has been successfully performed, indicating that the EMatcher object is ready for matching, and **FALSE** otherwise. |
| GetPatternType | Pixel type of the learnt pattern. |
| GetPatternWidth | Learnt pattern width. |
| GetPixelDimensions | Gets the physical pixel dimensions. |
| GetPosition | Returns an EMatchPosition object containing the position coordinates. |
| GetPositions | Returns a vector of EMatchPosition objects, each containing the position coordinates and other matching results. |
| GetScaleStep | Current value of scale step. |

| | |
|---|---|
| GetScaleXStep | Current value of scale X step. |
| GetScaleYStep | Current value of scale Y step. |
| GetVersion | Version number of the EMatcher object. |
| LearnPattern | Learns a pattern to subsequently match in an image. |
| Load | Loads the EMatcher. The given ESerializer must have been created for reading. |
| Match | Matches the pattern against an image. |
| operator= | Copies all the data from another EMatcher object into the current EMatcher object |
| Save | Saves the EMatcher. The given ESerializer must have been created for writing. |
| SetAdvancedLearning | Toggle advanced learning. |
| SetContrastMode | Contrast mode. |
| SetCorrelationMode | Correlation mode. |

| | |
|---|---|
| SetDontCareThreshold | "Don't care" threshold. |
| SetExtension | Sets the extension of the matching ROI, i.e. puts the horizontal and vertical distances, in pixels, that the found pattern occurrences may fall outside the matching ROI. Such occurrences partially outside the ROI have their score corrected by the ratio between the pattern area outside the ROI and the pattern area inside. |
| SetFilteringMode | Filtering mode. |
| SetFinalReduction | Index of the last reduction. |
| SetInitialMinScore | Minimum score applied as a selection criterion in the early stages of the matching process. |
| SetInterpolate | Interpolation mode. |
| SetMaxAngle | Maximum angle, in the current angle unit. |
| SetMaxInitialPositions | Maximum number of positions at the first stage of the matching process. |
| SetMaxPositions | Maximum number of positions. |
| SetMaxScale | Maximum scale factor for isotropic scaling. |

| | |
|---|---|
| SetMaxScaleX | Maximum horizontal scale factor for anisotropic scaling. |
| SetMaxScaleY | Maximum vertical scale factor for anisotropic scaling. |
| SetMinAngle | Minimum angle, in the current angle unit. |
| SetMinReducedArea | Minimum reduced area parameter. |
| SetMinScale | Minimum scale factor for isotropic scaling. |
| SetMinScaleX | Minimum horizontal scale factor for anisotropic scaling. |
| SetMinScaleY | Minimum vertical scale factor for anisotropic scaling. |
| SetMinScore | Minimum score. |
| SetPixelDimensions | Sets the physical pixel dimensions. |

# EMatcher::GetAdvancedLearning

# EMatcher::SetAdvancedLearning

Toggle advanced learning.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetAdvancedLearning() const
void SetAdvancedLearning(BOOL bState)
```

### Remarks

When enable, the advanced learning process will try to optimize learning parameters like the Minimum Reduced Area. The learning will take more time (from 1x to 5x longer) but the matching probabilty could be improved. The improvement strongly depends on the pattern source image. The advanced learning is automatically disabled when the method EMatcher::MinReducedArea is called.

# EMatcher::GetAngleStep

Current angle step.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetAngleStep()
```

# EMatcher::ClearImage

Releases the pointer to the image object that has been passed to the EMatcher object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ClearImage(
 )
```

### Remarks

It is the way to tell to the EMatcher object that its pointer is not valid anymore. The EMatcher::Match method keeps a copy of the image pointer given as parameter. So, if the user deletes this pointer, the EMatcher object should be informed.

# EMatcher::GetContrastMode

# EMatcher::SetContrastMode

Contrast mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EMatchContrastMode GetContrastMode()
void SetContrastMode(Euresys::Open_eVision_2_10::EMatchContrastMode eMode)
```

### Remarks

By default, the contrast mode is set to EMatchContrastMode_Normal.

# EMatcher::CopyLearntPattern

Copies the learnt pattern in the supplied image. If no pattern has been learned, an exception with code NoPatternLearnt will be thrown.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void CopyLearntPattern(
 EImageBW8& image
 )
void CopyLearntPattern(
 EImageC24& image
 )
```

### Parameters

*image*

Pointer to the image in which the learnt pattern will be returned.

# EMatcher::CopyTo

Copies all the data of the current EMatcher object into another EMatcher object and returns it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EMatcher* CopyTo(
 EMatcher* other
 )
```

### Parameters

*other*

Pointer to the EMatcher object in which the current EMatcher object parameters are to be copied. If **NULL** (default), a new EMatcher object will be created and returned.

# EMatcher::GetCorrelationMode

# EMatcher::SetCorrelationMode

Correlation mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::ECorrelationMode GetCorrelationMode()
void SetCorrelationMode(Euresys::Open_eVision_2_10::ECorrelationMode eMode)
```

## Remarks

This property tells what normalization rule is used to correlate the pattern to the image. By default, the correlation mode is set to ECorrelationMode_Normalized.

# EMatcher::GetDontCareThreshold

# EMatcher::SetDontCareThreshold

"Don't care" threshold.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetDontCareThreshold()
void SetDontCareThreshold(OEV_UINT32 un32Threshold)
```

### Remarks

If the pattern cannot be inscribed in a rectangle because there are foreign objects in a close neighborhood, mismatches can be avoided by using "don't care" pixels: all the pattern pixels whose value is strictly below **DontCareThreshold** will be ignored. By default, this property is set to **0**: no "don't care" pixel exists.

**Note.** When you use the "don't care" feature, either the pattern is well contrasted from its background -then set **DontCareThreshold** to an appropriate thresholding value- or it is not -then set the background pixels of the pattern to some low value (by a masking operation) and set **DontCareThreshold** to this low value plus one.

# EMatcher::DrawPosition

Draws a graphical representation of a given occurrence of the pattern in the image, using a rectangle and possibly a small line segment in the upper-left corner.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawPosition(
 HDC graphicContext,
 OEV_UINT32 index,
 BOOL bCorner,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
void DrawPosition(
 HDC graphicContext,
 const ERGBColor& color,
 OEV_UINT32 index,
 BOOL bCorner,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void DrawPosition(
 EDrawAdapter* graphicContext,
 OEV_UINT32 index,
 BOOL bCorner,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*index*

Occurrence index, in range **0..NumPositions-1**.

*bCorner*

**TRUE** if the corner mark is to be drawn. **FALSE** by default. (This mark is useful when large rotations are allowed.)

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*color*

The color in which to draw the overlay.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used.

# EMatcher::DrawPositions

Draws a graphical representation of all occurrences of the pattern in the image, using a rectangle and possibly a small line segment in the upper-left corner.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawPositions(
 HDC graphicContext,
 BOOL bCorner,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
void DrawPositions(
 HDC graphicContext,
 const ERGBColor& color,
 BOOL bCorner,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
void DrawPositions(
 EDrawAdapter* graphicContext,
 BOOL bCorner,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*bCorner*

**TRUE** if the corner mark is to be drawn. **FALSE** by default. (This mark is useful when large rotations are allowed.)

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*color*

The color in which to draw the overlay.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used for all occurrences (to vary the colors, draw the objects separately using the EMatcher::DrawPosition method instead).

# EMatcher::DrawPositionsWithCurrentPen

Draws a graphical representation of all occurrences of the pattern in the image, using a rectangle and possibly a small line segment in the upper-left corner.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawPositionsWithCurrentPen(
 HDC graphicContext,
 BOOL bCorner,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*bCorner*

**TRUE** if the corner mark is to be drawn. **FALSE** by default. (This mark is useful when large rotations are allowed.)

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used for all occurrences (to vary the colors, draw the objects separately using the EMatcher::DrawPosition method instead).

# EMatcher::DrawPositionWithCurrentPen

Draws a graphical representation of a given occurrence of the pattern in the image, using a rectangle and possibly a small line segment in the upper-left corner.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawPositionWithCurrentPen(
 HDC graphicContext,
 OEV_UINT32 index,
 BOOL bCorner,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*index*

Occurrence index, in range **0..NumPositions-1**.

*bCorner*

**TRUE** if the corner mark is to be drawn. **FALSE** by default. (This mark is useful when large rotations are allowed.)

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used.

# EMatcher::EMatcher

Constructs a matching context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EMatcher(
 )
void EMatcher(
 OEV_UINT32 maxNumDOF
 )
void EMatcher(
 const EMatcher& other
 )
```

### Parameters

*maxNumDOF*

Maximum number of degrees of freedom (this number must be comprised between **2** and **5**).

*other*

Another EMatcher object to be copied in the new EMatcher object.

### Remarks

With the default constructor (no argument), all parameters are initialized to their respective default values. The copy constructor constructs a matching context based on a pre-existing EMatcher object. The last constructor constructs a matching context with a specified number of degrees of freedom. **maximumNumberOfDegreesOfFreedom** is the maximum number of degrees of freedom that the EMatcher object being constructed will be allowed to use during its life. All other parameters are initialized to their respective default values. The degrees of freedom for a matching operation are the *translation* (2 modes), the *rotation* (1 mode), the *isotropic scaling* (1 mode) and the *anisotropic scaling* (1 more mode). There is no way to modify this parameter for an existing EMatcher object. The default value for **maximumNumberOfDegreesOfFreedom** is **5**, that is the maximum value. The minimum value for the number of degrees of freedom is **2**, in order to allow, at least, the pattern translation.

# EMatcher::GetFilteringMode

# EMatcher::SetFilteringMode

Filtering mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EFilteringMode GetFilteringMode()
void SetFilteringMode(Euresys::Open_eVision_2_10::EFilteringMode eFilteringMode)
```

### Remarks

To achieve acceptable time performance, EasyMatch works by sub-sampling the pattern in the early phases of the processing. The filtering mode parameter allows to select the pre-processing type applied to the image before the decimation: averaging or low-pass filtering. By default, this property is set to EFilteringMode_Uniform, whereas the EFilteringMode_LowPass mode is indicated if the image presents sharp gray-level transitions.

# EMatcher::GetFinalReduction

# EMatcher::SetFinalReduction

Index of the last reduction.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetFinalReduction()
void SetFinalReduction(OEV_UINT32 un32FinalReduction)
```

### Remarks

The pattern matching process is comprised of a few passes (typically 4) during which the position accuracy is improved by a factor of 2 (for all degrees of freedom). This is called a *reduction*. By default, the computation continues until an accuracy of one pixel is obtained. To speed up the process, the last reduction passes can be dropped, so lowering the accuracy. Anyway, the interpolation mode can then still be used. By default, this property is set to **0** (pixel accuracy). Value **1** corresponds to 2-pixels accuracy, **2** to 4, and so on. The range of values that can be used for this property is **0** to **NumReductions-1**.

# EMatcher::GetPixelDimensions

Gets the physical pixel dimensions.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void GetPixelDimensions(
 float& width,
 float& height
 )
```

### Parameters

*width*

Width of a pixel.

*height*

Height of a pixel.

# EMatcher::GetPosition

Returns an EMatchPosition object containing the position coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EMatchPosition GetPosition(
 OEV_UINT32 index
 )
```

### Parameters

*index*

0-based index to the desired position. The positions are ordered by decreasing score.

# EMatcher::GetInitialMinScore

# EMatcher::SetInitialMinScore

Minimum score applied as a selection criterion in the early stages of the matching process.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetInitialMinScore()
void SetInitialMinScore(float f32InitialMinScore)
```

## Remarks

When the search for matches starts, EasyMatch considers a set of candidate positions that it progressively refines. When they later appear to be bad candidates, they are rejected. Though it is the minimum score level that is used to reject bad matching positions at the final step of the matching process, the "initial minimum score" parameter is used to eliminate bad positions (whose score is not high enough) in the early stages of the matching processing. If the matching process is achieved in one step, only the minimum score parameter will be considered. By default, this property is set to **-1**.

# EMatcher::GetInterpolate

# EMatcher::SetInterpolate

Interpolation mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetInterpolate()
void SetInterpolate(BOOL bInterpolate)
```

### Remarks

By default, matching is done with a one-pixel precision for all degrees of freedom (translation, rotation and scaling). You can use an additional interpolation process to achieve sub-pixel accuracy. This generally leads to an improvement of the sub-pixel accuracy by a factor larger than 10. This is possible only when the found instances match closely the model. A score higher than 0.99 indicates that the instances are a close match of the model. In other words, the instance is considered to be more accurate when the score is higher. The added computational cost is low. By default, this property is set to **FALSE**.

# EMatcher::GetIsotropicScale

Flag indicating whether isotropic (as opposed to anisotropic) scaling is used.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetIsotropicScale()
```

### Remarks

**TRUE** if isotropic (as opposed to anisotropic) scaling is used, i.e. when the scale factors in both the X and Y directions are equal.

# EMatcher::LearnPattern

Learns a pattern to subsequently match in an image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void LearnPattern(
 EROIBW8* pattern
 )

void LearnPattern(
 EROIC24* pattern
 )
```

### Parameters

*pattern*

Pattern to learn.

### Remarks

The maximum size for a pattern is 1791x1791.

# EMatcher::Load

Loads the EMatcher. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# EMatcher::Match

Matches the pattern against an image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Match(
 EROIBW8* image
 )
void Match(
 EROIC24* image
 )
```

### Parameters

*image*

Pointer to the image/ROI within which the pattern will be searched for.

### Remarks

The matching results can be obtained by means of the EMatcher::NumPositions and EMatcher::GetPosition members.

# EMatcher::GetMaxAngle

# EMatcher::SetMaxAngle

Maximum angle, in the current angle unit.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
float GetMaxAngle()
void SetMaxAngle(float f32MaxAngle)
```

### Remarks

The rotation of the pattern is allowed within the range (**-1** <= **MinAngle** < **MaxAngle** <= **1** revolution). By default, both remain **0**.

# EMatcher::GetMaxInitialPositions

# EMatcher::SetMaxInitialPositions

Maximum number of positions at the first stage of the matching process.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetMaxInitialPositions()
void SetMaxInitialPositions(OEV_UINT32 un32MaxInitialPositions)
```

### Remarks

When the search for matches starts, EasyMatch considers a set of candidate positions that it progressively refines. When they later appear to be bad candidates, they are rejected. Eventually, a maximum of EMatcher::MaxPositions is returned. In some circumstances, when the image contains features roughly similar to the pattern, these can confuse the matching process, resulting in false matches. To overcome this situation, increasing the number of initial positions will help. By default, this property is set to **0**, indicating that the value of EMatcher::MaxPositions should be used instead.

# EMatcher::GetMaxPositions

# EMatcher::SetMaxPositions

Maximum number of positions.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetMaxPositions()
void SetMaxPositions(OEV_UINT32 un32MaxPositions)
```

## Remarks

Indicates how many matching positions have to be returned at a maximum. This number corresponds to the expected maximum number of occurrences of the pattern (it is sometimes advisable to use a few additional positions). By default, this property is set to **1**.

# EMatcher::GetMaxScale

# EMatcher::SetMaxScale

Maximum scale factor for isotropic scaling.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetMaxScale()
void SetMaxScale(float f32Scale)
```

### Remarks

Two scaling modes are allowed: in isotropic mode, the scale factor is identical in all directions; in anisotropic mode, the scale factors differ in the horizontal and vertical directions. To select the appropriate mode, it suffices to call the relevant **Set** member. The scaling of the pattern is allowed within the range (**0.5 <= MinScale < MaxScale <= 2**). By default, both remain **1**. The same holds for anisotropic scale factors.

# EMatcher::GetMaxScaleX

# EMatcher::SetMaxScaleX

Maximum horizontal scale factor for anisotropic scaling.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetMaxScaleX()
void SetMaxScaleX(float f32MaxScaleX)
```

### Remarks

Two scaling modes are allowed: in isotropic mode, the scale factor is identical in all directions; in anisotropic mode, the scale factors differ in the horizontal and vertical directions. To select the appropriate mode, it suffices to call the relevant **Set** member. The scaling of the pattern is allowed within the range (**0.5 <= MinScaleX < MaxScaleX <= 2**). By default, both remain **1**. The same holds for anisotropic scale factors.

# EMatcher::GetMaxScaleY

# EMatcher::SetMaxScaleY

Maximum vertical scale factor for anisotropic scaling.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetMaxScaleY()
void SetMaxScaleY(float f32MaxScaleY)
```

### Remarks

Two scaling modes are allowed: in isotropic mode, the scale factor is identical in all directions; in anisotropic mode, the scale factors differ in the horizontal and vertical directions. To select the appropriate mode, it suffices to call the relevant **Set** member. The scaling of the pattern is allowed within the range (**0.5** <= **MinScaleY** < **MaxScaleY** <= **2**). By default, both remain **1**. The same holds for anisotropic scale factors.

# EMatcher::GetMinAngle

# EMatcher::SetMinAngle

Minimum angle, in the current angle unit.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetMinAngle()
void SetMinAngle(float f32MinAngle)
```

### Remarks

The rotation of the pattern is allowed within the range (**-1** <= **MinAngle** < **MaxAngle** <= **1** revolution). By default, both remain **0**.

# EMatcher::GetMinReducedArea

# EMatcher::SetMinReducedArea

Minimum reduced area parameter.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetMinReducedArea()
void SetMinReducedArea(OEV_UINT32 un32Area)
```

## Remarks

To achieve acceptable time performance, EasyMatch works by under-sampling the pattern in the early phases of the processing. This property tells how many pixels of the pattern are kept, at a minimum. By default, this property is set to **64**, which is the right choice in most situations. Higher values are not recommended. Lower values can speed up the processing, but sometimes cause the matching process fail to find the best matches. To circumvent this problem, you can use guard positions, that is find more matches than expected and keep the good ones only.

**Note.** Changing this property invalidates any previous learning.

# EMatcher::GetMinScale

# EMatcher::SetMinScale

Minimum scale factor for isotropic scaling.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetMinScale()
void SetMinScale(float f32Scale)
```

### Remarks

Two scaling modes are allowed: in isotropic mode, the scale factor is identical in all directions; in anisotropic mode, the scale factors differ in the horizontal and vertical directions. To select the appropriate mode, it suffices to call the relevant **Set** member. The scaling of the pattern is allowed within the range (**0.5** <= **MinScale** < **MaxScale** <= **2**). By default, both remain **1**. The same holds for anisotropic scale factors.

# EMatcher::GetMinScaleX

# EMatcher::SetMinScaleX

Minimum horizontal scale factor for anisotropic scaling.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetMinScaleX()
void SetMinScaleX(float f32MinScaleX)
```

### Remarks

Two scaling modes are allowed: in isotropic mode, the scale factor is identical in all directions; in anisotropic mode, the scale factors differ in the horizontal and vertical directions. To select the appropriate mode, it suffices to call the relevant **Set** member. The scaling of the pattern is allowed within the range (**0.5** <= **MinScaleX** < **MaxScaleX** <= **2**). By default, both remain **1**. The same holds for anisotropic scale factors.

# EMatcher::GetMinScaleY

# EMatcher::SetMinScaleY

Minimum vertical scale factor for anisotropic scaling.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetMinScaleY()
void SetMinScaleY(float f32MinScaleY)
```

## Remarks

Two scaling modes are allowed: in isotropic mode, the scale factor is identical in all directions; in anisotropic mode, the scale factors differ in the horizontal and vertical directions. To select the appropriate mode, it suffices to call the relevant **Set** member. The scaling of the pattern is allowed within the range (**0.5 <= MinScaleY < MaxScaleY <= 2**). By default, both remain **1**. The same holds for anisotropic scale factors.

# EMatcher::GetMinScore

# EMatcher::SetMinScore

Minimum score.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetMinScore()
```

```
void SetMinScore(float f32MinScore)
```

### Remarks

This property indicates what score a match must reach to be considered as good, and to be returned in the list of positions; this selection criterion is applied at the final stage of the matching process. One good way to select the appropriate EMatcher::MinScore in a given context is to set it to **-1** (any match will be retained), set EMatcher::MaxPositions to more than needed, and examine the returned scores after a matching. By default, this property is set to **-1**.

# EMatcher::GetNumPositions

Number of good matches found, as defined by EMatcher::MinScore and EMatcher::MaxPositions properties.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumPositions()
```

# EMatcher::GetNumReductions

Number of reduction steps used in the matching process.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumReductions()
```

### Remarks

These depend on the actual pattern size, and on the **MinReducedArea** property. The **FinalReduction** property, used to speed up matching when coarse location is sufficient, must be set in range **0..NumReductions-1**.

# EMatcher::operator=

Copies all the data from another EMatcher object into the current EMatcher object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EMatcher& operator=(
 const EMatcher& other
 )
```

### Parameters

*other*

EMatcher object to be copied

# EMatcher::GetPatternHeight

Learnt pattern height.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetPatternHeight()
```

# EMatcher::GetPatternLearnt

Returns **TRUE** after a learning operation has been successfully performed, indicating that the EMatcher object is ready for matching, and **FALSE** otherwise.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetPatternLearnt() const
```

# EMatcher::GetPatternType

Pixel type of the learnt pattern.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::EImageType GetPatternType() const
```

# EMatcher::GetPatternWidth

Learnt pattern width.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 GetPatternWidth()
```

# EMatcher::GetPositions

Returns a vector of EMatchPosition objects, each containing the position coordinates and other matching results.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::vector<Euresys::Open_eVision_2_10::EMatchPosition> GetPositions() const
```

# EMatcher::Save

Saves the EMatcher. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Save(
 ESerializer* serializer
 )
```

## Parameters

*serializer*

The serializer.

# EMatcher::GetScaleStep

Current value of scale step.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetScaleStep()
```

# EMatcher::GetScaleXStep

Current value of scale X step.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetScaleXStep()
```

# EMatcher::GetScaleYStep

Current value of scale Y step.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetScaleYStep()
```

# EMatcher::SetExtension

Sets the extension of the matching ROI, i.e. puts the horizontal and vertical distances, in pixels, that the found pattern occurrences may fall outside the matching ROI. Such occurrences partially outside the ROI have their score corrected by the ratio between the pattern area outside the ROI and the pattern area inside.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetExtension(
 OEV_INT32 n32ExtensionX,
 OEV_INT32 n32ExtensionY
 )
```

### Parameters

*n32ExtensionX*

extension outside the matching ROI along its Width, in pixels.

*n32ExtensionY*

extension outside the matching ROI along its Height, in pixels.

# EMatcher::SetPixelDimensions

Sets the physical pixel dimensions.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetPixelDimensions(
 float width,
 float height
 )
```

### Parameters

*width*

Width of a pixel.

*height*

Height of a pixel.

### Remarks

When an image has been acquired in such a way that the pixels are "non-square" â€”the physical width and height of the area covered by a pixel are unequalâ€” a form of anisotropy results. When rotated, the objects become skewed; rectangles become parallelograms. In such a situation, the pixel aspect ratio must be known to compensate it during matching. The aspect ratio is given by specifying the true width and height of a pixel. The specification of the pixel dimensions is only useful when rotation is used. Only the aspect ratio matters, so that relative width and height values can be given. By default, the pixel width and height are set to **1.0**.

# EMatcher::GetVersion

Version number of the EMatcher object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
static OEV_UINT32 GetVersion()
```

# 4.110. EMatrixCode Class

Holds all the information regarding a single Data Matrix code: its decoded string, its grading, its errors,...

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Draw | Draws the EMatrixCode corner points and symbol finder pattern (when the symbol size has been correctly detected). |
| DrawErrors | Draws all symbol finder pattern cells where errors were detected and corrected. |
| DrawErrorsWithCurrentPen | Draws the detected errors. |
| DrawWithCurrentPen | Draws the EMatrixCode corner points and symbol finder pattern (when the symbol size has been correctly detected). |
| EMatrixCode | Constructs a EMatrixCode context. |
| GetAngle | **MatrixCode** angle. |
| GetAxialNonUniformity | Measured axial non-uniformity value (**1.0** to **0** scale), as defined by the AIM standard, after a read operation, provided that the print quality assessment has been enabled. |

| | |
|---|---|
| GetAxialNonUniformityGrade | Measured axial non-uniformity grading (**4** to **0** scale), as defined by the AIM standard, after a read operation, provided that the print quality assessment has been enabled. |
| GetCellDefects | Measured cell defects value, as defined by the Semi T10 standard, after a read operation, provided that the print quality assessment has been enabled. |
| GetCenter | EMatrixCode center. |
| GetContrast | Measured symbol contrast value, as defined by the AIM standard, after a read operation, provided that the print quality assessment has been enabled. |
| GetContrastGrade | Measured symbol contrast grading (**4** to **0** scale), as defined by the AIM standard, after a read operation, provided that the print quality assessment has been enabled. |
| GetContrastType | Symbol contrast type, as defined by EMatrixCodeContrastMode. |
| GetCorner | Gets a matrix code corner. |
| GetDataMatrixCellHeight | Measured data matrix cell height value, as defined by the Semi T10 standard, after a read operation, provided that the print quality assessment has been enabled. |

| | |
|---|---|
| GetDataMatrixCellWidth | Measured data matrix cell width value, as defined by the Semi T10 standard, after a read operation, provided that the print quality assessment has been enabled. |
| GetDecodedDataElement | Gets a character value of the matrix code. |
| GetDecodedString | Decoded Data Matrix symbol string. |
| GetFamily | ECC symbol family, as defined by EFamily. |
| GetFinderPatternDefects | Measured finder pattern defects value, as defined by the Semi T10 standard, after a read operation, provided that the print quality assessment has been enabled. |
| GetFlipping | Symbol flipping type, as defined by EFlipping. |
| GetFound | **TRUE** if a EMatrixCode object has been found in the ROI supplied to EMatrixCodeReader::Read, even if it could not be successfully decoded. |
| GetHorizontalMarkGrowth | Measured horizontal mark growth value, as defined by the Semi T10 standard, after a read operation, provided that the print quality assessment has been enabled. |

| | |
|---|---|
| GetHorizontalMarkMisplacement | Measured horizontal mark misplacement value, as defined by the Semi T10 standard, after a read operation, provided that the print quality assessment has been enabled. |
| GetIso15415GradingParameters | ISO/IEC 15415 grading parameters |
| GetIso29158GradingParameters | ISO/IEC 29158 grading parameters |
| GetLocationThreshold | Absolute threshold (**0** to **255** scale) that was used during location of the symbol. |
| GetLogicalSize | Symbol logical size, as defined by ELogicalSize. |
| GetLogicalSizeHeight | For a logical size of S1xS2, **LogicalSizeHeight** is the integer S1. |
| GetLogicalSizeWidth | For a logical size of S1xS2, **LogicalSizeWidth** is the integer S2. |
| GetMeasuredPrintGrowth | Raw, un-normalized print quality parameter (**1.0** to **0** scale), after a read operation, provided that the print quality assessment has been enabled. |
| GetNumErrors | Number of errors that were detected and corrected while decoding the symbol. |
| GetOverallGrade | Overall symbol grading (**4** to **0** scale), as defined by the AIM standard, after a read operation, provided that the print quality assessment has been enabled. |

| | |
|---|---|
| GetPrintGrowth | Normalized measured print growth value, as defined by the AIM standard, after a read operation, provided that the print quality assessment has been enabled. |
| GetPrintGrowthGrade | Measured print growth grading (**4** to **0** scale), as defined by the AIM standard, after a read operation, provided that the print quality assessment has been enabled. |
| GetReadingThreshold | Absolute threshold (**0** to **255** scale) that was used during reading of the symbol. |
| GetSemiT10GradingParameters | Semi T10-0701 grading parameters |
| GetSymbolContrastSNR | Measured symbol contrast signal to noise ratio value, as defined by the Semi T10 standard, after a read operation, provided that the print quality assessment has been enabled. |
| GetUnusedErrorCorrection | Measured unused error correction value (**1.0** to **0** scale), as defined by the AIM standard, after a read operation, provided that the print quality assessment has been enabled. |
| GetUnusedErrorCorrectionGrade | Measured unused error correction grading (**4** to **0** scale), as defined by the AIM standard, after a read operation, provided that the print quality assessment has been enabled. |
| GetVerticalMarkGrowth | Measured vertical mark growth value, as defined by the Semi T10 standard, after a read operation, provided that the print quality assessment has been enabled. |

| | |
|---|---|
| GetVerticalMarkMisplacement | Measured vertical mark misplacement value, as defined by the Semi T10 standard, after a read operation, provided that the print quality assessment has been enabled. |
| IsGS1 | **TRUE** if the decoded string uses the GS1 standard |
| Load | Saves a EMatrixCode. The given ESerializer must have been created for writing. |
| operator= | Copies all the data from another EMatrixCode object into the current EMatrixCode object |
| Save | Loads a EMatrixCode. The given ESerializer must have been created for reading. |
| SetCorner | Sets a matrix code corner. |

# EMatrixCode::GetAngle

**MatrixCode** angle.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetAngle() const
```

## Remarks

# EMatrixCode::GetAxialNonUniformity

Measured axial non-uniformity value (**1.0** to **0** scale), as defined by the AIM standard, after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetAxialNonUniformity() const
```

# EMatrixCode::GetAxialNonUniformityGrade

Measured axial non-uniformity grading (**4** to **0** scale), as defined by the AIM standard, after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 GetAxialNonUniformityGrade() const
```

# EMatrixCode::GetCellDefects

Measured cell defects value, as defined by the Semi T10 standard, after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCellDefects() const
```

### Remarks

Read-only.

# EMatrixCode::GetCenter

EMatrixCode center.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
const EPoint& GetCenter() const
```

# EMatrixCode::GetContrast

Measured symbol contrast value, as defined by the AIM standard, after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetContrast() const
```

### Remarks

This property is computed as the difference of the reference gray levels over their arithmetic average. Values range between **0** and **1.0**.

# EMatrixCode::GetContrastGrade

Measured symbol contrast grading (**4** to **0** scale), as defined by the AIM standard, after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 GetContrastGrade() const
```

# EMatrixCode::GetContrastType

Symbol contrast type, as defined by EMatrixCodeContrastMode.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::EMatrixCodeContrastMode GetContrastType() const
```

# EMatrixCode::GetDataMatrixCellHeight

Measured data matrix cell height value, as defined by the Semi T10 standard, after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetDataMatrixCellHeight() const
```

### Remarks

Read-only.

# EMatrixCode::GetDataMatrixCellWidth

Measured data matrix cell width value, as defined by the Semi T10 standard, after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetDataMatrixCellWidth() const
```

### Remarks

Read-only.

# EMatrixCode::GetDecodedString

Decoded Data Matrix symbol string.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::string GetDecodedString() const
```

# EMatrixCode::Draw

Draws the EMatrixCode corner points and symbol finder pattern (when the symbol size has been correctly detected).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*zoomX*

Horizontal zooming factor. A value greater than **1** means zoom in. By default, **TRUE** scale is used.

*zoomY*

Vertical zooming factor. A value greater than **1** means zoom in. If set to **0**, the default value, the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor, in pixels. By default, no panning occurs.

*panY*

Vertical panning factor, in pixels. By default, no panning occurs.

*color*

The color in which to draw the overlay.

### Remarks

The reference corner has a bold cross marking.

# EMatrixCode::DrawErrors

Draws all symbol finder pattern cells where errors were detected and corrected.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawErrors(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
void DrawErrors(
 HDC graphicContext,
 const ERGBColor& color,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void DrawErrors(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*zoomX*

Horizontal zooming factor. A value greater than **1** means zoom in. By default, **TRUE** scale is used.

*zoomY*

Vertical zooming factor. A value greater than **1** means zoom in. If set to **0**, the default value, the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor, in pixels. By default, no panning occurs.

*panY*

Vertical panning factor, in pixels. By default, no panning occurs.

*color*

The color in which to draw the overlay.

### Remarks

This member is intended to be called in conjunction with EMatrixCode::Draw.

# EMatrixCode::DrawErrorsWithCurrentPen

Draws the detected errors.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void DrawErrorsWithCurrentPen(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*graphicContext*

-

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

# EMatrixCode::DrawWithCurrentPen

Draws the EMatrixCode corner points and symbol finder pattern (when the symbol size has been correctly detected).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void DrawWithCurrentPen(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*zoomX*

Horizontal zooming factor. A value greater than **1** means zoom in. By default, **TRUE** scale is used.

*zoomY*

Vertical zooming factor. A value greater than **1** means zoom in. If set to **0**, the default value, the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor, in pixels. By default, no panning occurs.

*panY*

Vertical panning factor, in pixels. By default, no panning occurs.

### Remarks

The reference corner has a bold cross marking.

# EMatrixCode::EMatrixCode

Constructs a EMatrixCode context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EMatrixCode(
 )
```

```
void EMatrixCode(
 const EMatrixCode& other
 )
```

### Parameters

*other*

Another EMatrixCode object to be copied in the new EMatrixCode object.

### Remarks

The default constructor constructs an uninitialized EMatrixCode object. All properties are initialized to their respective default values. The copy constructor constructs a EMatrixCode context based on a pre-existing EMatrixCode object. All properties and internal data are copied.

# EMatrixCode::GetFamily

ECC symbol family, as defined by EFamily.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EFamily GetFamily() const
```

# EMatrixCode::GetFinderPatternDefects

Measured finder pattern defects value, as defined by the Semi T10 standard, after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetFinderPatternDefects() const
```

### Remarks

Read-only.

# EMatrixCode::GetFlipping

Symbol flipping type, as defined by EFlipping.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::EFlipping GetFlipping() const
```

# EMatrixCode::GetFound

**TRUE** if a EMatrixCode object has been found in the ROI supplied to EMatrixCodeReader::Read, even if it could not be successfully decoded.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetFound() const
```

### Remarks

If this property is **FALSE**, it is still possible that an unlocalized matrix code exists in the image. However, if **Found** is **FALSE**, no other property should be read.

# EMatrixCode::GetCorner

Gets a matrix code corner.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
const EPoint& GetCorner(
 OEV_INT32 index
 )
```

### Parameters

*index*

Index of the matrix code corner.

# EMatrixCode::GetDecodedDataElement

Gets a character value of the matrix code.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT8 GetDecodedDataElement(
 OEV_INT32 index
 )
```

### Parameters

*index*

Index of the character value.

### Remarks

This property makes it possible to see information not coded as ASCII characters.

# EMatrixCode::GetHorizontalMarkGrowth

Measured horizontal mark growth value, as defined by the Semi T10 standard, after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetHorizontalMarkGrowth() const
```

## Remarks

Read-only.

# EMatrixCode::GetHorizontalMarkMisplacement

Measured horizontal mark misplacement value, as defined by the Semi T10 standard, after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetHorizontalMarkMisplacement() const
```

## Remarks

Read-only.

# EMatrixCode::IsGS1

**TRUE** if the decoded string uses the GS1 standard

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL IsGS1(
 )
```

# EMatrixCode::GetIso15415GradingParameters

ISO/IEC 15415 grading parameters

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EMatrixCodeIso15415GradingParameters GetIso15415GradingParameters() const
```

## Remarks

Read-only.

# EMatrixCode::GetIso29158GradingParameters

ISO/IEC 29158 grading parameters

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EMatrixCodeIso29158GradingParameters GetIso29158GradingParameters() const
```

### Remarks

Read-only.

# EMatrixCode::Load

Saves a EMatrixCode. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# EMatrixCode::GetLocationThreshold

Absolute threshold (**0** to **255** scale) that was used during location of the symbol.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
OEV_INT32 GetLocationThreshold() const
```

# EMatrixCode::GetLogicalSize

Symbol logical size, as defined by ELogicalSize.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::ELogicalSize GetLogicalSize() const
```

# EMatrixCode::GetLogicalSizeHeight

For a logical size of S1xS2, **LogicalSizeHeight** is the integer S1.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetLogicalSizeHeight() const
```

# EMatrixCode::GetLogicalSizeWidth

For a logical size of S1xS2, **LogicalSizeWidth** is the integer S2.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetLogicalSizeWidth() const
```

# EMatrixCode::GetMeasuredPrintGrowth

Raw, un-normalized print quality parameter (**1.0** to **0** scale), after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetMeasuredPrintGrowth() const
```

## Remarks

This property is computed as the measured area of the active cells (same color as the finder pattern) over the area of ideal square cells, having sides equal to the pitches (such cells perfectly tile the symbol). Its value typically is on the order of **1.0**, regardless the symbol size.

# EMatrixCode::GetNumErrors

Number of errors that were detected and corrected while decoding the symbol.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetNumErrors() const
```

## Remarks

Such errors may be due to symbol degradation by scratches, blur, non-uniform illumination or slight changes in size.

# EMatrixCode::operator=

Copies all the data from another EMatrixCode object into the current EMatrixCode object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EMatrixCode& operator=(
 const EMatrixCode& other
 )
```

### Parameters

*other*

EMatrixCode object to be copied

# EMatrixCode::GetOverallGrade

Overall symbol grading (**4** to **0** scale), as defined by the AIM standard, after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetOverallGrade() const
```

# EMatrixCode::GetPrintGrowth

Normalized measured print growth value, as defined by the AIM standard, after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetPrintGrowth() const
```

## Remarks

The use of this property is a bit tricky: first a raw measure of the print growth is provided as EMatrixCode::MeasuredPrintGrowth. Then, the measurement is normalized from the EMatrixCodeReader::MinimumPrintGrowth /EMatrixCodeReader::MaximumPrintGrowth/EMatrixCodeReader::NominalPrintGrowth properties of the EMatrixCodeReader instance, which must be provided by the user. The normalized EMatrixCode::PrintGrowth ranges around **0**.

# EMatrixCode::GetPrintGrowthGrade

Measured print growth grading (**4** to **0** scale), as defined by the AIM standard, after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetPrintGrowthGrade() const
```

## Remarks

Read-only.

# EMatrixCode::GetReadingThreshold

Absolute threshold (**0** to **255** scale) that was used during reading of the symbol.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetReadingThreshold() const
```

## Remarks

Read-only.

# EMatrixCode::Save

Loads a EMatrixCode. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Save(
 ESerializer* serializer
 )
```

## Parameters

*serializer*

The ESerializer object that is read from.

# EMatrixCode::GetSemiT10GradingParameters

Semi T10-0701 grading parameters

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EMatrixCodeSemiT10GradingParameters GetSemiT10GradingParameters() const
```

## Remarks

Read-only.

# EMatrixCode::SetCorner

Sets a matrix code corner.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetCorner(
 OEV_INT32 index,
 const EPoint& corner
 )
```

## Parameters

*index*

Index of the matrix code corner.

*corner*

New corner.

# EMatrixCode::GetSymbolContrastSNR

Measured symbol contrast signal to noise ratio value, as defined by the Semi T10 standard, after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSymbolContrastSNR() const
```

## Remarks

Read-only.

# EMatrixCode::GetUnusedErrorCorrection

Measured unused error correction value (**1.0** to **0** scale), as defined by the AIM standard, after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetUnusedErrorCorrection() const
```

## Remarks

The EMatrixCode::UnusedErrorCorrection property takes into account the number of redundant bits used for error correction only; no erasure nor error detection bits are considered.

# EMatrixCode::GetUnusedErrorCorrectionGrade

Measured unused error correction grading (**4** to **0** scale), as defined by the AIM standard, after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetUnusedErrorCorrectionGrade() const
```

## Remarks

Read-only.

# EMatrixCode::GetVerticalMarkGrowth

Measured vertical mark growth value, as defined by the Semi T10 standard, after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetVerticalMarkGrowth() const
```

## Remarks

Read-only.

# EMatrixCode::GetVerticalMarkMisplacement

Measured vertical mark misplacement value, as defined by the Semi T10 standard, after a read operation, provided that the print quality assessment has been enabled.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetVerticalMarkMisplacement() const
```

### Remarks

Read-only.

# 4.111. EMatrixCode Class

Holds all the information regarding a single Data Matrix code: its decoded string, its grading, its errors and more.

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

## Methods

| | |
|---|---|
| DrawErrors | Draws the detected errors. |
| DrawGrid | Draws the detected Data Matrix grid. |
| DrawPosition | Draws the Data Matrix Position. This includes the outer edges of the code as well as the timing patterns. |

| | |
|---|---|
| EMatrixCode | Creates an EMatrixCode object. |
| GetCellPosition | Position of a cell of the Data Matrix |
| GetDecodedString | Decoded Data Matrix symbol string. |
| GetECC000Family | Retrieves the ECC000 Family for non-ECC200 Matrix Codes. |
| GetErrors | Retrieves the position of the errors detected in the Data Matrix symbol. |
| GetIsECC200 | Indicates if the Data Matrix is ECC200 or not. |
| GetIso15415GradingParameters | ISO/IEC 15415 grading parameters |
| GetIso29158GradingParameters | ISO/IEC TR 29158 grading parameters |
| GetPosition | Position of the Data Matrix |
| GetSemiT10GradingParameters | Semi T10-0701 grading parameters |
| GetSymbolHeight | Data Matrix Symbol Height (Number of cells in the vertical direction) |

| | |
|---|---|
| GetSymbolWidth | Data Matrix Symbol Width (Number of cells in the horizontal direction) |
| IsGS1 | **TRUE** if the decoded string uses the GS1 standard |
| operator= | Assignment operator |

# EMatrixCode::GetDecodedString

Decoded Data Matrix symbol string.

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

[C++]

```
std::string GetDecodedString() const
```

# EMatrixCode::DrawErrors

Draws the detected errors.

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

[C++]

```
void DrawErrors(
 HDC hDC,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*hDC*

Handle of the device context on which to draw.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

# EMatrixCode::DrawGrid

Draws the detected Data Matrix grid.

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

```
[C++]
```

```
void DrawGrid(
 HDC hDC,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*hDC*

Handle of the device context on which to draw.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

# EMatrixCode::DrawPosition

Draws the Data Matrix Position. This includes the outer edges of the code as well as the timing patterns.

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

```
[C++]
```

```
void DrawPosition(
 HDC hDC,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*hDC*

Handle of the device context on which to draw.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

# EMatrixCode::GetECC000Family

Retrieves the ECC000 Family for non-ECC200 Matrix Codes.

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

```
[C++]
Euresys::Open_eVision_2_10::EasyMatrixCode2::ECC000Family GetECC000Family() const
```

# EMatrixCode::EMatrixCode

Creates an EMatrixCode object.

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

[C++]

```
void EMatrixCode(
 )

void EMatrixCode(
 const EMatrixCode& other
 )
```

## Parameters

*other*

The reference EMatrixCode instance to copy this one from.

# EMatrixCode::GetErrors

Retrieves the position of the errors detected in the Data Matrix symbol.

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

[C++]

```
std::vector<Euresys::Open_eVision_2_10::EMatrixPosition> GetErrors() const
```

# EMatrixCode::GetCellPosition

Position of a cell of the Data Matrix

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

```
[C++]
EQuadrangle GetCellPosition(
 int x,
 int y
 )
EQuadrangle GetCellPosition(
 EMatrixPosition position
 )
```

### Parameters

*x*

The horizontal index of the cell.

*y*

The vertical index of the cell.

*position*

The position of the cell in the code.

# EMatrixCode::GetIsECC200

Indicates if the Data Matrix is ECC200 or not.

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

```
[C++]
```

```
bool GetIsECC200() const
```

# EMatrixCode::IsGS1

**TRUE** if the decoded string uses the GS1 standard

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

[C++]

```
bool IsGS1(
 )
```

# EMatrixCode::GetIso15415GradingParameters

ISO/IEC 15415 grading parameters

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

[C++]

```
EMatrixCodeIso15415GradingParameters GetIso15415GradingParameters() const
```

# EMatrixCode::GetIso29158GradingParameters

ISO/IEC TR 29158 grading parameters

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

[C++]

```
EMatrixCodeIso29158GradingParameters GetIso29158GradingParameters() const
```

# EMatrixCode::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

[C++]

```
EMatrixCode& operator=(
 const EMatrixCode& other
 )
```

## Parameters

*other*

The EMatrixCode instance to assign.

# EMatrixCode::GetPosition

Position of the Data Matrix

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

[C++]

```
EQuadrangle GetPosition() const
```

# EMatrixCode::GetSemiT10GradingParameters

Semi T10-0701 grading parameters

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

[C++]

```
EMatrixCodeSemiT10GradingParameters GetSemiT10GradingParameters() const
```

# EMatrixCode::GetSymbolHeight

Data Matrix Symbol Height (Number of cells in the vertical direction)

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

[C++]

```
int GetSymbolHeight() const
```

# EMatrixCode::GetSymbolWidth

Data Matrix Symbol Width (Number of cells in the horizontal direction)

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

[C++]

```
int GetSymbolWidth() const
```

# 4.112. EMatrixCodeReader Class

A EMatrixCodeReader instance is a tool that processes an ROI and returns a EMatrixCode instance.

### Remarks

A EMatrixCodeReader has properties that allow customizing the reading process. One of these properties is EMatrixCodeReader::SearchParams, which is an instance of ESearchParamsType. This object embeds the parameter space where a Data Matrix code will be searched and has an interface of its own. The EMatrixCodeReader class is found in the Euresys namespace.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EMatrixCodeReader | Default constructor for EMatrixCodeReader objects. |
| GetComputeGrading | Allows to choose whether the grading properties of the EMatrixCode object will be computed by the EMatrixCodeReader::Reset, EMatrixCodeReader::Read or EMatrixCodeReader::LearnMore methods. |
| GetLearnMaskElement | Allows to know which decoded parameters are learnt when the EMatrixCodeReader::Learn or EMatrixCodeReader::LearnMore methods are called. |
| GetMaxHeightWidthRatio | Maximum value for a Data Matrix aspect ratio |

| | |
|---|---|
| GetMaximumPrintGrowth | Maximum reference value in use for normalization of the **PrintGrowth** quality indicator. |
| GetMinimumPrintGrowth | Minimum reference value in use for normalization of the **PrintGrowth** quality indicator. |
| GetNominalPrintGrowth | Nominal reference value in use for normalization of the **PrintGrowth** quality indicator. |
| GetSearchParams | Parameter space that the algorithm uses to read a Data Matrix code from an ROI. |
| GetTimeOut | Time-out for the EMatrixCodeReader::Learn, EMatrixCodeReader::LearnMore and **Read** methods. |
| Learn | Tries to locate, decode and read the Data Matrix code in the given ROI. |
| LearnMore | Tries to locate, decode and read the Data Matrix code in the given ROI. |
| Load | Saves a EMatrixCodeReader. The given ESerializer must have been created for writing. |

| | |
|---|---|
| Read | Tries to locate, decode and read the Data Matrix code in the given ROI. |
| Reset | Resets the parameter search space to its default: the full range of all parameters. |
| Save | Loads a EMatrixCodeReader. The given ESerializer must have been created for reading. |
| SetComputeGrading | Allows to choose whether the grading properties of the EMatrixCode object will be computed by the EMatrixCodeReader::Reset, EMatrixCodeReader::Read or EMatrixCodeReader::LearnMore methods. |
| SetIso29158CalibrationParameters | Sets ISO/IEC 29158 calibration paramters |
| SetLearnMaskElement | Allows to choose which decoded parameters are learnt when the EMatrixCodeReader::Learn or EMatrixCodeReader::LearnMore methods are called. |
| SetMaxHeightWidthRatio | Maximum value for a Data Matrix aspect ratio |
| SetMaximumPrintGrowth | Maximum reference value in use for normalization of the **PrintGrowth** quality indicator. |

| | |
|---|---|
| SetMinimumPrintGrowth | Minimum reference value in use for normalization of the **PrintGrowth** quality indicator. |
| SetNominalPrintGrowth | Nominal reference value in use for normalization of the **PrintGrowth** quality indicator. |
| SetTimeOut | Time-out for the EMatrixCodeReader::Learn, EMatrixCodeReader::LearnMore and **Read** methods. |

# EMatrixCodeReader::GetComputeGrading

# EMatrixCodeReader::SetComputeGrading

Allows to choose whether the grading properties of the EMatrixCode object will be computed by the EMatrixCodeReader::Reset, EMatrixCodeReader::Read or EMatrixCodeReader::LearnMore methods.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetComputeGrading()
void SetComputeGrading(BOOL value)
```

## Remarks

Default: **FALSE**.

# EMatrixCodeReader::EMatrixCodeReader

Default constructor for EMatrixCodeReader objects.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EMatrixCodeReader(
 )
void EMatrixCodeReader(
 const EMatrixCodeReader& other
 )
```

### Parameters

*other*

-

# EMatrixCodeReader::GetLearnMaskElement

Allows to know which decoded parameters are learnt when the EMatrixCodeReader::Learn or EMatrixCodeReader::LearnMore methods are called.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool GetLearnMaskElement(
 Euresys::Open_eVision_2_10::ELearnParam index
 )
```

### Parameters

*index*

Parameter identifier, as defined in ELearnParam

# EMatrixCodeReader::Learn

Tries to locate, decode and read the Data Matrix code in the given ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EMatrixCode Learn(
 const EROIBW8& roi
 )
```

### Parameters

*roi*

ROI in which the Data Matrix has to be found.

### Remarks

If successful, it adds the parameters of the Data Matrix code found into the internal learning database. The decoding results can be found in the returned EMatrixCode object. The addition of the parameters of the Data Matrix code found into the internal learning database means that subsequent **Read** operations will be faster, but can only be performed on similar Data Matrix codes/conditions. Only the parameters that are tagged for learning are remembered for the subsequent EMatrixCodeReader::Read operations (see EMatrixCodeReader::SetLearnMaskElement). See the EMatrixCode::Found property for information about the outcome of the **Learn** process.

# EMatrixCodeReader::LearnMore

Tries to locate, decode and read the Data Matrix code in the given ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EMatrixCode LearnMore(
 const EROIBW8& roi
 )
```

### Parameters

*roi*

ROI in which the Data Matrix has to be found.

### Remarks

If successful, it cumulates the parameters of the Data Matrix code found with those already present in the internal learning database. The decoding results can be found in the returned EMatrixCode object. The cumulation of the parameters of the Data Matrix code found with those already present in the internal learning database means that subsequent **Read** operations will be faster, but can only be performed on similar Data Matrix codes/conditions. Only the parameters that are tagged for learning are remembered for the subsequent EMatrixCodeReader::Read operations (see EMatrixCodeReader::SetLearnMaskElement). See the EMatrixCode::Found property for information about the outcome of the **LearnMore** process.

# EMatrixCodeReader::Load

Saves a EMatrixCodeReader. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# EMatrixCodeReader::GetMaxHeightWidthRatio

# EMatrixCodeReader::SetMaxHeightWidthRatio

Maximum value for a Data Matrix aspect ratio

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetMaxHeightWidthRatio()
void SetMaxHeightWidthRatio(float value)
```

## Remarks

This property allows controlling what kind of objects are considered as potential MatrixCode instances in the image. When objects are found in the image, only those where the bounding box has an aspect ratio smaller than this value are taken into account for digitization and decoding. The default value is 3.8, and should be adjusted if the MatrixCode cells in your image are non-square, or if your matrix code uses a very non-square symbology such as 32x8. The supplied value must lie between 0.0 and 5.0.

# EMatrixCodeReader::GetMaximumPrintGrowth

# EMatrixCodeReader::SetMaximumPrintGrowth

Maximum reference value in use for normalization of the **PrintGrowth** quality indicator.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
float GetMaximumPrintGrowth()
void SetMaximumPrintGrowth(float value)
```

### Remarks

Default: **2.0**. After the standard, parameter **PrintGrowth** must be computed as normalized value related to three references: minimum, nominal and maximum values have to be provided. Parameter **MeasuredPrintGrowth** is the raw, un-normalized print quality parameter. It is computed as the measured area of the active cells over the area of ideal square cells, having sides equal to the pitches (such cells perfectly tile the symbol). Its value typically is on the order of unity. The **PrintGrowth** is derived from the **MeasuredPrintGrowth** by means of the three normalization parameters.

# EMatrixCodeReader::GetMinimumPrintGrowth

# EMatrixCodeReader::SetMinimumPrintGrowth

Minimum reference value in use for normalization of the **PrintGrowth** quality indicator.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetMinimumPrintGrowth()
void SetMinimumPrintGrowth(float value)
```

### Remarks

Default: **0.0**. After the standard, parameter **PrintGrowth** must be computed as normalized value related to three references: minimum, nominal and maximum values have to be provided. Parameter **MeasuredPrintGrowth** is the raw, un-normalized print quality parameter. It is computed as the measured area of the active cells over the area of ideal square cells, having sides equal to the pitches (such cells perfectly tile the symbol). Its value typically is on the order of unity. The **PrintGrowth** is derived from the **MeasuredPrintGrowth** by means of the three normalization parameters.

# EMatrixCodeReader::GetNominalPrintGrowth

# EMatrixCodeReader::SetNominalPrintGrowth

Nominal reference value in use for normalization of the **PrintGrowth** quality indicator.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetNominalPrintGrowth()
void SetNominalPrintGrowth(float value)
```

### Remarks

Default: **1.0**. After the standard, parameter **PrintGrowth** must be computed as normalized value related to three references: minimum, nominal and maximum values have to be provided. Parameter **MeasuredPrintGrowth** is the raw, un-normalized print quality parameter. It is computed as the measured area of the active cells over the area of ideal square cells, having sides equal to the pitches (such cells perfectly tile the symbol). Its value typically is on the order of unity. The **PrintGrowth** is derived from the **MeasuredPrintGrowth** by means of the three normalization parameters.

# EMatrixCodeReader::Read

Tries to locate, decode and read the Data Matrix code in the given ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EMatrixCode Read(
 const EROIBW8& roi
 )
```

### Parameters

*roi*

ROI in which the Data Matrix has to be found.

### Remarks

The decoding results can be found in the returned EMatrixCode object. See the EMatrixCode::Found property for information about the outcome of the **Read** process.

**Note.** This function throws an exception if the matrix code in the given ROI can not be read.

# EMatrixCodeReader::Reset

Resets the parameter search space to its default: the full range of all parameters.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Reset(
 )
```

### Remarks

This does not modify the learning mask.

# EMatrixCodeReader::Save

Loads a EMatrixCodeReader. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# EMatrixCodeReader::GetSearchParams

Parameter space that the algorithm uses to read a Data Matrix code from an ROI.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
ESearchParamsType& GetSearchParams()
```

### Remarks

It can be modified through automatic learning (using the EMatrixCodeReader::Learn or EMatrixCodeReader::LearnMore methods) or by using EMatrixCodeReader::SearchParams properties and methods.

# EMatrixCodeReader::SetIso29158CalibrationParameters

Sets ISO/IEC 29158 calibration paramters

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetIso29158CalibrationParameters(
 float Rcal,
 float MLcal,
 float SRcal,
 float SRtarget
 )
```

### Parameters

*Rcal*

Reported reflectance value, from a calibration standard.

*MLcal*

Mean of the light from a histogram of the calibrated standard.

*SRcal*

System response parameters(such as exposure and/or again) used to create an image of the calibration standard.

*SRtarget*

System response parameters(such as exposure and/or again) used to create an image of the symbole under test.

# EMatrixCodeReader::SetLearnMaskElement

Allows to choose which decoded parameters are learnt when the EMatrixCodeReader::Learn or EMatrixCodeReader::LearnMore methods are called.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetLearnMaskElement(
 Euresys::Open_eVision_2_10::ELearnParam index,
 bool value
 )
```

### Parameters

*index*

Parameter identifier, as defined in ELearnParam.

*value*

**TRUE** to enable the parameter for learning.

### Remarks

In order to enable a parameter for learning, you need to set corresponding item of LearnMask to TRUE. Default: all items are set to TRUE.

## EMatrixCodeReader::GetTimeOut

## EMatrixCodeReader::SetTimeOut

Time-out for the EMatrixCodeReader::Learn, EMatrixCodeReader::LearnMore and **Read** methods.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetTimeOut()
void SetTimeOut(OEV_UINT32 value)
```

### Remarks

If the processing time of one of these functions becomes longer than the set time-out, the processing is stopped and an exception is thrown. In that case, the error code of the exception is EError_TimeoutReached. The time-out is set in microseconds. This time-out is not a real time-out. The processing is stopped as soon as possible after the time-out has been reached. This means that the time elapsed effectively in the method can be greater than the time-out in itself.

# 4.113. EMatrixCodeReader Class

An EMatrixCodeReader instance can detect, decode and grade matrixcodes in an ROI, it returns a vector of EMatrixCode instances.

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

## Methods

| | |
|---|---|
| EMatrixCodeReader | Creates an EMatrixCodeReader object. |
| GetComputeGrading | Allows to choose whether the grading properties of the EMatrixCode object will be computed by the EMatrixCodeReader::Read method. The default setting for this property is `false`. |
| GetIso29158CalibrationParameters | ISO/IEC TR 29158 calibration parameters |
| GetMaxNumCodes | The maximum number of codes that the reader should try to find. |
| GetReadMode | The EReadMode used for the EMatrixCodeReader::Read method. The default value for this property is EReadMode_Speed. |
| GetReadResults | Outputs the codes that were found by the EMatrixCodeReader::Read method |
| GetTimeOut | The timeout for the EMatrixCodeReader::Read and EMatrixCodeReader::Learn method in microseconds. |
| Learn | Learns the optimal parameter settings for detecting data matrix codes in the given ROI |
| Load | Load the configuration for this EMatrixCodeReader instance. |

| | |
|---|---|
| operator= | Assignment operator |
| Read | Tries to locate, decode and read data matrix codes in the given ROI |
| ResetLearning | Forgets the learned parameter settings and resets their default values |
| Save | Save the configuration for this EMatrixCodeReader instance. |
| SetComputeGrading | Allows to choose whether the grading properties of the EMatrixCode object will be computed by the EMatrixCodeReader::Read method. The default setting for this property is `false`. |
| SetIso29158CalibrationParameters | ISO/IEC TR 29158 calibration parameters |
| SetMaxNumCodes | The maximum number of codes that the reader should try to find. |
| SetReadMode | The EReadMode used for the EMatrixCodeReader::Read method. The default value for this property is EReadMode_Speed. |
| SetTimeOut | The timeout for the EMatrixCodeReader::Read and EMatrixCodeReader::Learn method in microseconds. |
| StopProcess | Stops the EMatrixCodeReader::Read and/or EMatrixCodeReader::Learn process as soon as possible. |

# EMatrixCodeReader::GetComputeGrading

# EMatrixCodeReader::SetComputeGrading

Allows to choose whether the grading properties of the EMatrixCode object will be computed by the EMatrixCodeReader::Read method. The default setting for this property is `false`.

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

```
[C++]
bool GetComputeGrading() const
void SetComputeGrading(bool computeGrading)
```

# EMatrixCodeReader::EMatrixCodeReader

Creates an EMatrixCodeReader object.

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

```
[C++]
void EMatrixCodeReader(
 )
void EMatrixCodeReader(
 const EMatrixCodeReader& other
 )
```

## Parameters

*other*

Another EMatrixCodeReader object to be copied in the new EMatrixCodeReader object.

# EMatrixCodeReader::GetIso29158CalibrationParameters

# EMatrixCodeReader::SetIso29158CalibrationParameters

ISO/IEC TR 29158 calibration parameters

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

```
[C++]
EMatrixCodeIso29158CalibrationParameters GetIso29158CalibrationParameters() const
void SetIso29158CalibrationParameters(EMatrixCodeIso29158CalibrationParameters params)
```

# EMatrixCodeReader::Learn

Learns the optimal parameter settings for detecting data matrix codes in the given ROI

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

```
[C++]
void Learn(
 const EROIBW8& roi
 )
```

## Parameters

*roi*

The ROI in which the data matrix codes have to be found.

# EMatrixCodeReader::Load

Load the configuration for this EMatrixCodeReader instance.

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

```
[C++]
void Load(
 const std::string& path
 )
```

## Parameters

*path*

The path from which to load the configuration.

# EMatrixCodeReader::GetMaxNumCodes

# EMatrixCodeReader::SetMaxNumCodes

The maximum number of codes that the reader should try to find.

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

```
[C++]
OEV_UINT32 GetMaxNumCodes() const
void SetMaxNumCodes(OEV_UINT32 maxNumCodes)
```

## Remarks

By default, this parameter is set to 1. If this property is set to 0, the reader will try to find as many codes as possible.

# EMatrixCodeReader::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

```
[C++]
EMatrixCodeReader& operator=(
 const EMatrixCodeReader& other
 )
```

## Parameters

*other*

EMatrixCodeReader object to be copied.

# EMatrixCodeReader::Read

Tries to locate, decode and read data matrix codes in the given ROI

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

```
[C++]
void Read(
 const EROIBW8& roi
 )
```

## Parameters

*roi*

The ROI in which the data matrix codes have to be found.

# EMatrixCodeReader::GetReadMode

# EMatrixCodeReader::SetReadMode

The EReadMode used for the EMatrixCodeReader::Read method. The default value for this property is Speed.

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

```
[C++]
Euresys::Open_eVision_2_10::EasyMatrixCode2::EReadMode GetReadMode() const
void SetReadMode(Euresys::Open_eVision_2_10::EasyMatrixCode2::EReadMode mode)
```

# EMatrixCodeReader::GetReadResults

Outputs the codes that were found by the EMatrixCodeReader::Read method

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

```
[C++]
std::vector<Euresys::Open_eVision_2_10::EasyMatrixCode2::EMatrixCode> GetReadResults()
const
```

# EMatrixCodeReader::ResetLearning

Forgets the learned parameter settings and resets their default values

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

[C++]

```
void ResetLearning(
 )
```

# EMatrixCodeReader::Save

Save the configuration for this EMatrixCodeReader instance.

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

[C++]

```
void Save(
 const std::string& path
 )
```

## Parameters

*path*

The path to which to save the configuration.

# EMatrixCodeReader::StopProcess

Stops the EMatrixCodeReader::Read and/or EMatrixCodeReader::Learn process as soon as possible.

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

[C++]

```
void StopProcess(
 )
```

### Remarks

When this method is called the process is stopped at the first checkpoint.

## EMatrixCodeReader::GetTimeOut

## EMatrixCodeReader::SetTimeOut

The timeout for the EMatrixCodeReader::Read and EMatrixCodeReader::Learn method in microseconds.

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

```
[C++]
OEV_UINT32 GetTimeOut() const
void SetTimeOut(OEV_UINT32 timeout)
```

### Remarks

If the processing time of one of these methods becomes longer than the set time-out period, the process is stopped.
The EMatrixCodeReader::Read method will return all the codes it has decoded up to that point.
The EMatrixCodeReader::Learn method will only learn from those codes it has found within the time-out period.
Note that the time-out period is not exact: the process is stopped at the first checkpoint after the time-out period has elapsed.

# 4.114. EMeasurementUnit Class

The measurement units that are supported by Open eVision.

### Remarks

Measurement units are used to represent physical units, such as "meter" or "inch", and ease conversions between different unit systems. They are used to build dimensional values. The following length measurement units are predefined: **EUnit_um** (microns), **EUnit_mm**, **EUnit_cm**, **EUnit_dm**, **EUnit_m**, **EUnit_Dm**, **EUnit_Hm**, **EUnit_Km**, **EUnit_mil** (1/1000 inch), **EUnit_inch**, **EUnit_foot**, **EUnit_yard**, **EUnit_mile**.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| ConversionFactorTo | Returns the factor needed to convert from this measurement unit to a second one. |
| EMeasurementUnit | Constructs a measurement unit. |
| GetMagnitude | Relative magnitude of this unit, with respect to a standard unit (e.g. 1 mm = 0.001 m). |
| GetName | Pointer to a **NULL**-terminated string containing the unit abbreviation. |
| GetStockMeasurementUnit | - |
| SetMagnitude | Relative magnitude of this unit, with respect to a standard unit (e.g. 1 mm = 0.001 m). |
| SetName | Pointer to a **NULL**-terminated string containing the unit abbreviation. |

# EMeasurementUnit::ConversionFactorTo

Returns the factor needed to convert from this measurement unit to a second one.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float ConversionFactorTo(
 const EMeasurementUnit& Unit
 )
```

## Parameters

*Unit*

Reference to the second measurement unit.

# EMeasurementUnit::EMeasurementUnit

Constructs a measurement unit.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EMeasurementUnit(
 float magnitude,
 const std::string& name
 )

void EMeasurementUnit(
 EMeasurementUnit* pUnit
 )

void EMeasurementUnit(
 )
```

### Parameters

*magnitude*

Relative magnitude of this unit with respect to a standard (e.g. 1 mm = 0.001 m).

*name*

Unit abbreviation (e.g. "**mm**").

*pUnit*

-

## EMeasurementUnit::GetStockMeasurementUnit

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EMeasurementUnit* GetStockMeasurementUnit(
 Euresys::Open_eVision_2_10::EStockMeasurementUnit unit
 )
```

### Parameters

*unit*

-

## EMeasurementUnit::GetMagnitude

## EMeasurementUnit::SetMagnitude

Relative magnitude of this unit, with respect to a standard unit (e.g. 1 mm = 0.001 m).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetMagnitude() const
void SetMagnitude(float f32Magnitude)
```

## EMeasurementUnit::GetName

## EMeasurementUnit::SetName

Pointer to a **NULL**-terminated string containing the unit abbreviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::string GetName() const
void SetName(const std::string& name)
```

# 4.115. EMemorySerializer Class

Handles and EMemorySerializer context

**Base Class:** ESerializer
**Namespace:** Euresys::Open_eVision_2_10

### Methods

| | |
|---|---|
| Close | Closes the serializer |

| | |
|---|---|
| GetBuffer | Address of the internal buffer. |
| GetBufferSize | Size of the internal buffer |
| GetCurrentPosition | Current position in the buffer |
| GetWriting | Checks if the serializer is in writing mode |

# EMemorySerializer::GetBuffer

Address of the internal buffer.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
const void* GetBuffer() const
```

# EMemorySerializer::GetBufferSize

Size of the internal buffer

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetBufferSize() const
```

# EMemorySerializer::Close

Closes the serializer

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Close(
 )
```

# EMemorySerializer::GetCurrentPosition

Current position in the buffer

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetCurrentPosition()
```

# EMemorySerializer::GetWriting

Checks if the serializer is in writing mode

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetWriting() const
```

# 4.116. EMesh Class

Represents a 3D meshed object (https://en.wikipedia.org/wiki/Triangle_mesh).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| EMesh | Creates an EMesh object. |
| GetPointCloud | Returns the point cloud of the EMesh object. |
| GetTriangleCount | Returns the number of triangles.<br>If the EMesh object has no triangle mesh data, the method returns 0. |
| GetTriangleIndexes | Returns a pointer to the index array (an array of 'int') representing the list of triangles. Indexes are referring to the array of E3DPoint.<br>A triangle is defined by 3 indexes (T1a T1b T1c T2a T2b T2c T3a ....). The triangle index array size is a multiple of 3.<br>Index values must be in [0, N[ where N is the number of points in the point cloud.<br>If the EMesh object has no triangle mesh data, this method returns NULL. |

| | |
|---|---|
| Load | Loads the 3D Object. The given ESerializer must have been created for reading. |
| LoadSTL | Loads a triangle mesh from a STL file (https://en.wikipedia.org/wiki/STL_(file_format)) |
| operator= | Assignment operator. |
| Save | Saves the 3D Object. The given ESerializer must have been created for writing. |
| SaveSTL | Saves the triangle mesh to an STL file (https://en.wikipedia.org/wiki/STL_(file_format)). Only ASCII format is supported. |

E

M

# esh::EMesh

Creates an EMesh object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EMesh(
 )
void EMesh(
 const EPointCloud& point_cloud
 )
```

```
void EMesh(
 const EPointCloud& point_cloud,
 const std::vector<int>& triangle_indexes
 )

void EMesh(
 const EMesh& other
 )
```

### Parameters

*point_cloud*

A point cloud

*triangle_indexes*

The triangle mesh is a list of indexes, referring to the array contained in "point_cloud".
A triangle is defined by 3 indexes (T1a T1b T1c T2a T2b T2c T3a ....).
Index values must be in [0, N[ with N the number of points in the EPointCloud.
The triangle index array size is a multiple of 3.

*other*

Another EMesh.

# EMesh::Load

Loads the 3D Object. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# EMesh::LoadSTL

Loads a triangle mesh from a STL file (https://en.wikipedia.org/wiki/STL_(file_format))

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void LoadSTL(
 const std::string& path
 )
```

### Parameters

*path*

The path to the STL file.

# EMesh::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EMesh& operator=(
 const EMesh& other
 )
```

### Parameters

*other*

-

# EMesh::GetPointCloud

Returns the point cloud of the EMesh object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const EPointCloud& GetPointCloud() const
```

# EMesh::Save

Saves the 3D Object. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 ESerializer* serializer
 )
```

## Parameters

*serializer*

The serializer.

# EMesh::SaveSTL

Saves the triangle mesh to an STL file (https://en.wikipedia.org/wiki/STL_(file_format)). Only ASCII format is supported.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveSTL(
 const std::string& path
 )
```

### Parameters

*path*

The path to the STL file.

# EMesh::GetTriangleCount

Returns the number of triangles.
If the EMesh object has no triangle mesh data, the method returns 0.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
int GetTriangleCount() const
```

## EMesh::GetTriangleIndexes

Returns a pointer to the index array (an array of 'int') representing the list of triangles. Indexes are referring to the array of E3DPoint.
A triangle is defined by 3 indexes (T1a T1b T1c T2a T2b T2c T3a ....). The triangle index array size is a multiple of 3.
Index values must be in [0, N[ where N is the number of points in the point cloud.
If the EMesh object has no triangle mesh data, this method returns NULL.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const void* GetTriangleIndexes() const
```

# 4.117. EMeshToZMapConverter Class

Computes a EZMap from a EPointCloud. The value of the pixels of the ZMap are the distance between the 3D points and the reference plane.
All 3D points under the reference plane are discarded.
Various options can be set with methods EMeshToZMapConverter, EMeshToZMapConverter::SetFillMode, EMeshToZMapConverter, EMeshToZMapConverter, EMeshToZMapConverter...
When the conversion is called without defining specific parameters, the algorithm uses the following options:
- The reference plane is the horizontal plane.
- The orientation vector is selected automatically.
- The origin is set as the lowest left position of the projected point cloud on the reference plane.
- The resolution (the dimensions of the Z map) is estimated to have approximately one Point Cloud point per ZMap pixels.
- The scale is calculated from the point cloud ranges and the estimated resolution.
- The fill mode is enabled and the method is set to 'EFillUndefinedPixelsDirection_Local' (see method EDepthMap8::FillUndefinedPixels).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| Convert | Computes a EZMap from a world space EMesh. The value of the pixels of the ZMap are the distance between the 3D triangles and the reference plane. Various options can be set with methods EMeshToZMapConverter, EMeshToZMapConverter, EMeshToZMapConverter::SetMapSize, ... |
| EMeshToZMapConverter | Creates a EMeshToZMapConverter object. |
| EnableFillMode | Enables or disables fill mode. Fill mode parameters are defined by method EMeshToZMapConverter::SetFillMode.<br>Fill mode is enabled by default. If fill mode is disable, undefined pixels may remain in the EZMap. |
| GetExtension | Sets a metric value used to enlarge the point cloud 3D domain. That value affects X,Y and Z directions and can be used to generate an EZMap with borders of undefined pixels.<br>Default value is **0**, which means a ZMap without border. |
| GetFillUndefinedPixelsDirection | Gets the undefined pixel fill direction (see EFillUndefinedPixelsDirection). |
| GetFillUndefinedPixelsMethod | Gets the undefined pixel fill method (see EFillUndefinedPixelsMethod). |
| GetMapHeight | Gets the required height (number of pixels) of the generated EZMap.<br>By default, the required size is not set. |

| | |
|---|---|
| GetMapWidth | Gets the required width (number of pixels) of the generated EZMap.<br>By default, the required size is not set. |
| GetMapXResolution | Gets the resolution of the EZMap pixels along the X axis. |
| GetMapYResolution | Gets the resolution of the EZMap pixels along the Y axis. |
| GetMapZResolution | Gets/sets the EZMap Z resolution, in world space units per gray value.<br>The resolution is used to compute the transformation of the world Z position to an integer 8, 16 or 32 bits pixel value. |
| GetOrientationVector | Sets an explicit orientation for the EZMap.<br>Overides the orientation mode given by the method EMeshToZMapConverter. |
| GetOrientationVectorMode | Chooses the EZMap orientation from a list of pre-defined axis, automatic mode or user defined vector.<br>Use EMeshToZMapConverter to set an explicit orientation vector for the ZMap. |
| GetOrigin | Chooses the EZMap origin. It is the 3D world position used as origin of the ZMap upper left pixel (0,0). |
| GetReferencePlane | Sets the E3DPlane reference plane.<br>The resulting EZMap is the distance of the 3D points above that plane.<br>3D points below the reference plane are discarded. |

GetReferencePlaneMode

Sets an axis aligned reference plane.
Overides the explicit reference plane given by the method EMeshToZMapConverter.

GetWorldToZMapTransform

Explicitly sets the world to ZMap transformation.
"SetWorldToZMapTransform" overides the settings done by EMeshToZMapConverter, EMeshToZMapConverter and EMeshToZMapConverter.
That E3DTransformMatrix transform expresses how the world positions are transformed to the EZMap space.
The matrix must be a rigid transformation (translation and rotation only).
The resolution and scales of the ZMap are defined by EMeshToZMapConverter, EMeshToZMapConverter and EMeshToZMapConverter methods.

GetZMaptoWorldTransform

Explicitly sets the ZMap to World transformation.
"SetZMaptoWorldTransform" overides the settings done by EMeshToZMapConverter, EMeshToZMapConverter and EMeshToZMapConverter.
That E3DTransformMatrix transform expresses how the EZMap positions are transformed to the World space.
The matrix must be a rigid transformation (translation and rotation only).
The resolution and scales of the World are defined by EMeshToZMapConverter, EMeshToZMapConverter and EMeshToZMapConverter methods.

IsFillModeEnabled

Tells if the fill mode is enabled or not.
Use EMeshToZMapConverter::EnableFillMode to toggle the fill mode and EMeshToZMapConverter::SetFillMode to set the filling parameters.

| | |
|---|---|
| Load | Loads the converter configuration. The given ESerializer must have been created for reading. |
| operator= | Assignment operator |
| operator== | Comparison operator |
| Save | Saves the converter configuration. The given ESerializer must have been created for writing. |
| SetExtension | Sets a metric value used to enlarge the point cloud 3D domain. That value affects X,Y and Z directions and can be used to generate an EZMap with borders of undefined pixels.<br>Default value is **0**, which means a ZMap without border. |
| SetFillMode | Inpainting options used to fill the "holes" in the EZMap. A hole exists when no 3D point is projected at that pixel position in the ZMap. |
| SetMapSize | Sets the required size of the generated EZMap; expressed in number of pixels for width and height dimensions.<br>By default, the required size is not set. |
| SetMapXYResolution | Sets the resolution (possibly anisotropic) of the EZMap pixels along the X and Y axes, in world space units per pixel (e.g mm/pixel).<br>The resolution is used to compute the ZMap size (width and height), depending on the projected point cloud on the reference plane. |

| | |
|---|---|
| SetMapZResolution | Gets/sets the EZMap Z resolution, in world space units per gray value.<br>The resolution is used to compute the transformation of the world Z position to an integer 8, 16 or 32 bits pixel value. |
| SetOrientationVector | Sets an explicit orientation for the EZMap.<br>Overides the orientation mode given by the method EMeshToZMapConverter. |
| SetOrientationVectorMode | Chooses the EZMap orientation from a list of pre-defined axis, automatic mode or user defined vector.<br>Use EMeshToZMapConverter to set an explicit orientation vector for the ZMap. |
| SetOrigin | Chooses the EZMap origin. It is the 3D world position used as origin of the ZMap upper left pixel (0,0). |
| SetReferencePlane | Sets the E3DPlane reference plane.<br>The resulting EZMap is the distance of the 3D points above that plane.<br>3D points below the reference plane are discarded. |
| SetReferencePlaneMode | Sets an axis aligned reference plane.<br>Overides the explicit reference plane given by the method EMeshToZMapConverter. |

SetWorldToZMapTransform

Explicitly sets the world to ZMap transformation.
"SetWorldToZMapTransform" overides the settings done by EMeshToZMapConverter, EMeshToZMapConverter and EMeshToZMapConverter.
That E3DTransformMatrix transform expresses how the world positions are transformed to the EZMap space.
The matrix must be a rigid transformation (translation and rotation only).
The resolution and scales of the ZMap are defined by EMeshToZMapConverter, EMeshToZMapConverter and EMeshToZMapConverter methods.

SetZMaptoWorldTransform

Explicitly sets the ZMap to World transformation.
"SetZMaptoWorldTransform" overides the settings done by EMeshToZMapConverter, EMeshToZMapConverter and EMeshToZMapConverter.
That E3DTransformMatrix transform expresses how the EZMap positions are transformed to the World space.
The matrix must be a rigid transformation (translation and rotation only).
The resolution and scales of the World are defined by EMeshToZMapConverter, EMeshToZMapConverter and EMeshToZMapConverter methods.

UnsetMapSize

Unsets the resolution. Lets the conversion decides for the optimum resolution, depending on the projected point cloud and pixel scale.

UnsetMapXYResolution

Unsets the X and Y resolutions. Lets the conversion decide the optimal resolution, depending on the projected point cloud and EZMap size.

| | |
|---|---|
| UnsetMapZResolution | Unsets the ZMap Z resolution. Lets the conversion decide the optimal Z resolution. |
| UnsetOrigin | Lets the conversion process decides for the EZMap origin position (based on projected point cloud on the reference plane). Use EMeshToZMapConverter to enable and choose the ZMap origin. |
| UnsetWorldToZMapTransform | Disables the explicit world to ZMap transformation, set with EMeshToZMapConverter. |

# EMeshToZMapConverter::Convert

Computes a EZMap from a world space EMesh. The value of the pixels of the ZMap are the distance between the 3D triangles and the reference plane. Various options can be set with methods EMeshToZMapConverter, EMeshToZMapConverter, EMeshToZMapConverter::SetMapSize, ...

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Convert(
 const EMesh& obj,
 EZMap8& zmap
 )
void Convert(
 const EMesh& obj,
 EZMap16& zmap
 )
```

```
void Convert(
 const EMesh& obj,
 EZMap32f& zmap
 )
```

### Parameters

*obj*

The input 3D mesh.

*zmap*

The generated ZMap in 8, 16 or 32 bits format.

# EMeshToZMapConverter::EMeshToZMapConverter

Creates a EMeshToZMapConverter object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EMeshToZMapConverter(
 )
void EMeshToZMapConverter(
 const EMeshToZMapConverter& other
 )
```

### Parameters

*other*

Reference to the EMeshToZMapConverter object used for the initialization.

# EMeshToZMapConverter::EnableFillMode

Enables or disables fill mode. Fill mode parameters are defined by method EMeshToZMapConverter::SetFillMode.
Fill mode is enabled by default. If fill mode is disable, undefined pixels may remain in the EZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void EnableFillMode(
 bool state
 )
```

## Parameters

*state*

Set to true to enable fill mode.

# EMeshToZMapConverter::GetExtension

# EMeshToZMapConverter::SetExtension

Sets a metric value used to enlarge the point cloud 3D domain.
That value affects X,Y and Z directions and can be used to generate an EZMap with borders of undefined pixels.
Default value is **0**, which means a ZMap without border.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetExtension() const
void SetExtension(float size)
```

# EMeshToZMapConverter::GetFillUndefinedPixelsDirection

Gets the undefined pixel fill direction (see EFillUndefinedPixelsDirection).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsDirection
GetFillUndefinedPixelsDirection() const
```

# EMeshToZMapConverter::GetFillUndefinedPixelsMethod

Gets the undefined pixel fill method (see EFillUndefinedPixelsMethod).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsMethod
GetFillUndefinedPixelsMethod() const
```

# EMeshToZMapConverter::IsFillModeEnabled

Tells if the fill mode is enabled or not.
Use EMeshToZMapConverter::EnableFillMode to toggle the fill mode and EMeshToZMapConverter::SetFillMode to set the filling parameters.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
bool IsFillModeEnabled(
 )
```

# EMeshToZMapConverter::Load

Loads the converter configuration. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Load(
 ESerializer* serializer
 )
```

## Parameters

*serializer*

Pointer to the ESerializer created for reading.

# EMeshToZMapConverter::GetMapHeight

Gets the required height (number of pixels) of the generated EZMap.
By default, the required size is not set.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
int GetMapHeight() const
```

# EMeshToZMapConverter::GetMapWidth

Gets the required width (number of pixels) of the generated EZMap.
By default, the required size is not set.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
int GetMapWidth() const
```

# EMeshToZMapConverter::GetMapXResolution

Gets the resolution of the EZMap pixels along the X axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetMapXResolution() const
```

# EMeshToZMapConverter::GetMapYResolution

Gets the resolution of the EZMap pixels along the Y axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetMapYResolution() const
```

# EMeshToZMapConverter::GetMapZResolution

# EMeshToZMapConverter::SetMapZResolution

Gets/sets the EZMap Z resolution, in world space units per gray value.
The resolution is used to compute the transformation of the world Z position to an integer 8, 16 or 32 bits pixel value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetMapZResolution() const
void SetMapZResolution(float scale)
```

# EMeshToZMapConverter::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
EMeshToZMapConverter& operator=(
 const EMeshToZMapConverter& other
 )
```

### Parameters

*other*

Reference to the EMeshToZMapConverter object used for the assignment.

# EMeshToZMapConverter::operator==

Comparison operator

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool operator==(
 const EMeshToZMapConverter& other
 )
```

### Parameters

*other*

Reference to the EMeshToZMapConverter object used for the comparison.

# EMeshToZMapConverter::GetOrientationVector

# EMeshToZMapConverter::SetOrientationVector

Sets an explicit orientation for the EZMap.
Overides the orientation mode given by the method EMeshToZMapConverter.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPoint GetOrientationVector() const
void SetOrientationVector(const E3DPoint& direction)
```

### Remarks

The direction should be an E3DPoint representing the expected direction of the X (width) axis of the ZMap.
That direction will be used after projection on the reference plane normal.
That direction must NOT be aligned with the reference plane normal.

# EMeshToZMapConverter::GetOrientationVectorMode

# EMeshToZMapConverter::SetOrientationVectorMode

Chooses the EZMap orientation from a list of pre-defined axis, automatic mode or user defined vector.
Use EMeshToZMapConverter to set an explicit orientation vector for the ZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
Euresys::Open_eVision_2_10::Easy3D::EZMapOrientationVectorMode GetOrientationVectorMode
() const
void SetOrientationVectorMode(Euresys::Open_eVision_2_
10::Easy3D::EZMapOrientationVectorMode mode)
```

### Remarks

Choose between Automatic mode (default), world space axis or explicit user defined vector (see EZMapOrientationVectorMode).

# EMeshToZMapConverter::GetOrigin

# EMeshToZMapConverter::SetOrigin

Chooses the EZMap origin. It is the 3D world position used as origin of the ZMap upper left pixel (0,0).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPoint GetOrigin() const
void SetOrigin(const E3DPoint& position)
```

## Remarks

That position will be projected on the reference plane.
To let the conversion chooses for the origin, call EMeshToZMapConverter::UnsetOrigin.

# EMeshToZMapConverter::GetReferencePlane

# EMeshToZMapConverter::SetReferencePlane

Sets the E3DPlane reference plane.
The resulting EZMap is the distance of the 3D points above that plane.
3D points below the reference plane are discarded.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPlane GetReferencePlane() const
```

```
void SetReferencePlane(const E3DPlane& plane)
```

# EMeshToZMapConverter::GetReferencePlaneMode

# EMeshToZMapConverter::SetReferencePlaneMode

Sets an axis aligned reference plane.
Overides the explicit reference plane given by the method EMeshToZMapConverter.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
Euresys::Open_eVision_2_10::Easy3D::EZMapReferencePlaneMode GetReferencePlaneMode()
const

void SetReferencePlaneMode(Euresys::Open_eVision_2_10::Easy3D::EZMapReferencePlaneMode
mode)
```

## Remarks

Choose between X, Y or Z reference plane (see EZMapReferencePlaneMode).
The plane offset is set automatically on the point cloud lowest 3D point.

# EMeshToZMapConverter::Save

Saves the converter configuration. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

Pointer to the ESerializer created for writing.

# EMeshToZMapConverter::SetFillMode

Inpainting options used to fill the "holes" in the EZMap. A hole exists when no 3D point is projected at that pixel position in the ZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetFillMode(
 Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsDirection direction,
 Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsMethod method
 )
```

### Parameters

*direction*

Direction in which the undefined pixels are filled in a depthmap from EFillUndefinedPixelsDirection

*method*

Which values used to fill the undefined pixels in a depthmap from EFillUndefinedPixelsMethod

# EMeshToZMapConverter::SetMapSize

Sets the required size of the generated EZMap; expressed in number of pixels for width and height dimensions.
By default, the required size is not set.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SetMapSize(
 int width,
 int height
 )
```

### Parameters

*width*

The required width for the Generated ZMap.

*height*

The required height for the Generated ZMap.

# EMeshToZMapConverter::SetMapXYResolution

Sets the resolution (possibly anisotropic) of the EZMap pixels along the X and Y axes, in world space units per pixel (e.g mm/pixel).
The resolution is used to compute the ZMap size (width and height), depending on the projected point cloud on the reference plane.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SetMapXYResolution(
 float resolution
 )

void SetMapXYResolution(
 float resolutionX,
 float resolutionY
 )
```

### Parameters

*resolution*

The resolution for the isotropic case.

*resolutionX*

The resolution for the X axis.

*resolutionY*

The resolution for the Y axis.

### Remarks

The isotropic scale, for X and Y axis is in metric world units.

# EMeshToZMapConverter::UnsetMapSize

Unsets the resolution. Lets the conversion decides for the optimum resolution, depending on the projected point cloud and pixel scale.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void UnsetMapSize(
 )
```

# EMeshToZMapConverter::UnsetMapXYResolution

Unsets the X and Y resolutions. Lets the conversion decide the optimal resolution, depending on the projected point cloud and EZMap size.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void UnsetMapXYResolution(
 )
```

# EMeshToZMapConverter::UnsetMapZResolution

Unsets the ZMap Z resolution. Lets the conversion decide the optimal Z resolution.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void UnsetMapZResolution(
 )
```

# EMeshToZMapConverter::UnsetOrigin

Lets the conversion process decides for the EZMap origin position (based on projected point cloud on the reference plane).
Use EMeshToZMapConverter to enable and choose the ZMap origin.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void UnsetOrigin(
 )
```

# EMeshToZMapConverter::UnsetWorldToZMapTransform

Disables the explicit world to ZMap transformation, set with EMeshToZMapConverter.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void UnsetWorldToZMapTransform(
 )
```

# EMeshToZMapConverter::GetWorldToZMapTransform

# EMeshToZMapConverter::SetWorldToZMapTransform

Explicitly sets the world to ZMap transformation.
"SetWorldToZMapTransform" overides the settings done by EMeshToZMapConverter, EMeshToZMapConverter and EMeshToZMapConverter.
That E3DTransformMatrix transform expresses how the world positions are transformed to the EZMap space.
The matrix must be a rigid transformation (translation and rotation only).
The resolution and scales of the ZMap are defined by EMeshToZMapConverter, EMeshToZMapConverter and EMeshToZMapConverter methods.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix GetWorldToZMapTransform() const
void SetWorldToZMapTransform(const E3DTransformMatrix& matrix)
```

## EMeshToZMapConverter::GetZMaptoWorldTransform

## EMeshToZMapConverter::SetZMaptoWorldTransform

Explicitly sets the ZMap to World transformation.
"SetZMaptoWorldTransform" overides the settings done by EMeshToZMapConverter, EMeshToZMapConverter and EMeshToZMapConverter.
That E3DTransformMatrix transform expresses how the EZMap positions are transformed to the World space.
The matrix must be a rigid transformation (translation and rotation only).
The resolution and scales of the World are defined by EMeshToZMapConverter, EMeshToZMapConverter and EMeshToZMapConverter methods.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix GetZMaptoWorldTransform() const
void SetZMaptoWorldTransform(const E3DTransformMatrix& matrix)
```

# 4.118. EMovingAverage Class

Temporal integration of a number of images to reduce noise.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Average | Performs averaging of the source image with the preceding ones, and computes the de-noised image. |
| EMovingAverage | Constructs an EMovingAverage object. |
| GetSize | Queries a moving average context for the current parameter settings. |
| GetSrcImage | Returns the image in which you must store the next image to be averaged. |
| Reset | Restarts the average process as if no image had ever been handed to the EMovingAverage object. |
| SetSize | Initializes a moving average context with appropriate parameters. |

E

M

# ovingAverage::Average

Performs averaging of the source image with the preceding ones, and computes the de-noised image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Average(
 EROIBW8* destinationImage
 )

void Average(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage
 )
```

### Parameters

*destinationImage*

Pointer to the destination image.

*sourceImage*

Pointer to the source image.

### Remarks

The overload with only the destinationImage may be used only when the buffer allocation scheme has been set to internal (see EMovingAverage::SetSize or EMovingAverage::EMovingAverage)

# EMovingAverage::EMovingAverage

Constructs an EMovingAverage object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EMovingAverage(
 const EMovingAverage& other
 )

void EMovingAverage(
 )
```

```
void EMovingAverage(
 OEV_UINT32 period,
 OEV_INT32 width,
 OEV_INT32 height,
 BOOL internalAllocationScheme
 )
```

### Parameters

*other*

-

*period*

Number of images on which to integrate. A power of 2 is recommended.

*width*

Image width (all images used for averaging must be of the same size).

*height*

Image height (all images used for averaging must be of the same size).

*internalAllocationScheme*

Buffer allocation scheme. When **TRUE**, the moving average context provides the image to be acquired into (see member EMovingAverage::SrcImage).

### Remarks

The default constructor constructs a void moving average context. A void moving average context has no internal buffers allocated and cannot be used for integration. Use the EMovingAverage::SetSize member after construction, or the initializing constructor instead. The sizing constructor constructs and initializes a moving average context.

# EMovingAverage::GetSize

Queries a moving average context for the current parameter settings.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void GetSize(
 OEV_UINT32& numberOfImages,
 OEV_INT32& imageWidth,
 OEV_INT32& imageHeight,
 BOOL& internalAllocationScheme
 )
```

### Parameters

*numberOfImages*

Number of images on which to integrate.

*imageWidth*

Image width (all images used for averaging must be of the same size).

*imageHeight*

Image height (all images used for averaging must be of the same size).

*internalAllocationScheme*

Buffer allocation scheme. When **TRUE**, the moving average context provides the image to be acquired into (see member EMovingAverage::SrcImage).

# EMovingAverage::Reset

Restarts the average process as if no image had ever been handed to the EMovingAverage object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Reset(
 )
```

### Remarks

The behavior is thus the same as after a EMovingAverage::SetSize operation.

# EMovingAverage::SetSize

Initializes a moving average context with appropriate parameters.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetSize(
 OEV_UINT32 numberOfImages,
 OEV_INT32 imageWidth,
 OEV_INT32 imageHeight,
 BOOL internalAllocationScheme
 )
```

### Parameters

*numberOfImages*

Number of images on which to integrate. A power of 2 is recommended.

*imageWidth*

Image width.

*imageHeight*

Image height.

*internalAllocationScheme*

Buffer allocation scheme. When **TRUE**, the moving average context provides the image to be acquired into (see member EMovingAverage::SrcImage).

# EMovingAverage::GetSrcImage

Returns the image in which you must store the next image to be averaged.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageBW8* GetSrcImage()
```

### Remarks

This method may be used only when the buffer allocation scheme has been set to internal (see EMovingAverage::SetSize or EMovingAverage::EMovingAverage)

# 4.119. EObject Class

This class represents an object (blob) in an encoded image.

### Remarks

This class inherits from the ECodedElement class and provides additional methods to access the holes of a particular object.

The extraction of the holes is lazy. This means that the holes are not computed before they get accessed. For this reason, the first access to the holes is slower than the subsequent accesses. On the other hand, the applications that do not make use of the holes are not penalized by the cost of hole extraction.

**Base Class:** ECodedElement

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| GetHole | Returns a specified hole in the object. |
| GetHoleCount | Returns the number of holes in the object. |

## EObject::GetHole

Returns a specified hole in the object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EHole& GetHole(
 OEV_UINT32 index
 )
```

### Parameters

*index*

The index of the hole of interest.

## EObject::GetHoleCount

Returns the number of holes in the object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetHoleCount()
```

# 4.120. EObjectBasedCalibrationGenerator Class

Represents an object-based 3D calibration generator.
The class performs the computation of a calibration model based on the scan of the reference object.

**Base Class:** ECalibrationGenerator

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| Compute | Computes an EObjectBasedCalibrationModel from the EDepthMap of the calibration object. |
| EObjectBasedCalibrationGenerator | Creates a EObjectBasedCalibrationGenerator. |
| GetCalibrationObjectScaleX | Returns the X axis scale of the calibration object. |
| GetCalibrationObjectScaleY | Returns the Y axis scale of the calibration object. |
| GetCalibrationObjectScaleZ | Returns the Z axis scale of the calibration object. |
| GetCalibrationObjectSizeA | Returns the 'A' size of the calibration object. |
| GetCalibrationObjectSizeB | Returns the 'B' size of the calibration object. |
| GetCalibrationObjectSizeC | Returns the 'C' size of the calibration object. |
| GetCalibrationObjectType | Gets the EObjectBasedCalibrationType used for the computation of the EObjectBasedCalibrationModel.<br>The type of the object used for the calibration is **E3DObjectBasedCalibrationType_NotDefined** by default. |

| | |
|---|---|
| GetNumCalibrationPasses | Sets/Gets the number of calibration passes.<br>Each passes will refine the calibration model but the computation will be longer.<br>The number of passes is **1** by default. |
| GetPrecisionVsSpeedTradeOff | Sets/Gets the trade-off between precision and speed for the calibration.<br>The Precision vs speed trade-off mode from EObjectBasedCalibrationPrecisionVsSpeedTradeOff is **EObjectBasedCalibrationPrecisionVsSpeedTradeOff_Balanced** by default. |
| GetRangeX | Sets/Gets the 'X' axis range for the calibration object detection in the EDepthMap.<br>If max value is smaller than min value, then max will not be used, we use depth map **width - 1** instead.<br>If min value is smaller than 0, then min will not be used, we use **0** instead. |
| GetRangeY | Sets/Gets the 'Y' axis range for the calibration object detection in the depth map.<br>If max value is smaller than min value, then max will not be used, we use depth map **height - 1** instead.<br>If min value is smaller than 0, then min will not be used, we use **0** instead. |
| GetRangeZ | Sets/Gets the 'Z' axis range for the calibration object detection in the depth map.<br>if max value is smaller than min value, then max will not be used, we use the maximum float value instead.<br>if min value is smaller than 0, then min will not be used, we use **0** instead. |

| | |
|---|---|
| Load | Loads the parameters used by the object based calibration generator. |
| operator= | Assignment operator. |
| Save | Saves the parameters used by the object based calibration generator. |
| SetCalibrationObjectScale | Sets the scale of your target calibration model compared to a reference model.<br>Refer to the user guide for the EObjectBasedCalibrationModel reference design.<br>Use that value to set the unit of the calibrated positions in the point cloud.<br>The scale will affect the world coordinates after conversion of the EDepthMap to EPointCloud.<br>For example, if "1 reference unit" is equal to "10 millimeters", set "10" as scale value.<br>Then applying the calibration will produce results in millimeters.<br>Changing these values affect the calibration object sizes defined by EObjectBasedCalibrationGenerator::SetCalibrationObjectType. |
| SetCalibrationObjectType | Sets the EObjectBasedCalibrationType used for the computation of the EObjectBasedCalibrationModel.<br>The type of the object used for the calibration is **E3DObjectBasedCalibrationType_NotDefined** by default. |
| SetNumCalibrationPasses | Sets/Gets the number of calibration passes.<br>Each passes will refine the calibration model but the computation will be longer.<br>The number of passes is **1** by default. |

| | |
|---|---|
| SetPrecisionVsSpeedTradeOff | Sets/Gets the trade-off between precision and speed for the calibration.<br>The Precision vs speed trade-off mode from EObjectBasedCalibrationPrecisionVsSpeedTradeOff is **EObjectBasedCalibrationPrecisionVsSpeedTradeOff_Balanced** by default. |
| SetRangeX | Sets/Gets the 'X' axis range for the calibration object detection in the EDepthMap.<br>If max value is smaller than min value, then max will not be used, we use depth map **width - 1** instead.<br>If min value is smaller than 0, then min will not be used, we use **0** instead. |
| SetRangeY | Sets/Gets the 'Y' axis range for the calibration object detection in the depth map.<br>If max value is smaller than min value, then max will not be used, we use depth map **height - 1** instead.<br>If min value is smaller than 0, then min will not be used, we use **0** instead. |
| SetRangeZ | Sets/Gets the 'Z' axis range for the calibration object detection in the depth map.<br>if max value is smaller than min value, then max will not be used, we use the maximum float value instead.<br>if min value is smaller than 0, then min will not be used, we use **0** instead. |

# EObjectBasedCalibrationGenerator::GetCalibrationObjectScaleX

Returns the X axis scale of the calibration object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetCalibrationObjectScaleX() const
```

# EObjectBasedCalibrationGenerator::GetCalibrationObjectScaleY

Returns the Y axis scale of the calibration object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetCalibrationObjectScaleY() const
```

# EObjectBasedCalibrationGenerator::GetCalibrationObjectScaleZ

Returns the Z axis scale of the calibration object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetCalibrationObjectScaleZ() const
```

# EObjectBasedCalibrationGenerator::GetCalibrationObjectSizeA

Returns the 'A' size of the calibration object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetCalibrationObjectSizeA() const
```

# EObjectBasedCalibrationGenerator::GetCalibrationObjectSizeB

Returns the 'B' size of the calibration object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetCalibrationObjectSizeB() const
```

# EObjectBasedCalibrationGenerator::GetCalibrationObjectSizeC

Returns the 'C' size of the calibration object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
float GetCalibrationObjectSizeC() const
```

# EObjectBasedCalibrationGenerator::Compute

Computes an EObjectBasedCalibrationModel from the EDepthMap of the calibration object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EObjectBasedCalibrationModel Compute(
 const EDepthMap8& dm
 )
EObjectBasedCalibrationModel Compute(
 const EDepthMap16& dm
 )
EObjectBasedCalibrationModel Compute(
 const EDepthMap32f& dm
 )
```

## Parameters

*dm*

The input depth map of the calibration object.

# EObjectBasedCalibrationGenerator::EObjectBasedCalibrationGenerator

Creates a EObjectBasedCalibrationGenerator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void EObjectBasedCalibrationGenerator(
 )

void EObjectBasedCalibrationGenerator(
 const EObjectBasedCalibrationGenerator& other
 )
```

### Parameters

*other*

The other EObjectBasedCalibrationGenerator.

# EObjectBasedCalibrationGenerator::GetCalibrationObjectType

Gets the EObjectBasedCalibrationType used for the computation of the EObjectBasedCalibrationModel.
The type of the object used for the calibration is **E3DObjectBasedCalibrationType_NotDefined** by default.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
Euresys::Open_eVision_2_10::Easy3D::EObjectBasedCalibrationType GetCalibrationObjectType
(
 )
```

# EObjectBasedCalibrationGenerator::Load

Loads the parameters used by the object based calibration generator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer.

# EObjectBasedCalibrationGenerator::GetNumCalibrationPasses

# EObjectBasedCalibrationGenerator::SetNumCalibrationPasses

Sets/Gets the number of calibration passes.
Each passes will refine the calibration model but the computation will be longer.
The number of passes is **1** by default.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
int GetNumCalibrationPasses() const
void SetNumCalibrationPasses(int numPasses)
```

# EObjectBasedCalibrationGenerator::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EObjectBasedCalibrationGenerator& operator=(
 const EObjectBasedCalibrationGenerator& other
 )
```

### Parameters

*other*

The other EObjectBasedCalibrationGenerator.

# EObjectBasedCalibrationGenerator::GetPrecisionVsSpeedTradeOff

# EObjectBasedCalibrationGenerator::SetPrecisionVsSpeedTradeOff

Sets/Gets the trade-off between precision and speed for the calibration.
The Precision vs speed trade-off mode from EObjectBasedCalibrationPrecisionVsSpeedTradeOff is **EObjectBasedCalibrationPrecisionVsSpeedTradeOff_Balanced** by default.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
Euresys::Open_eVision_2_10::Easy3D::EObjectBasedCalibrationPrecisionVsSpeedTradeOff
GetPrecisionVsSpeedTradeOff() const

void SetPrecisionVsSpeedTradeOff(Euresys::Open_eVision_2_
10::Easy3D::EObjectBasedCalibrationPrecisionVsSpeedTradeOff tradeOff)
```

# EObjectBasedCalibrationGenerator::GetRangeX

# EObjectBasedCalibrationGenerator::SetRangeX

Sets/Gets the 'X' axis range for the calibration object detection in the EDepthMap.
If max value is smaller than min value, then max will not be used, we use depth map **width - 1** instead.
If min value is smaller than 0, then min will not be used, we use **0** instead.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
EIntegerRange GetRangeX() const
void SetRangeX(const EIntegerRange& Range_X)
```

# EObjectBasedCalibrationGenerator::GetRangeY

# EObjectBasedCalibrationGenerator::SetRangeY

Sets/Gets the 'Y' axis range for the calibration object detection in the depth map.
If max value is smaller than min value, then max will not be used, we use depth map **height - 1** instead.
If min value is smaller than 0, then min will not be used, we use **0** instead.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
EIntegerRange GetRangeY() const
void SetRangeY(const EIntegerRange& Range_Y)
```

# EObjectBasedCalibrationGenerator::GetRangeZ

# EObjectBasedCalibrationGenerator::SetRangeZ

Sets/Gets the 'Z' axis range for the calibration object detection in the depth map.
if max value is smaller than min value, then max will not be used, we use the maximum float value instead.
if min value is smaller than 0, then min will not be used, we use **0** instead.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EFloatRange GetRangeZ() const
void SetRangeZ(const EFloatRange& Range_Z)
```

# EObjectBasedCalibrationGenerator::Save

Saves the parameters used by the object based calibration generator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 ESerializer* serializer
 )
```

## Parameters

*serializer*

The ESerializer.

# EObjectBasedCalibrationGenerator::SetCalibrationObjectScale

Sets the scale of your target calibration model compared to a reference model.
Refer to the user guide for the EObjectBasedCalibrationModel reference design.
Use that value to set the unit of the calibrated positions in the point cloud.
The scale will affect the world coordinates after conversion of the EDepthMap to EPointCloud.
For example, if "1 reference unit" is equal to "10 millimeters", set "10" as scale value.
Then applying the calibration will produce results in millimeters. Changing these values affect the calibration object sizes defined by EObjectBasedCalibrationGenerator::SetCalibrationObjectType.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetCalibrationObjectScale(
 float scale
 )
void SetCalibrationObjectScale(
 float scaleX,
 float scaleY,
 float scaleZ
 )
```

## Parameters

*scale*

Sets the same scale on axis X, Y and Z relative to the calibration object reference size.

*scaleX*

Sets the scale on X axis relative to the calibration object reference size.

*scaleY*

Sets the scale on Y axis relative to the calibration object reference size.

*scaleZ*

Sets the scale on Z axis relative to the calibration object reference size.

# EObjectBasedCalibrationGenerator::SetCalibrationObjectType

Sets the EObjectBasedCalibrationType used for the computation of the EObjectBasedCalibrationModel.
The type of the object used for the calibration is **E3DObjectBasedCalibrationType_NotDefined** by default.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetCalibrationObjectType(
 Euresys::Open_eVision_2_10::Easy3D::EObjectBasedCalibrationType type
 )

void SetCalibrationObjectType(
 Euresys::Open_eVision_2_10::Easy3D::EObjectBasedCalibrationType type,
 float sizeA,
 float sizeB,
 float sizeC
 )
```

## Parameters

*type*

Sets the type of object calibration to detect in the EDepthMap.

*sizeA*

Sets the size 'A' of object calibration (**1** by default).

*sizeB*

Sets the size 'B' of object calibration (**1** by default).

*sizeC*

Sets the size 'C' of object calibration (**1** by default).

## Remarks

'sizeA', 'sizeB', and 'sizeC' values are in metric unit. The resulting point cloud positions after applying the calibration will follow the same metric unit. Refer to the Open eVision user guide, Easy3D calibration section, for the definition of 'A', 'B' and 'C' dimensions.

# 4.121. EObjectBasedCalibrationModel Class

EObjectBasedCalibrationModel is an Easy3D calibration model built from a scan of a reference object.

**Base Class:** ECalibrationModel

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| EObjectBasedCalibrationModel | Creates an EObjectBasedCalibrationModel. |
| GetCalibrationError | Returns the estimate of the calibration error (in metric units). |
| GetCalibrationRelativeError | Returns the estimate of the calibration error (in relative units). |
| GetType | Returns the type of calibration model, see ECalibrationType. |
| IsInitialized | Returns true if the model is initialized. |
| Load | Loads the model. The given ESerializer must have been created for reading. |
| operator= | Assignment operator. |

| | |
|---|---|
| Save | Saves the model. The given ESerializer must have been created for writing. |

# EObjectBasedCalibrationModel::GetCalibrationError

Returns the estimate of the calibration error (in metric units).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetCalibrationError() const
```

# EObjectBasedCalibrationModel::GetCalibrationRelativeError

Returns the estimate of the calibration error (in relative units).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetCalibrationRelativeError() const
```

# EObjectBasedCalibrationModel::EObjectBasedCalibrationModel

Creates an EObjectBasedCalibrationModel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EObjectBasedCalibrationModel(
 )
void EObjectBasedCalibrationModel(
 const EObjectBasedCalibrationModel& other
 )
```

### Parameters

*other*

Another EObjectBasedCalibrationModel.

# EObjectBasedCalibrationModel::IsInitialized

Returns true if the model is initialized.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool IsInitialized(
 )
```

# EObjectBasedCalibrationModel::Load

Loads the model. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# EObjectBasedCalibrationModel::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EObjectBasedCalibrationModel& operator=(
 const EObjectBasedCalibrationModel& other
 )
```

### Parameters

*other*

Another EObjectBasedCalibrationModel.

# EObjectBasedCalibrationModel::Save

Saves the model. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

## EObjectBasedCalibrationModel::GetType

Returns the type of calibration model, see ECalibrationType.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
Euresys::Open_eVision_2_10::Easy3D::ECalibrationType GetType() const
```

# 4.122. EObjectRunsIterator Class

Iterator to the runs of a coded element.

### Remarks

This class is responsible for the sequential access to the individual runs of a coded element.

A run is defined as a maximal sequence of consecutive pixels on the same run, that all belong to the same coded element.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EObjectRunsIterator | Constructs an iterator to the runs of a coded element. |
| First | Rewinds to the first run of the coded element. |
| GetEndX | Returns the abscissa (X-coordinate) of the rightmost pixel of the run the iterator is currently over. |
| GetLength | Returns the lengths of the run the iterator is currently over. |
| GetStartX | Returns the abscissa (X-coordinate) of the leftmost pixel of the run the iterator is currently over. |
| GetY | Returns the ordinate (Y-coordinate) of the run the iterator is currently over. |
| IsDone | Tests whether all the runs have been spanned. |
| Next | Advance to the next run in the iterator. |
| operator= | Assignment operator. |

# EObjectRunsIterator::GetEndX

Returns the abscissa (X-coordinate) of the rightmost pixel of the run the iterator is currently over.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetEndX() const
```

## Remarks

An exception is thrown if the iterator has reached its end. Use EObjectRunsIterator::IsDone to check this condition.

# EObjectRunsIterator::EObjectRunsIterator

Constructs an iterator to the runs of a coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EObjectRunsIterator(
 )

void EObjectRunsIterator(
 const ECodedElement& codedElement
 )

void EObjectRunsIterator(
 const EObjectRunsIterator& other
 )
```

## Parameters

*codedElement*

The coded element of interest, in the case the iterator is to be constructed from a given coded element.

*other*

The iterator to be copied, in the case of the copy constructor.

# EObjectRunsIterator::First

Rewinds to the first run of the coded element.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void First(
 )
```

# EObjectRunsIterator::GetIsDone

Tests whether all the runs have been spanned.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
bool IsDone() const
```

# EObjectRunsIterator::GetLength

Returns the lengths of the run the iterator is currently over.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetLength() const
```

### Remarks

An exception is thrown if the iterator has reached its end. Use EObjectRunsIterator::IsDone to check this condition.

# EObjectRunsIterator::Next

Advance to the next run in the iterator.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Next(
 )
```

### Remarks

An exception is thrown if the iterator has reached its end. Use EObjectRunsIterator::IsDone to check this condition.

# EObjectRunsIterator::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
EObjectRunsIterator& operator=(
 const EObjectRunsIterator& other
 )
```

### Parameters

*other*

The iterator to be copied.

# EObjectRunsIterator::GetStartX

Returns the abscissa (X-coordinate) of the leftmost pixel of the run the iterator is currently over.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetStartX() const
```

### Remarks

An exception is thrown if the iterator has reached its end. Use EObjectRunsIterator::IsDone to check this condition.

# EObjectRunsIterator::GetY

Returns the ordinate (Y-coordinate) of the run the iterator is currently over.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetY() const
```

### Remarks

An exception is thrown if the iterator has reached its end. Use EObjectRunsIterator::IsDone to check this condition.

# 4.123. EObjectSelection Class

This container class handles the selection of a subset of coded elements taken from a coded image.

### Remarks

This class provides methods to perform a selection of objects or holes and to retrieve the features of the coded elements in the collection.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Add | Add a coded element to the selection. |
| AddHole | Adds a single hole of a coded image. |
| AddHoles | Adds all the holes contained in an object, in a layer or in a coded image. |
| AddHolesOfSelectedObjects | Adds the holes of all the objects that are currently selected. |
| AddLayer | Adds all the objects of a single layer in a coded image. |
| AddObject | Adds a single object of a layer in a coded image. |

| | |
|---|---|
| AddObjects | Adds all the objects of all the layers of a given coded image. |
| AddObjectsUsingFloatFeature | Selects the objects that fullfil a condition on a floating-point feature. |
| AddObjectsUsingIntegerFeature | Selects the objects that fullfil a condition on an integer feature. |
| AddObjectsUsingRectangle | Adds all the objects of a coded image whose location fullfil a criterion based on a rectangle. |
| AddObjectsUsingUnsignedIntegerFeature | Selects the objects that fullfil a condition on an unsigned integer feature. |
| AddObjectUsingPosition | Adds the object of a coded image that lies at a particular location. |
| Clear | Remove all the coded elements that are contained in the selection. |
| ClearFeatureCache | Clears the internal cache for the computed features. |
| EObjectSelection | - |
| FeatureAverage | Computes the average of the values of the given feature across the selection. |

| | |
|---|---|
| FeatureDeviation | Computes the standard deviation of the values of the given feature across the selection. |
| FeatureVariance | Computes the variance of the values of the given feature across the selection. |
| FloatFeatureMaximum | Computes the maximum value of the given feature across the selection. |
| FloatFeatureMinimum | Computes the minimum value of the given feature across the selection. |
| GetAttachedImage | Sets the attached image for computing the features that depend on an image. |
| GetElement | Index-based access to the coded elements of the selection. |
| GetElementCount | Returns the number of coded elements selection. |
| GetFeretAngle | Angle of the Feret box (in the current angle units). |
| GetFloatFeature | Returns the value of a floating-point feature for a selected coded element. |

| | |
|---|---|
| GetIndexOfElement | Retrieves the index of a given coded element in the selection. |
| GetIntegerFeature | Returns the value of an integer feature for a selected coded element. |
| GetUnsignedIntegerFeature | Returns the value of an unsigned integer feature for a selected coded element. |
| IntegerFeatureMaximum | Computes the maximum value of the given feature across the selection. |
| IntegerFeatureMinimum | Computes the minimum value of the given feature across the selection. |
| IsSelected | Tests whether a given coded element is present in the selection. |
| Remove | Remove a coded element from the selection. |
| RemoveHole | Removes a single hole of a coded image. |
| RemoveHoles | Removes all the holes contained in an object, in a layer or in a coded image. |
| RemoveLayer | Removes all the objects of a single layer in a coded image. |

| | |
|---|---|
| RemoveObject | Removes a single object of a layer in a coded image. |
| RemoveObjectsUsingRectangle | Removes all the objects of a coded image whose location fullfil a criterion based on a rectangle. |
| RemoveObjectUsingPosition | Removes the object of a coded image that lies at a particular location. |
| RemoveSelectedHoles | Remove all the holes that are currently present in the selection. |
| RemoveUsingFloatFeature | Removes the selected coded elements that fullfil a condition on a floating-point feature. |
| RemoveUsingIntegerFeature | Removes the selected coded elements that fullfil a condition on an unsigned integer feature. |
| RemoveUsingUnsignedIntegerFeature | Removes the selected coded elements that fullfil a condition on an unsigned integer feature. |
| RenderMask | Creates a Flexible Mask from the selection. |
| SetAttachedImage | Sets the attached image for computing the features that depend on an image. |
| SetFeretAngle | Angle of the Feret box (in the current angle units). |

| | |
|---|---|
| Sort | Sorts the selected coded elements according to the value of a feature. |
| UnsignedIntegerFeatureMaximum | Computes the maximum value of the given feature across the selection. |
| UnsignedIntegerFeatureMinimum | Computes the minimum value of the given feature across the selection. |

# EObjectSelection::Add

Add a coded element to the selection.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Add(
 ECodedElement& element
 )
```

## Parameters

*element*

The coded element.

# EObjectSelection::AddHole

Adds a single hole of a coded image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddHole(
 ECodedImage2& codedImage,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex
 )
void AddHole(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex
 )
```

### Parameters

*codedImage*

The coded image.

*objectIndex*

The index of the parent object in the layer.

*holeIndex*

The index of the hole in the parent object.

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, the default layer will be taken into consideration.

### Remarks

This methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

# EObjectSelection::AddHoles

Adds all the holes contained in an object, in a layer or in a coded image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void AddHoles(
 ECodedImage2& codedImage
 )

void AddHoles(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex
 )

void AddHoles(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex
 )
```

### Parameters

*codedImage*

The source coded image.

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, all the layers are taken into consideration.

*objectIndex*

The index of the parent object. If this parameter is left unspecified, all the objects are taken into consideration.

# EObjectSelection::AddHolesOfSelectedObjects

Adds the holes of all the objects that are currently selected.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void AddHolesOfSelectedObjects(
 )
```

# EObjectSelection::AddLayer

Adds all the objects of a single layer in a coded image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddLayer(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex
 )
void AddLayer(
 ECodedImage2& codedImage
 )
```

## Parameters

*codedImage*

The coded image.

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, the default layer will be taken into consideration.

## Remarks

This methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

# EObjectSelection::AddObject

Adds a single object of a layer in a coded image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void AddObject(
 ECodedImage2& codedImage,
 OEV_UINT32 objectIndex
 )

void AddObject(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex
 )
```

### Parameters

*codedImage*

The coded image.

*objectIndex*

The index of the object in the layer.

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, the default layer will be taken into consideration.

### Remarks

This methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

# EObjectSelection::AddObjects

Adds all the objects of all the layers of a given coded image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void AddObjects(
 ECodedImage2& image
 )
```

### Parameters

*image*

The coded image.

# EObjectSelection::AddObjectsUsingFloatFeature

Selects the objects that fullfil a condition on a floating-point feature.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void AddObjectsUsingFloatFeature(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex,
 Euresys::Open_eVision_2_10::EFeature feature,
 float threshold,
 Euresys::Open_eVision_2_10::ESingleThresholdMode mode
 )

void AddObjectsUsingFloatFeature(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex,
 Euresys::Open_eVision_2_10::EFeature feature,
 float lowBound,
 float highBound,
 Euresys::Open_eVision_2_10::EDoubleThresholdMode mode
 )

void AddObjectsUsingFloatFeature(
 ECodedImage2& codedImage,
 Euresys::Open_eVision_2_10::EFeature feature,
 float threshold,
 Euresys::Open_eVision_2_10::ESingleThresholdMode mode
 )

void AddObjectsUsingFloatFeature(
 ECodedImage2& codedImage,
 Euresys::Open_eVision_2_10::EFeature feature,
 float lowBound,
 float highBound,
 Euresys::Open_eVision_2_10::EDoubleThresholdMode mode
 )
```

### Parameters

*codedImage*

The source coded image.

*layerIndex*

The index of the layer of interest. If left unspecified, all the layers are taken into consideration.

*feature*

The feature that serves as a filter.

*threshold*

The single threshold on the feature.

*mode*

Specifies the way the threshold is interpreted.

*lowBound*

The low bound of the range on the feature.

*highBound*

The high bound of the range on the feature.

### Remarks

Most features are floating-point. These methods thus tend to be the most widely used.

# EObjectSelection::AddObjectsUsingIntegerFeature

Selects the objects that fullfil a condition on an integer feature.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddObjectsUsingIntegerFeature(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex,
 Euresys::Open_eVision_2_10::EFeature feature,
 int threshold,
 Euresys::Open_eVision_2_10::ESingleThresholdMode mode
 )
```

```
void AddObjectsUsingIntegerFeature(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex,
 Euresys::Open_eVision_2_10::EFeature feature,
 int lowBound,
 int highBound,
 Euresys::Open_eVision_2_10::EDoubleThresholdMode mode
 )
void AddObjectsUsingIntegerFeature(
 ECodedImage2& codedImage,
 Euresys::Open_eVision_2_10::EFeature feature,
 int threshold,
 Euresys::Open_eVision_2_10::ESingleThresholdMode mode
 )
void AddObjectsUsingIntegerFeature(
 ECodedImage2& codedImage,
 Euresys::Open_eVision_2_10::EFeature feature,
 int lowBound,
 int highBound,
 Euresys::Open_eVision_2_10::EDoubleThresholdMode mode
 )
```

## Parameters

*codedImage*

The source coded image.

*layerIndex*

The index of the layer of interest. If left unspecified, all the layers are taken into consideration.

*feature*

The feature that serves as a filter.

*threshold*

The single threshold on the feature.

*mode*

Specifies the way the threshold is interpreted.

*lowBound*

The low bound of the range on the feature.

*highBound*

The high bound of the range on the feature.

# EObjectSelection::AddObjectsUsingRectangle

Adds all the objects of a coded image whose location fullfil a criterion based on a rectangle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddObjectsUsingRectangle(
 ECodedImage2& codedImage,
 int x,
 int y,
 OEV_UINT32 width,
 OEV_UINT32 height,
 Euresys::Open_eVision_2_10::ERectangleMode mode
 )
void AddObjectsUsingRectangle(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex,
 int x,
 int y,
 OEV_UINT32 width,
 OEV_UINT32 height,
 Euresys::Open_eVision_2_10::ERectangleMode mode
 )
```

## Parameters

*codedImage*

The source coded image.

*x*

The X-coordinate of the top-left corner of the selection rectangle.

*y*

The Y-coordinate of the top-left corner of the selection rectangle.

*width*

The width of the selection rectangle.

*height*

The height of the selection rectangle.

*mode*

The comparison mode with respect to the selection rectangle.

*layerIndex*

If specified, only the specified layer is taken into consideration.

# EObjectSelection::AddObjectsUsingUnsignedIntegerFeature

Selects the objects that fullfil a condition on an unsigned integer feature.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void AddObjectsUsingUnsignedIntegerFeature(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex,
 Euresys::Open_eVision_2_10::EFeature feature,
 OEV_UINT32 threshold,
 Euresys::Open_eVision_2_10::ESingleThresholdMode mode
 )

void AddObjectsUsingUnsignedIntegerFeature(
 ECodedImage2& codedImage,
 Euresys::Open_eVision_2_10::EFeature feature,
 OEV_UINT32 threshold,
 Euresys::Open_eVision_2_10::ESingleThresholdMode mode
 )

void AddObjectsUsingUnsignedIntegerFeature(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex,
 Euresys::Open_eVision_2_10::EFeature feature,
 OEV_UINT32 lowBound,
 OEV_UINT32 highBound,
 Euresys::Open_eVision_2_10::EDoubleThresholdMode mode
 )
```

```
void AddObjectsUsingUnsignedIntegerFeature(
 ECodedImage2& codedImage,
 Euresys::Open_eVision_2_10::EFeature feature,
 OEV_UINT32 lowBound,
 OEV_UINT32 highBound,
 Euresys::Open_eVision_2_10::EDoubleThresholdMode mode
 )
```

### Parameters

*codedImage*

The source coded image.

*layerIndex*

The index of the layer of interest. If left unspecified, all the layers are taken into consideration.

*feature*

The feature that serves as a filter.

*threshold*

The single threshold on the feature.

*mode*

Specifies the way the threshold is interpreted.

*lowBound*

The low bound of the range on the feature.

*highBound*

The high bound of the range on the feature.

# EObjectSelection::AddObjectUsingPosition

Adds the object of a coded image that lies at a particular location.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void AddObjectUsingPosition(
 ECodedImage2& codedImage,
 int x,
 int y
 )
```

### Parameters

*codedImage*

The source coded image.

*x*

The X-coordinate of the object.

*y*

The Y-coordinate of the object.

### Remarks

If no object lies at the specified coordinate, the selection is not changed.

# EObjectSelection::GetAttachedImage

# EObjectSelection::SetAttachedImage

Sets the attached image for computing the features that depend on an image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
const EROIBW8* GetAttachedImage() const
void SetAttachedImage(const EROIBW8* image)
```

### Remarks

An image must be attached before dealing with the following features: EFeature_PixelMin, EFeature_PixelMax, EFeature_WeightedGravityCenterX, EFeature_WeightedGravityCenterY, EFeature_PixelGrayAverage, EFeature_PixelGrayVariance, EFeature_PixelGrayDeviation.

# EObjectSelection::Clear

Remove all the coded elements that are contained in the selection.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Clear(
 )
```

# EObjectSelection::ClearFeatureCache

Clears the internal cache for the computed features.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ClearFeatureCache(
 )
```

## Remarks

This is useful to reduce memory consumption.

# EObjectSelection::GetElementCount

Returns the number of coded elements selection.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetElementCount()
```

# EObjectSelection::EObjectSelection

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EObjectSelection(
 const EObjectSelection& other
 )
void EObjectSelection(
 )
```

### Parameters

*other*

-

# EObjectSelection::FeatureAverage

Computes the average of the values of the given feature across the selection.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float FeatureAverage(
 Euresys::Open_eVision_2_10::EFeature feature
 )
```

### Parameters

*feature*

The feature of interest.

# EObjectSelection::FeatureDeviation

Computes the standard deviation of the values of the given feature across the selection.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float FeatureDeviation(
 Euresys::Open_eVision_2_10::EFeature feature
 )
```

### Parameters

*feature*

The feature of interest.

# EObjectSelection::FeatureVariance

Computes the variance of the values of the given feature across the selection.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float FeatureVariance(
 Euresys::Open_eVision_2_10::EFeature feature
 )
```

### Parameters

*feature*

The feature of interest.

# EObjectSelection::GetFeretAngle

# EObjectSelection::SetFeretAngle

Angle of the Feret box (in the current angle units).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetFeretAngle()

void SetFeretAngle(float angle)
```

### Remarks

A Feret angle must be set for the following features: EFeature_FeretBoxCenterX, EFeature_FeretBoxCenterY, EFeature_FeretBoxWidth, EFeature_FeretBoxHeight.

# EObjectSelection::FloatFeatureMaximum

Computes the maximum value of the given feature across the selection.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float FloatFeatureMaximum(
 Euresys::Open_eVision_2_10::EFeature feature
 )
```

### Parameters

*feature*

The feature of interest.

### Remarks

Most features are floating-point. This method thus tends to be the most widely used.

# EObjectSelection::FloatFeatureMinimum

Computes the minimum value of the given feature across the selection.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float FloatFeatureMinimum(
 Euresys::Open_eVision_2_10::EFeature feature
 )
```

### Parameters

*feature*

The feature of interest.

### Remarks

Most features are floating-point. This method thus tends to be the most widely used.

# EObjectSelection::GetElement

Index-based access to the coded elements of the selection.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ECodedElement& GetElement(
 OEV_UINT32 index
 )
```

### Parameters

*index*

The index of the coded element of interest.

# EObjectSelection::GetFloatFeature

Returns the value of a floating-point feature for a selected coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetFloatFeature(
 OEV_UINT32 index,
 Euresys::Open_eVision_2_10::EFeature feature
 )
```

### Parameters

*index*

The index of the selected coded element.

*feature*

The feature of interest.

### Remarks

Most features are floating-point. This method thus tends to be the most widely used.

# EObjectSelection::GetIndexOfElement

Retrieves the index of a given coded element in the selection.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetIndexOfElement(
 const ECodedElement& element
 )
```

### Parameters

*element*

The coded element of interest.

### Remarks

An exception is thrown if the coded element is not present in the selection.

# EObjectSelection::GetIntegerFeature

Returns the value of an integer feature for a selected coded element.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
int GetIntegerFeature(
 OEV_UINT32 index,
 Euresys::Open_eVision_2_10::EFeature feature
 )
```

### Parameters

*index*

The index of the selected coded element.

*feature*

The feature of interest.

# EObjectSelection::GetUnsignedIntegerFeature

Returns the value of an unsigned integer feature for a selected coded element.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetUnsignedIntegerFeature(
 OEV_UINT32 index,
 Euresys::Open_eVision_2_10::EFeature feature
 )
```

### Parameters

*index*

The index of the selected coded element.

*feature*

The feature of interest.

# EObjectSelection::IntegerFeatureMaximum

Computes the maximum value of the given feature across the selection.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
int IntegerFeatureMaximum(
 Euresys::Open_eVision_2_10::EFeature feature
 )
```

## Parameters

*feature*

The feature of interest.

# EObjectSelection::IntegerFeatureMinimum

Computes the minimum value of the given feature across the selection.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
int IntegerFeatureMinimum(
 Euresys::Open_eVision_2_10::EFeature feature
 )
```

## Parameters

*feature*

The feature of interest.

# EObjectSelection::IsSelected

Tests whether a given coded element is present in the selection.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
bool IsSelected(
 ECodedElement& element
 )

bool IsSelected(
 ECodedImage2& codedImage,
 OEV_UINT32 objectIndex
 )

bool IsSelected(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex
 )

bool IsSelected(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex
 )
```

## Parameters

*element*

The coded element.

*codedImage*

The coded image.

*objectIndex*

The index of the object in the layer of interest.

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, the default layer will be taken into consideration.

*holeIndex*

If specified, the index of the hole in the object. If unspecified, one tests the presence of the object.

### Remarks

This methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

# EObjectSelection::Remove

Remove a coded element from the selection.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Remove(
 ECodedElement& element
 )
```

### Parameters

*element*

The coded element.

# EObjectSelection::RemoveHole

Removes a single hole of a coded image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void RemoveHole(
 ECodedImage2& codedImage,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex
 )
void RemoveHole(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex,
 OEV_UINT32 holeIndex
 )
```

### Parameters

*codedImage*

The coded image.

*objectIndex*

The index of the parent object in the layer.

*holeIndex*

The index of the hole in the parent object.

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, the default layer will be taken into consideration.

### Remarks

This methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

# EObjectSelection::RemoveHoles

Removes all the holes contained in an object, in a layer or in a coded image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void RemoveHoles(
 ECodedImage2& codedImage
 )

void RemoveHoles(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex
 )

void RemoveHoles(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex
 )
```

### Parameters

*codedImage*

The source coded image.

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, all the layers are taken into consideration.

*objectIndex*

The index of the parent object. If this parameter is left unspecified, all the objects are taken into consideration.

# EObjectSelection::RemoveLayer

Removes all the objects of a single layer in a coded image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void RemoveLayer(
 ECodedImage2& codedImage
 )
```

```
void RemoveLayer(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex
 )
```

### Parameters

*codedImage*

The coded image.

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, the default layer will be taken into consideration.

### Remarks

This methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

# EObjectSelection::RemoveObject

Removes a single object of a layer in a coded image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void RemoveObject(
 ECodedImage2& codedImage,
 OEV_UINT32 objectIndex
 )

void RemoveObject(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex,
 OEV_UINT32 objectIndex
 )
```

### Parameters

*codedImage*

The coded image.

*objectIndex*

The index of the object in the layer.

*layerIndex*

The index of the layer of interest. If this parameter is left unspecified, the default layer will be taken into consideration.

### Remarks

This methods throws an exception if no layer index is specified and if, simultaneously, the coded image contains several layers. Indeed, in such a case, no default layer exists.

# EObjectSelection::RemoveObjectsUsingRectangle

Removes all the objects of a coded image whose location fullfil a criterion based on a rectangle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveObjectsUsingRectangle(
 ECodedImage2& codedImage,
 int x,
 int y,
 OEV_UINT32 width,
 OEV_UINT32 height,
 Euresys::Open_eVision_2_10::ERectangleMode mode
 )
void RemoveObjectsUsingRectangle(
 ECodedImage2& codedImage,
 OEV_UINT32 layerIndex,
 int x,
 int y,
 OEV_UINT32 width,
 OEV_UINT32 height,
 Euresys::Open_eVision_2_10::ERectangleMode mode
 )
```

### Parameters

*codedImage*

The source coded image.

*x*

The X-coordinate of the top-left corner of the selection rectangle.

*y*

The Y-coordinate of the top-left corner of the selection rectangle.

*width*

The width of the selection rectangle.

*height*

The height of the selection rectangle.

*mode*

The comparison mode with respect to the selection rectangle.

*layerIndex*

If specified, only the specified layer is taken into consideration.

# EObjectSelection::RemoveObjectUsingPosition

Removes the object of a coded image that lies at a particular location.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveObjectUsingPosition(
 ECodedImage2& codedImage,
 int x,
 int y
 )
```

## Parameters

*codedImage*

The source coded image.

*x*

The X-coordinate of the object.

*y*

The Y-coordinate of the object.

### Remarks

If no object lies at the specified coordinate, the selection is not changed.

# EObjectSelection::RemoveSelectedHoles

Remove all the holes that are currently present in the selection.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveSelectedHoles(
 )
```

# EObjectSelection::RemoveUsingFloatFeature

Removes the selected coded elements that fullfil a condition on a floating-point feature.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveUsingFloatFeature(
 Euresys::Open_eVision_2_10::EFeature feature,
 float threshold,
 Euresys::Open_eVision_2_10::ESingleThresholdMode mode
 )
void RemoveUsingFloatFeature(
 Euresys::Open_eVision_2_10::EFeature feature,
 float low,
 float high,
 Euresys::Open_eVision_2_10::EDoubleThresholdMode mode
 )
```

### Parameters

*feature*

The feature that serves as a filter.

*threshold*

The single threshold on the feature.

*mode*

Specifies the way the threshold is interpreted.

*low*

The low bound of the range on the feature.

*high*

The high bound of the range on the feature.

### Remarks

Most features are floating-point. These methods thus tend to be the most widely used.

# EObjectSelection::RemoveUsingIntegerFeature

Removes the selected coded elements that fullfil a condition on an unsigned integer feature.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveUsingIntegerFeature(
 Euresys::Open_eVision_2_10::EFeature feature,
 int threshold,
 Euresys::Open_eVision_2_10::ESingleThresholdMode mode
 )

void RemoveUsingIntegerFeature(
 Euresys::Open_eVision_2_10::EFeature feature,
 int low,
 int high,
 Euresys::Open_eVision_2_10::EDoubleThresholdMode mode
 )
```

### Parameters

*feature*

The feature that serves as a filter.

*threshold*

The single threshold on the feature.

*mode*

Specifies the way the threshold is interpreted.

*low*

The low bound of the range on the feature.

*high*

The high bound of the range on the feature.

# EObjectSelection::RemoveUsingUnsignedIntegerFeature

Removes the selected coded elements that fullfil a condition on an unsigned integer feature.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveUsingUnsignedIntegerFeature(
 Euresys::Open_eVision_2_10::EFeature feature,
 OEV_UINT32 threshold,
 Euresys::Open_eVision_2_10::ESingleThresholdMode mode
 )
void RemoveUsingUnsignedIntegerFeature(
 Euresys::Open_eVision_2_10::EFeature feature,
 OEV_UINT32 low,
 OEV_UINT32 high,
 Euresys::Open_eVision_2_10::EDoubleThresholdMode mode
 )
```

## Parameters

*feature*

The feature that serves as a filter.

*threshold*

The single threshold on the feature.

*mode*

Specifies the way the threshold is interpreted.

*low*

The low bound of the range on the feature.

*high*

The high bound of the range on the feature.

# EObjectSelection::RenderMask

Creates a Flexible Mask from the selection.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RenderMask(
 EROIBW8& result,
 int offsetX,
 int offsetY
 )
void RenderMask(
 EROIBW8& result
 )
```

## Parameters

*result*

The image in which the generated mask will be stored.

*offsetX*

The X-offset that must be applied to bring the zero X-coordinate in the coded image on the first column of the result image (defaults to zero).

*offsetY*

The Y-offset that must be applied to bring the zero Y-coordinate in the coded image on the first row of the result image (defaults to zero).

## Remarks

The size of the result image will not be changed: It must be properly sized beforehand.

This method generates an exception if several layers are encoded, in which case no default layer exists.

# EObjectSelection::Sort

Sorts the selected coded elements according to the value of a feature.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Sort(
 Euresys::Open_eVision_2_10::EFeature feature
 )

void Sort(
 Euresys::Open_eVision_2_10::EFeature feature,
 Euresys::Open_eVision_2_10::ESortDirection direction
 )
```

### Parameters

*feature*

The feature to sort with.

*direction*

The sorting direction. By default, the features are sorted by increasing values.

# EObjectSelection::UnsignedIntegerFeatureMaximum

Computes the maximum value of the given feature across the selection.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 UnsignedIntegerFeatureMaximum(
 Euresys::Open_eVision_2_10::EFeature feature
 )
```

### Parameters

*feature*

The feature of interest.

## EObjectSelection::UnsignedIntegerFeatureMinimum

Computes the minimum value of the given feature across the selection.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 UnsignedIntegerFeatureMinimum(
 Euresys::Open_eVision_2_10::EFeature feature
 )
```

### Parameters

*feature*

The feature of interest.

# 4.124. EOCR Class

Manages a complete context for the font-dependent printed character reader implemented in EasyOCR.

**Namespace:** Euresys::Open_eVision_2_10

### Methods

| | |
|---|---|
| AddChar | Add a character coordinates to the list. |

| | |
|---|---|
| AddPatternFromImage | Adds a new pattern to the font. |
| BuildObjects | Segments the source image, i.e. detects and labels the objects. |
| CharGetDstX | Returns the abscissa of the lower right corner of a recognized character. |
| CharGetDstY | Returns the ordinate of the lower right corner of a recognized character. |
| CharGetHeight | Returns the height of a recognized character. |
| CharGetOrgX | Returns the abscissa of the upper left corner of a recognized character. |
| CharGetOrgY | Returns the ordinate of the upper left corner of a recognized character. |
| CharGetTotalDstX | Returns the abscissa of the lower right corner of a recognized character. |

| | |
|---|---|
| CharGetTotalDstY | Returns the ordinate of the lower right corner of a recognized character. |
| CharGetTotalOrgX | Returns the abscissa of the upper left corner of a recognized character. |
| CharGetTotalOrgY | Returns the ordinate of the upper left corner of a recognized character. |
| CharGetWidth | Returns the width of a recognized character. |
| DrawChar | Draws the bounding box of a given character. |
| DrawChars | Draws the bounding boxes of all characters in the image. |
| DrawCharsWithCurrentPen | Draws the bounding boxes of all characters in the image. |
| DrawCharWithCurrentPen | Draws the bounding box of a given character. |
| EmptyChars | Empties the list of known characters. |
| EOCR | Constructs an OCR context. |

| | |
|---|---|
| FindAllChars | Locates the characters by filtering the objects according to their size, and grouping them if the segmentation mode is set to ESegmentationMode_RepasteObjects. |
| GetCharSpacing | Spacing that separates characters. |
| GetCompareAspectRatio | Flag indicating whether the character aspect ratio is used to compute the recognition score. |
| GetConfidenceRatio | Returns the degree of confidence in the recognized character. |
| GetCutLargeChars | Flag indicating whether the large chars cutting mode is used. |
| GetFirstCharCode | Returns the code of the pattern that matches at best a recognized character. |
| GetFirstCharDistance | Computes the degree of similarity between the best matching pattern and a recognized character. |
| GetMatchingMode | Matching mode to use to compare characters to the template. |
| GetMaxCharHeight | Maximum character height. |
| GetMaxCharWidth | Maximum character width. |
| GetMinCharHeight | Minimum character height. |

| | |
|---|---|
| GetMinCharWidth | Minimum character width. |
| GetNoiseArea | Noise area. |
| GetNumChars | Number of recognized characters. |
| GetNumPatterns | Number of patterns in the current font. |
| GetPatternBitmap | Returns a pointer to an image holding the pattern of the given index. This image should not be modified. |
| GetPatternClass | Returns the class of a given pattern in the current font. |
| GetPatternCode | Returns the character code of a given pattern in the current font. |
| GetPatternHeight | Current pattern height, as set at the last EOCR::NewFont or EOCR::Load operation. |
| GetPatternWidth | Current pattern width, as set at the last EOCR::NewFont or EOCR::Load operation. |
| GetRelativeSpacing | Relative spacing value. |
| GetRelativeThreshold | Relative threshold used for image segmentation. |

| | |
|---|---|
| GetRemoveBorder | Flag indicating whether all blobs touching the ROI edges are automatically discarded. |
| GetRemoveNarrowOrFlat | Flag indicating whether the characters are discarded when either dimension falls below the minimum value |
| GetSecondCharCode | Returns the code of the pattern that matches at second best a recognized character. |
| GetSecondCharDistance | Computes the degree of similarity between the second best matching pattern and a recognized character. |
| GetSegmentationMode | Segmentation mode. |
| GetShiftingMode | Shifting mode to use to compare characters to the template. |
| GetShiftXTolerance | Horizontal translation tolerance around the nominal position when the character positions are explicitly specified. |
| GetShiftYTolerance | Vertical translation tolerance around the nominal position when the character positions are explicitly specified. |
| GetTextColor | Text color. |
| GetThreshold | Threshold mode used for image segmentation. |

| | |
|---|---|
| GetTrueThreshold | Absolute threshold level when using a single threshold. |
| HitChar | Returns **TRUE** if the cursor is placed over the character specified by **charIndex**. |
| HitChars | Returns **TRUE** if the cursor is placed over a character and sets the **charIndex** accordingly. |
| LearnPattern | Adds a new pattern to the font. |
| LearnPatterns | Adds all the patterns from the source image to the font. |
| Load | Loads the EOCR object configuration. The given ESerializer must have been created for reading. |
| MatchChar | Reads a single character, i.e. finds the best match between the patterns in the font and the given character. |
| NewFont | Empties the contents of the font and sets the size of the pattern array to be used from then on. |
| operator= | Copies all the data from another EOCR object into the current EOCR object |

| | |
|---|---|
| ReadText | Reads one or more rows of characters, i.e. finds the best match between the patterns in the font and the segmented characters. |
| ReadTextWide | Reads one or more rows of characters, i.e. finds the best match between the patterns in the font and the segmented characters. |
| Recognize | Achieves all processing phases (blob analysis, character segmentation and pattern recognition) in a single operation. |
| RecognizeWide | Achieves all processing phases (blob analysis, character segmentation and pattern recognition) in a single operation. |
| RemovePattern | Removes a given pattern from the current font. |
| Save | Saves the EOCR object configuration. The given ESerializer must have been created for writing. |
| SetCharSpacing | Spacing that separates characters. |
| SetCompareAspectRatio | Flag indicating whether the character aspect ratio is used to compute the recognition score. |
| SetCutLargeChars | Flag indicating whether the large chars cutting mode is used. |
| SetMatchingMode | Matching mode to use to compare characters to the template. |

| | |
|---|---|
| SetMaxCharHeight | Maximum character height. |
| SetMaxCharWidth | Maximum character width. |
| SetMinCharHeight | Minimum character height. |
| SetMinCharWidth | Minimum character width. |
| SetNoiseArea | Noise area. |
| SetPatternClass | Sets the class of a given pattern in the current font. |
| SetPatternCode | Sets the character code of a given pattern in the current font. |
| SetRelativeSpacing | Relative spacing value. |
| SetRelativeThreshold | Relative threshold used for image segmentation. |
| SetRemoveBorder | Flag indicating whether all blobs touching the ROI edges are automatically discarded. |
| SetRemoveNarrowOrFlat | Flag indicating whether the characters are discarded when either dimension falls below the minimum value |

| | |
|---|---|
| SetSegmentationMode | Segmentation mode. |
| SetShiftingMode | Shifting mode to use to compare characters to the template. |
| SetShiftXTolerance | Horizontal translation tolerance around the nominal position when the character positions are explicitly specified. |
| SetShiftYTolerance | Vertical translation tolerance around the nominal position when the character positions are explicitly specified. |
| SetTextColor | Text color. |
| SetThreshold | Threshold mode used for image segmentation. |

E

O

# CR::AddChar

Add a character coordinates to the list.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void AddChar(
 OEV_INT32 originX,
 OEV_INT32 originY,
 OEV_INT32 endX,
 OEV_INT32 endY
 )
```

### Parameters

*originX*

Abscissa of the upper left corner of the bounding box of the character.

*originY*

Ordinate of the upper left corner of the bounding box of the character.

*endX*

Abscissa of the bottom right corner of the bounding box of the character.

*endY*

Ordinate of the bottom right corner of the bounding box of the character.

### Remarks

It is to use when you know the exact position of the characters to be recognized, to bypass the segmentation step, for efficiency or reliability purposes. To use this feature, simply specify the character positions by successive calls to **AddChar**. When this is done, the remainder of the OCR processing steps can take place. To empty the list of known characters, call EOCR::EmptyChars.

# EOCR::AddPatternFromImage

Adds a new pattern to the font.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void AddPatternFromImage(
 EROIBW8* sourceImage,
 OEV_INT32 originX,
 OEV_INT32 originY,
 OEV_INT32 width,
 OEV_INT32 height,
 OEV_INT32 code,
 OEV_UINT32 classes
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*originX*

Abscissa of the upper left corner of the bounding box of the pattern.

*originY*

Ordinate of the upper left corner of the bounding box of the pattern.

*width*

Width of the bounding box of the pattern.

*height*

Height of the bounding box of the pattern.

*code*

Character code of the pattern.

*classes*

Class of the pattern, as defined by EOCRClass.

### Remarks

The pattern is extracted from an image by specifying a bounding rectangle.

# EOCR::BuildObjects

Segments the source image, i.e. detects and labels the objects.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void BuildObjects(
 EROIBW8* sourceImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

### Remarks

An object is a connected set of pixels above or below **Threshold** (according to **TextColor**).

# EOCR::CharGetDstX

Returns the abscissa of the lower right corner of a recognized character.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 CharGetDstX(
 OEV_INT32 index
 )
```

### Parameters

*index*

Character number (in range **0..NumChars-1**).

# EOCR::CharGetDstY

Returns the ordinate of the lower right corner of a recognized character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 CharGetDstY(
 OEV_INT32 index
 )
```

### Parameters

*index*

Character number (in range **0..NumChars-1**).

# EOCR::CharGetHeight

Returns the height of a recognized character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 CharGetHeight(
 OEV_INT32 index
 )
```

### Parameters

*index*

Character number (in range **0..NumChars-1**).

# EOCR::CharGetOrgX

Returns the abscissa of the upper left corner of a recognized character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 CharGetOrgX(
 OEV_INT32 index
 )
```

### Parameters

*index*

Character number (in range **0..NumChars-1**).

# EOCR::CharGetOrgY

Returns the ordinate of the upper left corner of a recognized character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 CharGetOrgY(
 OEV_INT32 index
 )
```

### Parameters

*index*

Character number (in range **0..NumChars-1**).

# EOCR::CharGetTotalDstX

Returns the abscissa of the lower right corner of a recognized character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 CharGetTotalDstX(
 OEV_INT32 index
 )
```

### Parameters

*index*

Character number (in range **0..NumChars-1**).

### Remarks

The coordinates are computed with respect to the parent image rather than the ROI.

# EOCR::CharGetTotalDstY

Returns the ordinate of the lower right corner of a recognized character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 CharGetTotalDstY(
 OEV_INT32 index
 )
```

### Parameters

*index*

Character number (in range **0..NumChars-1**).

### Remarks

The coordinates are computed with respect to the parent image rather than the ROI.

# EOCR::CharGetTotalOrgX

Returns the abscissa of the upper left corner of a recognized character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 CharGetTotalOrgX(
 OEV_INT32 index
 )
```

### Parameters

*index*

Character number (in range **0..NumChars-1**).

### Remarks

The coordinates are computed with respect to the parent image rather than the ROI.

# EOCR::CharGetTotalOrgY

Returns the ordinate of the upper left corner of a recognized character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 CharGetTotalOrgY(
 OEV_INT32 index
 )
```

### Parameters

*index*

Character number (in range **0..NumChars-1**).

### Remarks

The coordinates are computed with respect to the parent image rather than the ROI.

# EOCR::CharGetWidth

Returns the width of a recognized character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 CharGetWidth(
 OEV_INT32 index
 )
```

### Parameters

*index*

Character number (in range **0..NumChars-1**).

# EOCR::GetCharSpacing

# EOCR::SetCharSpacing

Spacing that separates characters.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetCharSpacing()
void SetCharSpacing(OEV_INT32 n32CharSpacing)
```

### Remarks

When two objects are separated by a vertical gap larger or equal than the spacing, they are considered to belong to distinct characters. The segmentation parameters *must* be the same for both learning and recognition process.

# EOCR::GetCompareAspectRatio

# EOCR::SetCompareAspectRatio

Flag indicating whether the character aspect ratio is used to compute the recognition score.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetCompareAspectRatio()
void SetCompareAspectRatio(BOOL bCompareAspectRatio)
```

### Remarks

When **TRUE**, the character aspect ratio is used to compute the recognition score.

# EOCR::GetCutLargeChars

# EOCR::SetCutLargeChars

Flag indicating whether the large chars cutting mode is used.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
BOOL GetCutLargeChars()

void SetCutLargeChars(BOOL bCutLargeChars)
```

### Remarks

If **TRUE**, candidate characters larger than **MaxWidth** are split into as many parts as required. If **FALSE**, large characters are discarded. The segmentation parameters *must* be the same for both learning and recognition process.

# EOCR::DrawChar

Draws the bounding box of a given character.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawChar(
 HDC graphicContext,
 OEV_UINT32 charIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawChar(
 HDC graphicContext,
 const ERGBColor& color,
 OEV_UINT32 charIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void DrawChar(
 EDrawAdapter* graphicContext,
 OEV_UINT32 charIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*charIndex*

Character index, in range **0..NumChars-1**.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*color*

The color in which to draw the overlay.

### Remarks

Drawing is done in the device context associated to the desired window. The current pen is used.

# EOCR::DrawChars

Draws the bounding boxes of all characters in the image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawChars(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawChars(
 HDC graphicContext,
 const ERGBColor& color,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void DrawChars(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*color*

The color in which to draw the overlay.

## Remarks

Drawing is done in the device context associated to the desired window. The current pen is used for all characters (to vary the colors, draw the objects separately using the EOCR::DrawChar method instead).

# EOCR::DrawCharsWithCurrentPen

Draws the bounding boxes of all characters in the image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawCharsWithCurrentPen(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

## Remarks

Drawing is done in the device context associated to the desired window. The current pen is used for all characters (to vary the colors, draw the objects separately using the EOCR::DrawChar method instead).

# EOCR::DrawCharWithCurrentPen

Draws the bounding box of a given character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawCharWithCurrentPen(
 HDC graphicContext,
 OEV_UINT32 charIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*charIndex*

Character index, in range **0..NumChars-1**.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

## Remarks

Drawing is done in the device context associated to the desired window. The current pen is used.

# EOCR::EmptyChars

Empties the list of known characters.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EmptyChars(
 )
```

## Remarks

It is to use when you know the exact position of the characters to be recognized. See member EOCR::AddChar.

# EOCR::EOCR

Constructs an OCR context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EOCR(
 )

void EOCR(
 const EOCR& other
 )
```

## Parameters

*other*

Another EOCR object to be copied in the new EOCR object.

## Remarks

By default, the **Threshold** property is set to **128**.

# EOCR::FindAllChars

Locates the characters by filtering the objects according to their size, and grouping them if the segmentation mode is set to RepasteObjects.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void FindAllChars(
 EROIBW8* sourceImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI. This parameter is not used and kept only for compatibility purposes.

### Remarks

The characters are sorted from left to right. This operation must be performed after a call to EOCR::BuildObjects and before a call to EOCR::ReadText.

# EOCR::GetConfidenceRatio

Returns the degree of confidence in the recognized character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetConfidenceRatio(
 OEV_INT32 index
 )
```

### Parameters

*index*

Character number (in range **0..NumChars-1**).

### Remarks

A value of 100 % means there is no difference between the recognized character and the best matching pattern. A value of 0 % means there is no way to distinguish between the best and second best matching pattern. The computation of the confidence ratio is based on the first and second characters distance parameters (see EOCR::GetFirstCharDistance and EOCR::GetSecondCharDistance).

# EOCR::GetFirstCharCode

Returns the code of the pattern that matches at best a recognized character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetFirstCharCode(
 OEV_INT32 index
 )
```

### Parameters

*index*

Character number (in range **0..NumChars-1**).

# EOCR::GetFirstCharDistance

Computes the degree of similarity between the best matching pattern and a recognized character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetFirstCharDistance(
 OEV_INT32 index
 )
```

### Parameters

*index*

Character number (in range **0..NumChars-1**).

### Remarks

Returns **0** for a perfect match and **1** for a total mismatch.

# EOCR::GetPatternBitmap

Returns a pointer to an image holding the pattern of the given index. This image should not be modified.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageBW8* GetPatternBitmap(
 OEV_INT32 index
 )
```

### Parameters

*index*

Pattern number (in range **0.. GetNumPatterns() -1**).

# EOCR::GetPatternClass

Returns the class of a given pattern in the current font.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetPatternClass(
 OEV_INT32 index
 )
```

### Parameters

*index*

Pattern number (in range **0..NumPatterns-1**).

# EOCR::GetPatternCode

Returns the character code of a given pattern in the current font.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetPatternCode(
 OEV_INT32 index
 )
```

### Parameters

*index*

Pattern number (in range **0..NumPatterns-1**).

# EOCR::GetSecondCharCode

Returns the code of the pattern that matches at second best a recognized character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetSecondCharCode(
 OEV_INT32 index
 )
```

### Parameters

*index*

Character number (in range **0..NumChars-1**).

# EOCR::GetSecondCharDistance

Computes the degree of similarity between the second best matching pattern and a recognized character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSecondCharDistance(
 OEV_INT32 index
 )
```

### Parameters

*index*

Character number (in range **0..NumChars-1**).

### Remarks

Returns **0** for a perfect match and **1** for a total mismatch.

# EOCR::HitChar

Returns **TRUE** if the cursor is placed over the character specified by **charIndex**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL HitChar(
 OEV_INT32 cursorX,
 OEV_INT32 cursorY,
 OEV_UINT32 charIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*cursorX*

Current cursor coordinates.

*cursorY*

Current cursor coordinates.

*charIndex*

Index of the character to be hit.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0**, the default value, the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

### Remarks

If zooming and/or panning were used when drawing the ROI, the same values must be used with EOCR::HitChar.

# EOCR::HitChars

Returns **TRUE** if the cursor is placed over a character and sets the **charIndex** accordingly.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL HitChars(
 OEV_INT32 cursorX,
 OEV_INT32 cursorY,
 OEV_UINT32& charIndex,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*cursorX*

Current cursor coordinates.

*cursorY*

Current cursor coordinates.

*charIndex*

Index of the character hit.

*zoomX*

Horizontal zooming factor. By default, **TRUE** scale is used.

*zoomY*

Vertical zooming factor. If set to **0**, the default value, the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

### Remarks

If zooming and/or panning were used when drawing the ROI, the same values must be used with EOCR::HitChars.

# EOCR::LearnPattern

Adds a new pattern to the font.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void LearnPattern(
 EROIBW8* sourceImage,
 OEV_UINT32 charIndex,
 OEV_INT32 code,
 OEV_UINT32 classes,
 BOOL autoSegmentation
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*charIndex*

Index of the character (ASCII or Unicode) to learn.

*code*

Character code of the pattern.

*classes*

Class of the pattern, as defined by EOCRClass.

*autoSegmentation*

Boolean indicating whether the calculation of the true threshold has to be forced (default **TRUE**) or bypassed (**FALSE**).

### Remarks

The pattern is selected by its index value, as assigned by the EOCR::FindAllChars process.

# EOCR::LearnPatterns

Adds all the patterns from the source image to the font.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void LearnPatterns(
 EROIBW8* sourceImage,
 const std::string& text,
 OEV_UINT32 singleClass,
 BOOL autoSegmentation
 )
void LearnPatterns(
 EROIBW8* sourceImage,
 const std::string& text,
 const std::vector<OEV_UINT32>& classes,
 BOOL autoSegmentation
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*text*

String containing character codes of the patterns.

*singleClass*

If specified, all the characters of the string are associated with the same class(es), that is specified by **singleClass**.

*autoSegmentation*

Boolean indicating whether the calculation of the true threshold has to be forced (default **TRUE**) or bypassed (**FALSE**).

*classes*

If specified, the i-th character of the string is associated with the class specified by the i-th element of the vector **classes**.

### Remarks

Patterns are ordered by their index value, as assigned by the EOCR::FindAllChars process.

# EOCR::Load

Loads the EOCR object configuration. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# EOCR::MatchChar

Reads a single character, i.e. finds the best match between the patterns in the font and the given character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void MatchChar(
 EROIBW8* sourceImage,
 OEV_UINT32 classes,
 OEV_INT32 index,
 OEV_INT32 shiftX,
 OEV_INT32 shiftY
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*classes*

Logical mask obtained by combining the values of EOCRClass, to specify to what classes the character may belong.

*index*

Character number (in range **0..NumChars-1**).

*shiftX*

Horizontal translation applied to the character.

*shiftY*

Vertical translation applied to the character.

### Remarks

A shift can be apply to the character. This operation can only be performed after a call to EOCR::FindAllChars.

# EOCR::GetMatchingMode

# EOCR::SetMatchingMode

Matching mode to use to compare characters to the template.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EMatchingMode GetMatchingMode()
void SetMatchingMode(Euresys::Open_eVision_2_10::EMatchingMode eMatchingMode)
```

### Remarks

By default, the root-mean-square error method is used. These modes are meant to be used without thresholding, that is when one of the EOCRColor_DarkOnLight and EOCRColor_LightOnDark colors are used.

# EOCR::GetMaxCharHeight

# EOCR::SetMaxCharHeight

Maximum character height.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetMaxCharHeight()
void SetMaxCharHeight(OEV_INT32 n32MaxCharHeight)
```

### Remarks

A character larger than the maximum width or higher than the maximum height is discarded. The segmentation parameters *must* be the same for both learning and recognition process.

# EOCR::GetMaxCharWidth

# EOCR::SetMaxCharWidth

Maximum character width.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetMaxCharWidth()
void SetMaxCharWidth(OEV_INT32 n32MaxCharWidth)
```

### Remarks

A character larger than the maximum width or higher than the maximum height is discarded. The segmentation parameters *must* be the same for both learning and recognition process.

# EOCR::GetMinCharHeight

# EOCR::SetMinCharHeight

Minimum character height.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetMinCharHeight()
void SetMinCharHeight(OEV_INT32 n32MinCharHeight)
```

### Remarks

A character both narrower than the minimum width and lower than the minimum height is discarded. The segmentation parameters *must* be the same for both learning and recognition process.

# EOCR::GetMinCharWidth

# EOCR::SetMinCharWidth

Minimum character width.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetMinCharWidth()
void SetMinCharWidth(OEV_INT32 n32MinCharWidth)
```

### Remarks

A character both narrower than the minimum width and lower than the minimum height is discarded. The segmentation parameters *must* be the same for both learning and recognition process.

# EOCR::NewFont

Empties the contents of the font and sets the size of the pattern array to be used from then on.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void NewFont(
 OEV_UINT32 patternWidth,
 OEV_UINT32 patternHeight
 )
```

### Parameters

*patternWidth*

Width of the normalized pattern representation, in pixels.

*patternHeight*

Height of the normalized pattern representation, in pixels.

### Remarks

A larger pattern array improves the recognition sensitivity, at the expense of increased processing time.

# EOCR::GetNoiseArea

# EOCR::SetNoiseArea

Noise area.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetNoiseArea()
void SetNoiseArea(OEV_INT32 n32NoiseArea)
```

### Remarks

When a blob has an area smaller than this value, it is ignored. The segmentation parameters *must* be the same for both learning and recognition process.

# EOCR::GetNumChars

Number of recognized characters.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetNumChars()
```

# EOCR::GetNumPatterns

Number of patterns in the current font.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetNumPatterns()
```

# EOCR::operator=

Copies all the data from another EOCR object into the current EOCR object

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EOCR& operator=(
 const EOCR& other
 )
```

### Parameters

*other*

EOCR object to be copied

# EOCR::GetPatternHeight

Current pattern height, as set at the last EOCR::NewFont or EOCR::Load operation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetPatternHeight()
```

# EOCR::GetPatternWidth

Current pattern width, as set at the last EOCR::NewFont or EOCR::Load operation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetPatternWidth()
```

# EOCR::ReadText

Reads one or more rows of characters, i.e. finds the best match between the patterns in the font and the segmented characters.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::string ReadText(
 EROIBW8* sourceImage,
 OEV_INT32 maximumNumberOfCharacters,
 OEV_UINT32 classes,
 BOOL autoSegmentation
 )

std::string ReadText(
 EROIBW8* sourceImage,
 OEV_INT32 maximumNumberOfCharacters,
 const std::vector<OEV_UINT32>& classes,
 BOOL autoSegmentation
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*maximumNumberOfCharacters*

Maximum number of characters to be read.

*classes*

Pointer to an array of logical masks obtained by combining the values of EOCRClass, to specify to what classes the character may belong. Each mask value in the array applies to the corresponding character.

*autoSegmentation*

Boolean indicating whether the calculation of the true threshold has to be forced (default **TRUE**) or bypassed (**FALSE**).

## Remarks

This operation can only be performed after a call to EOCR::FindAllChars. In case the text contains character with a code > 255, EOCR::ReadText will throw an exception. In this case, you should use EOCR::ReadTextWide.

# EOCR::ReadTextWide

Reads one or more rows of characters, i.e. finds the best match between the patterns in the font and the segmented characters.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::wstring ReadTextWide(
 EROIBW8* sourceImage,
 OEV_INT32 maximumNumberOfCharacters,
 OEV_UINT32 classes,
 BOOL autoSegmentation
 )

std::wstring ReadTextWide(
 EROIBW8* sourceImage,
 OEV_INT32 maximumNumberOfCharacters,
 const std::vector<OEV_UINT32>& classes,
 BOOL autoSegmentation
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*maximumNumberOfCharacters*

Maximum number of characters to be read.

*classes*

Pointer to an array of logical masks obtained by combining the values of EOCRClass, to specify to what classes the character may belong. Each mask value in the array applies to the corresponding character.

*autoSegmentation*

Boolean indicating whether the calculation of the true threshold has to be forced (default **TRUE**) or bypassed (**FALSE**).

### Remarks

This operation can only be performed after a call to EOCR::FindAllChars. In case the text contains character with a code > 65535, ReadTextWide will throw an exception. In this case, you should read the text character by character by using EOCR::GetFirstCharCode.

# EOCR::Recognize

Achieves all processing phases (blob analysis, character segmentation and pattern recognition) in a single operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::string Recognize(
 EROIBW8* sourceImage,
 OEV_INT32 maximumNumberOfCharacters,
 OEV_UINT32 classes
 )

std::string Recognize(
 EROIBW8* sourceImage,
 OEV_INT32 maximumNumberOfCharacters,
 const std::vector<OEV_UINT32>& classes
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*maximumNumberOfCharacters*

Maximum number of characters to be read.

*classes*

Pointer to an array of logical mask obtained by combining the values of EOCRClass, to specify to what classes the character may belong. Each mask value in the array applies to the corresponding character.

### Remarks

This method does the same as a sequence of EOCR::BuildObjects/EOCR::FindAllChars/EOCR::ReadText,

# EOCR::RecognizeWide

Achieves all processing phases (blob analysis, character segmentation and pattern recognition) in a single operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::wstring RecognizeWide(
 EROIBW8* sourceImage,
 OEV_INT32 maximumNumberOfCharacters,
 OEV_UINT32 classes
 )

std::wstring RecognizeWide(
 EROIBW8* sourceImage,
 OEV_INT32 maximumNumberOfCharacters,
 const std::vector<OEV_UINT32>& classes
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*maximumNumberOfCharacters*

Maximum number of characters to be read.

*classes*

Pointer to an array of logical mask obtained by combining the values of EOCRClass, to specify to what classes the character may belong. Each mask value in the array applies to the corresponding character.

## Remarks

This method does the same as a sequence of EOCR::BuildObjects/EOCR::FindAllChars/EOCR::ReadText,

# EOCR::GetRelativeSpacing

# EOCR::SetRelativeSpacing

Relative spacing value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetRelativeSpacing()
void SetRelativeSpacing(float f32RelativeSpacing)
```

### Remarks

This value is the ratio of the width of the spaces between characters and the character width. For example, characters 25 pixels wide separated by 10 pixels gaps correspond to a relative spacing of **10/25 = 0.4**. The default value of this parameter is **0**, corresponding to no spacing. This parameter is relevant only when the **CutLargeChars** mode is enabled. The segmentation parameters *must* be the same for both learning and recognition process.

# EOCR::GetRelativeThreshold

# EOCR::SetRelativeThreshold

Relative threshold used for image segmentation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetRelativeThreshold()
```

```
void SetRelativeThreshold(float f32Threshold)
```

### Remarks

The **RelativeThreshold** is the fraction of the image pixels that will be set below the threshold. Only used when the EOCR::Threshold property is set to EThresholdMode_Relative. The default value is 0.5.

# EOCR::GetRemoveBorder

# EOCR::SetRemoveBorder

Flag indicating whether all blobs touching the ROI edges are automatically discarded.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetRemoveBorder()
void SetRemoveBorder(BOOL bRemoveBorder)
```

### Remarks

If **TRUE**, all blobs touching the ROI edges are automatically discarded. By default, this feature is turned on. The segmentation parameters *must* be the same for both learning and recognition process.

# EOCR::GetRemoveNarrowOrFlat

# EOCR::SetRemoveNarrowOrFlat

Flag indicating whether the characters are discarded when either dimension falls below the minimum value

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetRemoveNarrowOrFlat()

void SetRemoveNarrowOrFlat(BOOL bRemoveNarrowOrFlat)
```

### Remarks

If **TRUE**, characters are discarded if either their width or their height is smaller than the minimum value. By default, this feature is disabled (only smaller characters in *both* height and width are discarded â€"the condition could be written **Narrow**<bc>And</bc>**Flat**). The segmentation parameters *must* be the same for both learning and recognition process.

# EOCR::RemovePattern

Removes a given pattern from the current font.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void RemovePattern(
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index of the pattern to be removed from the current font.

# EOCR::Save

Saves the EOCR object configuration. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# EOCR::GetSegmentationMode

# EOCR::SetSegmentationMode

Segmentation mode.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::ESegmentationMode GetSegmentationMode()

void SetSegmentationMode(Euresys::Open_eVision_2_10::ESegmentationMode
eSegmentationMode)
```

### Remarks

The segmentation parameters *must* be the same for both learning and recognition process.

# EOCR::SetPatternClass

Sets the class of a given pattern in the current font.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetPatternClass(
 OEV_INT32 index,
 OEV_UINT32 classIdx
 )
```

### Parameters

*index*

Pattern number (in range **0..NumPatterns-1**).

*classIdx*

The class.

# EOCR::SetPatternCode

Sets the character code of a given pattern in the current font.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetPatternCode(
 OEV_INT32 index,
 OEV_INT32 code
 )
```

### Parameters

*index*

Pattern number (in range **0..NumPatterns-1**).

*code*

The character code.

# EOCR::GetShiftingMode

# EOCR::SetShiftingMode

Shifting mode to use to compare characters to the template.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EShiftingMode GetShiftingMode()
void SetShiftingMode(Euresys::Open_eVision_2_10::EShiftingMode eShiftingMode)
```

# EOCR::GetShiftXTolerance

# EOCR::SetShiftXTolerance

Horizontal translation tolerance around the nominal position when the character positions are explicitly specified.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetShiftXTolerance()
void SetShiftXTolerance(OEV_UINT32 un32ShiftXTolerance)
```

## Remarks

By default, no shifting is allowed (**ShiftX = 0**). The tolerance can be used in two ways: either each character is moved individually until the best match is found, or the set of characters is matched as a whole, until the best average error is reached. See EOCR::ShiftingMode.

# EOCR::GetShiftYTolerance

# EOCR::SetShiftYTolerance

Vertical translation tolerance around the nominal position when the character positions are explicitly specified.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetShiftYTolerance()
void SetShiftYTolerance(OEV_UINT32 un32ShiftYTolerance)
```

## Remarks

By default, no shifting is allowed (**ShiftY = 0**). The tolerance can be used in two ways: either each character is moved individually until the best match is found, or the set of characters is matched as a whole, until the best average error is reached. See EOCR::ShiftingMode.

# EOCR::GetTextColor

# EOCR::SetTextColor

Text color.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EOCRColor GetTextColor()
void SetTextColor(Euresys::Open_eVision_2_10::EOCRColor eTextColor)
```

### Remarks

The segmentation parameters *must* be the same for both learning and recognition process.

# EOCR::GetThreshold

# EOCR::SetThreshold

Threshold mode used for image segmentation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetThreshold()
void SetThreshold(OEV_INT32 un32Threshold)
```

### Remarks

Threshold value as defined by the EThresholdMode enum. By default, the "minimum residue" mode is used to determine the threshold value. In case of an absolute threshold, the threshold value is given instead.

# EOCR::GetTrueThreshold

Absolute threshold level when using a single threshold.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetTrueThreshold()
```

### Remarks

This value is valid only when the EOCR::BuildObjects or EOCR::ReadText method has been called beforehand.

# 4.125. EOCR2 Class

Manages a complete context for the font-dependent printed character reader implemented in EasyOCR2.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddCharactersToDatabase | Reads reference characters from disk and adds them to the database used for text recognition. The characters can be read from a trueType (.ttf/.ttc) file or from an EasyOCR2 database file. |
| ClearCharacterDatabase | Clears the reference character database from this EOCR2 instance. |
| ClearResult | Clear the current detected text if it exists. |
| Detect | Finds the text in an image as follows:<br>(1) Detects potential characters in the image following the given text polarity.<br>(2) Fits bounding boxes to the detected characters, following the given topology and character width/height.<br>(3) Extracts the detected characters from the image.<br>The detected characters are output as an EOCR2Text structure. |
| DrawDetection | Draws the bounding boxes found by the topology detection algorithm. |
| DrawDetectionWithCurrentPen | Draws the bounding boxes found by the topology detection algorithm with the current pen. |

| | |
|---|---|
| DrawRecognition | Draws the recognized text next to the character bounding box in the image. |
| DrawRecognitionWithCurrentPen | Draws the recognized text next to the character bounding box in the image with the current pen. |
| DrawSegmentation | Draws the blobs found by the segmentation algorithm. |
| DrawSegmentationWithCurrentPen | Draws the blobs found by the segmentation algorithm with the current pen. |
| EOCR2 | Constructs an EOCR2 context. |
| GetCharacterDatabase | Sets the EOCR2CharacterDatabase used for recognizing text. |
| GetCharsHeight | Sets the expected character height in pixels. |
| GetCharsMaxFragmentation | Sets the **CharsMaxFragmentation** parameter for the segmentation algorithm. This will determine the minimum size a blob should be in order to be considered a potential character. A high setting will allow only larger blobs, a low setting will also allow smaller blobs.<br>The minimum blob size to be considered a potential character is defined as:<br>`CharsMaxFragmentation * CharsHeight * CharsWidth` |

GetCharsSpacingBias

Sets the **CharSpacingBias** parameter for the topology fitting algorithm, which optimises the spacing of the bounding boxes to optimally fit the detected blobs. This will determine whether the method is biased toward finding narrow spacing, wide spacing or is neutral.

GetCharsWidth

Sets the expected character width in pixels.

GetCharsWidthBias

Sets the **CharsWidthBias** parameter for the topology fitting algorithm, which optimises the width of the bounding boxes to optimally fit the detected blobs. This will determine whether the method is biased toward finding narrow boxes, wide boxes or is neutral.

GetCharsWidthRange

Sets the range of expected character widths in pixels.

GetCharsWidthTolerance

Sets the **CharsWidthTolerance** parameter for the topology fitting algorithm, which will attempt to optimise the width of the bounding boxes to optimally fit the detected blobs. This will determine the range of bounding box sizes that will be accepted, defined as:

```
 (1 - CharsWidthTolerance) * CharsWidth <=
BBoxWidth <= (1 + CharsWidthTolerance) * CharsWidth
```

GetDetectionDelta

Sets the **DetectionDelta** parameter for the segmentation algorithm. This will determine the range of grayscale-values used to determine the stability of a blob. A low setting will make the algorithm more sensitive to noise, a high setting will make the algorithm insensitive to blobs with low contrast to the background.

| | |
|---|---|
| GetDetectionMethod | Sets the **EOCR2DetectionMethod** parameter for the topology fitting algorithm, which will place textboxes on the segmentation results, matching the given topology. |
| GetMaxVariation | Sets the **maxVariation** parameter for the segmentation algorithm. This parameter determines how stable a blob in the image should be in order to be considered a potential character, a region with clearly defined edges is generally considered stable while a blurry region is not. A high setting allows more unstable blobs, a low setting allows only very stable blobs. |
| GetNumDetectionPasses | Sets the **NumDetectionPasses** parameter for the topology fitting algorithm, which will place textboxes on the segmentation results (blobs), matching the given topology. The first pass will consider all blobs, subsequent passes will only consider those blobs that are inside the textboxes from the previous pass, sometimes resulting in a more optimal fit. |
| GetReadText | Outputs an EOCR2Text structure containing the detailed detection and recognition results. |
| GetRelativeSpacesWidthRange | Sets the range of expected spaces between words as a fraction of the character width. |
| GetSegmentationMethod | Sets the **EOCR2SegmentationMethod** parameter for the segmentation algorithm, which will detect blobs in the image. |

| | |
|---|---|
| GetTextAngleRange | Sets the **TextAngleRange** parameter for the topology fitting algorithm, which will attempt to find the angle of the text in the image with respect to the horizontal. This will determine the center of the range of angles that will be tested, defined as:<br>`TextAngleRange.min() <= angle <= TextAngleRange.max()` |
| GetTextAngleTolerance | Sets the **TextAngleTolerance** parameter for the topology fitting algorithm, which finds the angle of the text in the image with respect to the horizontal. This will limit the range of angles that will be tested, defined as:<br>`BaseAngle - AngleTolerance <= angle <= BaseAngle + AngleTolerance` |
| GetTextBaseAngle | Sets the **TextBaseAngle** parameter for the topology fitting algorithm, which will attempt to find the angle of the text in the image with respect to the horizontal. This will determine the center of the range of angles that will be tested, defined as:<br>`BaseAngle - AngleTolerance <= angle <= BaseAngle + AngleTolerance` |
| GetTextPolarity | Sets the **TextPolarity** parameter for the segmentation algorithm. This will determine whether the algorithm searches for light blobs in a dark background or for dark blobs in a light background. |
| GetTimeOut | Time-out for the EOCR2::Read, EOCR2::Detect and EOCR2::Recognize methods. |

| | |
|---|---|
| GetTopology | Sets the topology of the text that should be found in the image. A modified version of Regex expressions are used, where:<br>**.**(dot) represents any character (not including a space).<br>**L** represents a letter.<br>**Lu** represents an uppercase letter.<br>**Ll** represents a lowercase letter.<br>**N** represents a digit.<br>**P** represents a punctuation character !"#&'()*,-./:;<>?[\]_{\|}~<br>**S** represents the symbols $+-<=>\|~<br>**\n** represents a line break.<br>' ' (space) represents a space between two words.<br>Combinations can be made, for example: `[LN]` represents an alpha-numeric character.<br>To specify multiple characters, simply add `{n}` at the end for n characters. If the amount of characters is uncertain, specify `{n,m}` for a minimum of n characters and a maximum of m characters.<br>The topology "`[LuN]{3,5}PN{4} \n .{5} LL`" represents a text comprised of 2 lines:<br>The first line has 1 word composed of 3 to 5 uppercase alpha-numeric characters, followed by a punctuation character and 4 numbers.<br>The second line has 2 words. The first word comprises of 5 wild-card characters, the second word has 2 alphabetic characters (upper- or lowercase). |
| HitTestChar | Tests the cursor position for the presence of a character. If one is present under the cursor, it returns true and fills the EOCR2Char object passed as parameter. |
| HitTestLine | Tests the cursor position for the presence of a line. If one is present under the cursor, it returns true and fills the EOCR2Line object passed as parameter. |
| HitTestText | Tests the cursor position for the presence of a text. If one is present under the cursor, it returns true and fills the EOCR2Text object passed as parameter. |

| | |
|---|---|
| HitTestWord | Tests the cursor position for the presence of a word. If one is present under the cursor, it returns true and fills the EOCR2Word object passed as parameter. |
| Learn | Learns reference characters from a given EOCR2Text/EOCR2Line/EOCR2Word/EOCR2Char instance, containing user-specified character codes. |
| Load | Loads a model, containing parameter settings used for all operations in EOCR2, from disk. |
| operator= | Assignment operator, copies another EOCR2 instance to this one. |
| Read | Performs all steps required for reading text from an image:<br>(1) Detects potential characters in the image following the given text polarity and character width/height.<br>(2) Fits bounding boxes to the detected characters, following the given topology and character width/height.<br>(3) Recognizes the detected characters using the given reference character database.<br>The read text is output as a string. |
| Recognize | Recognizes the characters in a given EOCR2Text instance, based on a given reference font. |
| Save | Saves the model to disk, containing the current parameter setting used for all operations in EOCR2. |
| SaveCharacterDatabase | Saves the current reference character database to disk. |

| | |
|---|---|
| SetCharacterDatabase | Sets the EOCR2CharacterDatabase used for recognizing text. |
| SetCharsHeight | Sets the expected character height in pixels. |
| SetCharsMaxFragmentation | Sets the **CharsMaxFragmentation** parameter for the segmentation algorithm. This will determine the minimum size a blob should be in order to be considered a potential character. A high setting will allow only larger blobs, a low setting will also allow smaller blobs.<br>The minimum blob size to be considered a potential character is defined as:<br>`CharsMaxFragmentation * CharsHeight * CharsWidth` |
| SetCharsSpacingBias | Sets the **CharSpacingBias** parameter for the topology fitting algorithm, which optimises the spacing of the bounding boxes to optimally fit the detected blobs. This will determine whether the method is biased toward finding narrow spacing, wide spacing or is neutral. |
| SetCharsWidthBias | Sets the **CharsWidthBias** parameter for the topology fitting algorithm, which optimises the width of the bounding boxes to optimally fit the detected blobs. This will determine whether the method is biased toward finding narrow boxes, wide boxes or is neutral. |
| SetCharsWidthRange | Sets the range of expected character widths in pixels. |

| | |
|---|---|
| SetDetectionDelta | Sets the **DetectionDelta** parameter for the segmentation algorithm. This will determine the range of grayscale-values used to determine the stability of a blob. A low setting will make the algorithm more sensitive to noise, a high setting will make the algorithm insensitive to blobs with low contrast to the background. |
| SetDetectionMethod | Sets the **EOCR2DetectionMethod** parameter for the topology fitting algorithm, which will place textboxes on the segmentation results, matching the given topology. |
| SetMaxVariation | Sets the **maxVariation** parameter for the segmentation algorithm. This parameter determines how stable a blob in the image should be in order to be considered a potential character, a region with clearly defined edges is generally considered stable while a blurry region is not. A high setting allows more unstable blobs, a low setting allows only very stable blobs. |
| SetNumDetectionPasses | Sets the **NumDetectionPasses** parameter for the topology fitting algorithm, which will place textboxes on the segmentation results (blobs), matching the given topology. The first pass will consider all blobs, subsequent passes will only consider those blobs that are inside the textboxes from the previous pass, sometimes resulting in a more optimal fit. |
| SetRelativeSpacesWidthRange | Sets the range of expected spaces between words as a fraction of the character width. |
| SetSegmentationMethod | Sets the **EOCR2SegmentationMethod** parameter for the segmentation algorithm, which will detect blobs in the image. |

| | |
|---|---|
| SetTextAngleRange | Sets the **TextAngleRange** parameter for the topology fitting algorithm, which will attempt to find the angle of the text in the image with respect to the horizontal. This will determine the center of the range of angles that will be tested, defined as: `TextAngleRange.min() <= angle <= TextAngleRange.max()` |
| SetTextPolarity | Sets the **TextPolarity** parameter for the segmentation algorithm. This will determine whether the algorithm searches for light blobs in a dark background or for dark blobs in a light background. |
| SetTimeOut | Time-out for the EOCR2::Read, EOCR2::Detect and EOCR2::Recognize methods. |

SetTopology

# CR2::AddCharactersToDatabase

Reads reference characters from disk and adds them to the database used for text recognition. The characters can be read from a trueType (.ttf/.ttc) file or from an EasyOCR2 database file.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void AddCharactersToDatabase(
 const std::string& file
 )

void AddCharactersToDatabase(
 const std::string& file,
 Euresys::Open_eVision_2_
10::EasyOCR2CharacterFilter filter
 )
```

E Sets the topology of the text that should be found in the image. A modified version of Regex expressions are used, where:
**.**(dot) represents any character (not including a space).
O **L** represents a letter.
**Lu** represents an uppercase letter.
**Ll** represents a lowercase letter.
**N** represents a digit.
**P** represents a punctuation character !"#&'()*,-./:;<>?[\]_{|}~
**S** represents the symbols $+-<=>|~
**\n** represents a line break.

for a minimum of n characters and a maximum of m characters.
The topology "`[LuN]{3,5}PN{4} \n .{5} LL`" represents a text comprised of 2 lines:
The first line has 1 word composed of 3 to 5 uppercase alpha-numeric characters, followed by a punctuation character and 4 numbers.
The second line has 2 words. The first word comprises of 5 wild-card characters, the second word has 2 alphabetic characters (upper- or lowercase).

## Parameters

*file*

A string containing the path of the file.

*filter*

Optional parameter; an EasyOCR2CharacterFilter that tells the method which subset of the character-set should be loaded.

# EOCR2::GetCharacterDatabase

# EOCR2::SetCharacterDatabase

Sets the EOCR2CharacterDatabase used for recognizing text.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EOCR2CharacterDatabase GetCharacterDatabase()
void SetCharacterDatabase(const EOCR2CharacterDatabase& database)
```

## Remarks

Setting a new characterDatabase will overwrite the current one.

# EOCR2::GetCharsHeight

# EOCR2::SetCharsHeight

Sets the expected character height in pixels.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetCharsHeight()
void SetCharsHeight(int height)
```

# EOCR2::GetCharsMaxFragmentation

# EOCR2::SetCharsMaxFragmentation

Sets the **CharsMaxFragmentation** parameter for the segmentation algorithm. This will determine the minimum size a blob should be in order to be considered a potential character. A high setting will allow only larger blobs, a low setting will also allow smaller blobs.
The minimum blob size to be considered a potential character is defined as:
`CharsMaxFragmentation * CharsHeight * CharsWidth`

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetCharsMaxFragmentation()
void SetCharsMaxFragmentation(float charMaxFragmentation)
```

## Remarks

This parameter should be set between 0.0 and 1.0, the default setting is 0.1.

# EOCR2::GetCharsSpacingBias

# EOCR2::SetCharsSpacingBias

Sets the **CharSpacingBias** parameter for the topology fitting algorithm, which optimises the spacing of the bounding boxes to optimally fit the detected blobs. This will determine whether the method is biased toward finding narrow spacing, wide spacing or is neutral.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EasyOCR2CharSpacingBias GetCharsSpacingBias()
void SetCharsSpacingBias(Euresys::Open_eVision_2_10::EasyOCR2CharSpacingBias
charSpacingBias)
```

### Remarks

The default setting for this parameter is EasyOCR2CharSpacingBias_Neutral

# EOCR2::GetCharsWidthBias

# EOCR2::SetCharsWidthBias

Sets the **CharsWidthBias** parameter for the topology fitting algorithm, which optimises the width of the bounding boxes to optimally fit the detected blobs. This will determine whether the method is biased toward finding narrow boxes, wide boxes or is neutral.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EasyOCR2CharWidthBias GetCharsWidthBias()
void SetCharsWidthBias(Euresys::Open_eVision_2_10::EasyOCR2CharWidthBias charWidthBias)
```

### Remarks

The default setting for this parameter is EasyOCR2CharWidthBias_Neutral

# EOCR2::GetCharsWidthRange

# EOCR2::SetCharsWidthRange

Sets the range of expected character widths in pixels.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EIntegerRange& GetCharsWidthRange()
void SetCharsWidthRange(const EIntegerRange& range)
```

### Remarks

The CharsWidthRange is returned by reference, changing it will affect the internal state of the EOCR2 object.

# EOCR2::ClearCharacterDatabase

Clears the reference character database from this EOCR2 instance.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ClearCharacterDatabase(
 )
```

# EOCR2::ClearResult

Clear the current detected text if it exists.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ClearResult(
 )
```

# EOCR2::Detect

Finds the text in an image as follows:
(1) Detects potential characters in the image following the given text polarity.
(2) Fits bounding boxes to the detected characters, following the given topology and character width/height.
(3) Extracts the detected characters from the image.
The detected characters are output as an EOCR2Text structure.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EOCR2Text Detect(
 const EROIBW8& srcRoi
 )
EOCR2Text Detect(
 const EROIBW8& srcRoi,
 const ERegion& region
 )
```

## Parameters

*srcRoi*

The source image/ROI.

*region*

The region of interest where the detection is perfomed

### Remarks

The variables Topology, CharHeight, CharWidth should be set before performing this operation. If the srcRoi is smaller than 3X3, an exception will be thrown.

# EOCR2::GetDetectionDelta

# EOCR2::SetDetectionDelta

Sets the **DetectionDelta** parameter for the segmentation algorithm. This will determine the range of grayscale-values used to determine the stability of a blob. A low setting will make the algorithm more sensitive to noise, a high setting will make the algorithm insensitive to blobs with low contrast to the background.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetDetectionDelta()
void SetDetectionDelta(int delta)
```

### Remarks

This parameter should be set between 0 and 127, the default setting is 12.

# EOCR2::GetDetectionMethod

# EOCR2::SetDetectionMethod

Sets the **EOCR2DetectionMethod** parameter for the topology fitting algorithm, which will place textboxes on the segmentation results, matching the given topology.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EOCR2DetectionMethod GetDetectionMethod()
void SetDetectionMethod(Euresys::Open_eVision_2_10::EOCR2DetectionMethod method)
```

### Remarks

The default setting for this parameter is EOCR2DetectionMethod_FixedWidth.

# EOCR2::DrawDetection

Draws the bounding boxes found by the topology detection algorithm.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawDetection(
 HDC hdc,
 Euresys::Open_eVision_2_10::EasyOCR2DrawDetectionStyle style,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
void DrawDetection(
 HDC hdc,
 ERGBColor color,
 Euresys::Open_eVision_2_10::EasyOCR2DrawDetectionStyle style,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

```
void DrawDetection(
 EDrawAdapter* drawAdapter,
 Euresys::Open_eVision_2_10::EasyOCR2DrawDetectionStyle style,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

### Parameters

*hdc*

Handle of the device context on which to draw.

*style*

The style in which each detection box should be drawn.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*color*

The color in which to draw the detection.

*drawAdapter*

-

# EOCR2::DrawDetectionWithCurrentPen

Draws the bounding boxes found by the topology detection algorithm with the current pen.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawDetectionWithCurrentPen(
 HDC hdc,
 Euresys::Open_eVision_2_10::EasyOCR2DrawDetectionStyle style,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

### Parameters

*hdc*

Handle of the device context on which to draw.

*style*

The style in which each detection box should be drawn.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

# EOCR2::DrawRecognition

Draws the recognized text next to the character bounding box in the image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawRecognition(
 HDC hdc,
 Euresys::Open_eVision_2_10::EasyOCR2DrawRecognitionStyle style,
 OEV_UINT32 cHeight,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )

void DrawRecognition(
 EDrawAdapter* drawAdapter,
 Euresys::Open_eVision_2_10::EasyOCR2DrawRecognitionStyle style,
 OEV_UINT32 cHeight,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )

void DrawRecognition(
 HDC hdc,
 ERGBColor textColor,
 ERGBColor backgroundColor,
 Euresys::Open_eVision_2_10::EasyOCR2DrawRecognitionStyle style,
 OEV_UINT32 cHeight,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )

void DrawRecognition(
 EDrawAdapter* drawAdapter,
 ERGBColor textColor,
 ERGBColor backgroundColor,
 Euresys::Open_eVision_2_10::EasyOCR2DrawRecognitionStyle style,
 OEV_UINT32 cHeight,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

### Parameters

*hdc*

Handle of the device context on which to draw.

*style*

The style in which each recognition result should be drawn.

*cHeight*

The character-height with which the recognized text should be displayed.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*drawAdapter*

-

*textColor*

The color in which to draw the recognized text.

*backgroundColor*

The color of the box in which the text is displayed.

# EOCR2::DrawRecognitionWithCurrentPen

Draws the recognized text next to the character bounding box in the image with the current pen.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawRecognitionWithCurrentPen(
 HDC hdc,
 Euresys::Open_eVision_2_10::EasyOCR2DrawRecognitionStyle style,
 OEV_UINT32 cHeight,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

### Parameters

*hdc*

Handle of the device context on which to draw.

*style*

The style in which each recognition result should be drawn.

*cHeight*

The character-height with which the recognized text should be displayed.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

# EOCR2::DrawSegmentation

Draws the blobs found by the segmentation algorithm.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawSegmentation(
 HDC hdc,
 Euresys::Open_eVision_2_10::EasyOCR2DrawSegmentationStyle style,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )

void DrawSegmentation(
 HDC hdc,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EasyOCR2DrawSegmentationStyle style,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )

void DrawSegmentation(
 EDrawAdapter* drawAdapter,
 Euresys::Open_eVision_2_10::EasyOCR2DrawSegmentationStyle style,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*hdc*

Handle of the device context on which to draw.

*style*

The style in which each blob should be drawn.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*color*

The color in which to draw the segmentation.

*drawAdapter*

-

# EOCR2::DrawSegmentationWithCurrentPen

Draws the blobs found by the segmentation algorithm with the current pen.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawSegmentationWithCurrentPen(
 HDC hdc,
 Euresys::Open_eVision_2_10::EasyOCR2DrawSegmentationStyle style,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*hdc*

Handle of the device context on which to draw.

*style*

The style in which each blob should be drawn.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

# EOCR2::EOCR2

Constructs an EOCR2 context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EOCR2(
 )
void EOCR2(
 const EOCR2& other
 )
```

### Parameters

*other*

-

# EOCR2::HitTestChar

Tests the cursor position for the presence of a character. If one is present under the cursor, it returns true and fills the EOCR2Char object passed as parameter.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
bool HitTestChar(
 EOCR2Char& character,
 int& lineN,
 int& wordN,
 int& charN,
 int x,
 int y,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*character*

Returns the character if one was detected under the cursor.

*lineN*

Returns the line-number of the character if one was detected under the cursor.

*wordN*

Returns the word-number of the character if one was detected under the cursor.

*charN*

Returns the character-number of the character if one was detected under the cursor.

*x*

Horizontal position of the cursor.

*y*

Vertical position of the cursor.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

# EOCR2::HitTestLine

Tests the cursor position for the presence of a line. If one is present under the cursor, it returns true and fills the EOCR2Line object passed as parameter.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool HitTestLine(
 EOCR2Line& line,
 int x,
 int y,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*line*

Object to fill if a line was detected under the cursor.

*x*

Horizontal position of the cursor.

*y*

Vertical position of the cursor.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

# EOCR2::HitTestText

Tests the cursor position for the presence of a text. If one is present under the cursor, it returns true and fills the EOCR2Text object passed as parameter.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool HitTestText(
 EOCR2Text& text,
 int x,
 int y,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*text*

Object to fill if a text was detected under the cursor.

*x*

Horizontal position of the cursor.

*y*

Vertical position of the cursor.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

# EOCR2::HitTestWord

Tests the cursor position for the presence of a word. If one is present under the cursor, it returns true and fills the EOCR2Word object passed as parameter.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool HitTestWord(
 EOCR2Word& word,
 int x,
 int y,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*word*

Object to fill if a word was detected under the cursor.

*x*

Horizontal position of the cursor.

*y*

Vertical position of the cursor.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

# EOCR2::Learn

Learns reference characters from a given EOCR2Text/EOCR2Line/EOCR2Word/EOCR2Char instance, containing user-specified character codes.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Learn(
 const EOCR2Char& character
 )
void Learn(
 const EOCR2Word& word
 )
void Learn(
 const EOCR2Line& line
 )
void Learn(
 const EOCR2Text& text
 )
```

## Parameters

*character*

A single EOCR2Char character, containing the detected character from the reference image as well as the corresponding character code.

*word*

A single EOCR2Word word, containing the detected characters in a single word from the reference image as well as the corresponding character codes.

*line*

A single EOCR2Line line, containing the detected characters in a single line from the reference image as well as the corresponding character codes.

*text*

A complete EOCR2Text text, containing all detected characters from the reference image as well as the corresponding character codes.

### Remarks

The EOCR2Text/EOCR2Line/EOCR2Word/EOCR2Char instance should contain detected characters from a reference image as well as their corresponding character codes.

# EOCR2::Load

Loads a model, containing parameter settings used for all operations in EOCR2, from disk.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Load(
 const std::string& modelPath
 )
```

### Parameters

*modelPath*

A string containing the full path to the model file.

# EOCR2::GetMaxVariation

# EOCR2::SetMaxVariation

Sets the **maxVariation** parameter for the segmentation algorithm. This parameter determines how stable a blob in the image should be in order to be considered a potential character, a region with clearly defined edges is generally considered stable while a blurry region is not. A high setting allows more unstable blobs, a low setting allows only very stable blobs.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetMaxVariation()
void SetMaxVariation(float maxVariation)
```

### Remarks

This parameter should be set between 0.0 and 1.0, the default setting is 0.25.

# EOCR2::GetNumDetectionPasses

# EOCR2::SetNumDetectionPasses

Sets the **NumDetectionPasses** parameter for the topology fitting algorithm, which will place textboxes on the segmentation results (blobs), matching the given topology. The first pass will consider all blobs, subsequent passes will only consider those blobs that are inside the textboxes from the previous pass, sometimes resulting in a more optimal fit.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetNumDetectionPasses()
void SetNumDetectionPasses(int nDetectionPasses)
```

### Remarks

The default setting for this parameter is 1, the setting can be either 1 or 2.

# EOCR2::operator=

Assignment operator, copies another EOCR2 instance to this one.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EOCR2& operator=(
 const EOCR2& other
 )
```

### Parameters

*other*

The EOCR2 instance to copy from.

# EOCR2::Read

Performs all steps required for reading text from an image:
(1) Detects potential characters in the image following the given text polarity and character width/height.
(2) Fits bounding boxes to the detected characters, following the given topology and character width/height.
(3) Recognizes the detected characters using the given reference character database.
The read text is output as a string.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::string Read(
 const EROIBW8& srcRoi
 )
std::string Read(
 const EROIBW8& srcRoi,
 const ERegion& region
 )
```

### Parameters

*srcRoi*

The source image/ROI.

*region*

The region of interest where the reading is perfomed

### Remarks

The variables TextPolarity, Topology, CharHeight, CharWidth should be set and a reference character database should be set before performing this operation. If the srcRoi is smaller than 3X3, an exception will be thrown.

# EOCR2::GetReadText

Outputs an EOCR2Text structure containing the detailed detection and recognition results.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EOCR2Text GetReadText()
```

# EOCR2::Recognize

Recognizes the characters in a given EOCR2Text instance, based on a given reference font.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::string Recognize(
 const EOCR2Text& text
 )

std::string Recognize(
 const EOCR2Text& text,
 const EROIBW8& srcRoi
 )
```

### Parameters

*text*

EOCR2Text structure containing the detected characters from the image.

*srcRoi*

The source image/ROI.

### Remarks

A reference character database should be provided before performing this operation. The overloaded function that uses an ROI is not yet implemented.

# EOCR2::GetRelativeSpacesWidthRange

# EOCR2::SetRelativeSpacesWidthRange

Sets the range of expected spaces between words as a fraction of the character width.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EFloatRange& GetRelativeSpacesWidthRange()
void SetRelativeSpacesWidthRange(const EFloatRange& range)
```

### Remarks

This parameter only affects the detection when the detectionMethod is set to EOCR2DetectionMethod_FixedWidth. The RelativeSpacesWidthRange is returned by reference, changing it will affect the internal state of the EOCR2 object.

# EOCR2::Save

Saves the model to disk, containing the current parameter setting used for all operations in EOCR2.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Save(
 const std::string& modelPath
 )
```

### Parameters

*modelPath*

A string containing the full path to the model file.

### Remarks

It is advised to use a file extension that is non-standard (for instance *.ocr2)

# EOCR2::SaveCharacterDatabase

Saves the current reference character database to disk.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SaveCharacterDatabase(
 const std::string& file
 )
```

### Parameters

*file*

A string containing the path of the file.

# EOCR2::GetSegmentationMethod

# EOCR2::SetSegmentationMethod

Sets the **EOCR2SegmentationMethod** parameter for the segmentation algorithm, which will detect blobs in the image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::EOCR2SegmentationMethod GetSegmentationMethod()
void SetSegmentationMethod(Euresys::Open_eVision_2_10::EOCR2SegmentationMethod method)
```

### Remarks

The default setting for this parameter is EOCR2SegmentationMethod_Local.

# EOCR2::GetTextAngleRange

# EOCR2::SetTextAngleRange

Sets the **TextAngleRange** parameter for the topology fitting algorithm, which will attempt to find the angle of the text in the image with respect to the horizontal. This will determine the center of the range of angles that will be tested, defined as:

```
TextAngleRange.min() <= angle <= TextAngleRange.max()
```

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EFloatRange& GetTextAngleRange()
void SetTextAngleRange(const EFloatRange& range)
```

### Remarks

The textAngleRange is returned by reference, changing it will affect the internal state of the EOCR2 object. The default setting for this parameter is -20° to +20°.

# EOCR2::GetTextPolarity

# EOCR2::SetTextPolarity

Sets the **TextPolarity** parameter for the segmentation algorithm. This will determine whether the algorithm searches for light blobs in a dark background or for dark blobs in a light background.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EasyOCR2TextPolarity GetTextPolarity()
void SetTextPolarity(Euresys::Open_eVision_2_10::EasyOCR2TextPolarity polarity)
```

### Remarks

Default setting is EasyOCR2TextPolarity_WhiteOnBlack.

# EOCR2::GetTimeOut

# EOCR2::SetTimeOut

Time-out for the EOCR2::Read, EOCR2::Detect and EOCR2::Recognize methods.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT64 GetTimeOut()
void SetTimeOut(OEV_UINT64 value)
```

### Remarks

If the processing time of one of these functions becomes longer than the set time-out, the processing is stopped and an exception is thrown. In that case, the error code of the exception is EError_TimeoutReached. The time-out is set in microseconds. This time-out is not a real time-out. The processing is stopped as soon as possible after the time-out has been reached. This means that the time elapsed effectively in the method can be greater than the time-out in itself.

# EOCR2::GetTopology

# EOCR2::SetTopology

Sets the topology of the text that should be found in the image. A modified version of Regex expressions are used, where:
**.**(dot) represents any character (not including a space).
**L** represents a letter.
**Lu** represents an uppercase letter.
**Ll** represents a lowercase letter.
**N** represents a digit.
**P** represents a punctuation character !"#&'()*,-./:;<>?[\]_{|}~
**S** represents the symbols $+-<=>|~
**\n** represents a line break.
' ' (space) represents a space between two words.
Combinations can be made, for example: `[LN]` represents an alpha-numeric character.
To specify multiple characters, simply add `{n}` at the end for n characters. If the amount of characters is uncertain, specify `{n,m}` for a minimum of n characters and a maximum of m characters.
The topology "`[LuN]{3,5}PN{4} \n .{5} LL`" represents a text comprised of 2 lines:
The first line has 1 word composed of 3 to 5 uppercase alpha-numeric characters, followed by a punctuation character and 4 numbers.
The second line has 2 words. The first word comprises of 5 wildcard characters, the second word has 2 alphabetic characters (upper- or lowercase).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::string GetTopology()
void SetTopology(const std::string& topology)
```

# 4.126. EOCR2Char Class

Holds all information related to a single detected character.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EOCR2Char | Constructs an EOCR2Char context. |
| GetBitmap | The bitmap associated to this character. |
| GetBoundingBox | The bounding box associated to this character. |
| GetCandidates | The list of recognition results for all reference-characters with their respective scores. |
| GetText | The true value of the character, this is used during the learning phase. |

| | |
|---|---|
| operator= | Copies all the data from another EOCR2Char object into the current EOCR2Char object |
| SetText | The true value of the character, this is used during the learning phase. |
| SetTextCode | The true value of the character, this is used during the learning phase. |

# EOCR2Char::GetBitmap

The bitmap associated to this character.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EROIBW8& GetBitmap()
```

# EOCR2Char::GetBoundingBox

The bounding box associated to this character.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
ERectangle GetBoundingBox()
```

# EOCR2Char::GetCandidates

The list of recognition results for all reference-characters with their respective scores.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::vector<Euresys::Open_eVision_2_10::EOCR2CharacterCandidate> GetCandidates()
```

# EOCR2Char::EOCR2Char

Constructs an EOCR2Char context.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EOCR2Char(
 )

void EOCR2Char(
 const EOCR2Char& other
 )
```

## Parameters

*other*

EOCR2Char object to be copied.

# EOCR2Char::operator=

Copies all the data from another EOCR2Char object into the current EOCR2Char object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EOCR2Char& operator=(
 const EOCR2Char& other
 )
```

### Parameters

*other*

EOCR2Char object to be copied

# EOCR2Char::GetText

# EOCR2Char::SetText

The true value of the character, this is used during the learning phase.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::string GetText()
void SetText(const std::string& text)
```

### Remarks

It is also possible to set the character value using a UINT16 code, this is done with EOCR2Char::TextCode.

# EOCR2Char::SetTextCode

The true value of the character, this is used during the learning phase.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetTextCode(OEV_UINT16 code)
```

### Remarks

It is also possible to set the character value using a std::string, this is done with EOCR2Char::Text.

# 4.127. EOCR2CharacterCluster Class

Holds all information related to character cluster.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddCharacter | Adds a character to the cluster. |
| Clear | Clears the cluster. |
| EOCR2CharacterCluster | Constructs an EOCR2CharacterCluster context. |

| | |
|---|---|
| GetCharacter | Returns a single character from the cluster. |
| GetCharacterCount | Returns the number of characters in this cluster. |
| GetCharacters | Returns all characters from this cluster. |
| GetCode | The ASCII code of the cluster. |
| operator= | Copies all the data from another EOCR2CharacterCluster object into the current EOCR2CharacterCluster object |
| RemoveCharacter | Removes a character from the cluster. |
| SetCode | The ASCII code of the cluster. |

E

O

# CR2CharacterCluster::AddCharacter

Adds a character to the cluster.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void AddCharacter(
 EOCR2DatabaseCharacter& character
 )
```

### Parameters

*character*

The EOCR2DatabaseCharacter to be added to the database.

# EOCR2CharacterCluster::GetCharacterCount

Returns the number of characters in this cluster.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
int GetCharacterCount()
```

# EOCR2CharacterCluster::GetCharacters

Returns all characters from this cluster.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::vector<Euresys::Open_eVision_2_10::EOCR2DatabaseCharacter> GetCharacters()
```

# EOCR2CharacterCluster::Clear

Clears the cluster.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Clear(
 )
```

# EOCR2CharacterCluster::GetCode

# EOCR2CharacterCluster::SetCode

The ASCII code of the cluster.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT16 GetCode()
void SetCode(OEV_UINT16& code)
```

# EOCR2CharacterCluster::EOCR2CharacterCluster

Constructs an EOCR2CharacterCluster context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EOCR2CharacterCluster(
 )
void EOCR2CharacterCluster(
 const EOCR2CharacterCluster& other
 )
```

### Parameters

*other*

EOCR2CharacterCluster object to be copied.

# EOCR2CharacterCluster::GetCharacter

Returns a single character from the cluster.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EOCR2DatabaseCharacter GetCharacter(
 const int& index
 )
```

### Parameters

*index*

The index of this character.

# EOCR2CharacterCluster::operator=

Copies all the data from another EOCR2CharacterCluster object into the current EOCR2CharacterCluster object

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EOCR2CharacterCluster& operator=(
 const EOCR2CharacterCluster& other
 )
```

### Parameters

*other*

EOCR2CharacterCluster object to be copied

## EOCR2CharacterCluster::RemoveCharacter

Removes a character from the cluster.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void RemoveCharacter(
 int index
 )
```

### Parameters

*index*

The index of the character to be removed.

# 4.128. EOCR2CharacterDatabase Class

Holds all information related to a character database.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddCharacter | Adds a single character to the database. |
| AddCharacters | Add characters to the database. |
| AddCluster | Adds the characters from an EOCR2CharacterCluster to the database |
| AddClusters | Adds the characters from a vector of EOCR2CharacterCluster objects to the database |
| ClearDatabase | Clears the database. |
| ClusterDatabase | Performs a clustering on the database. |
| EOCR2CharacterDatabase | Constructs an EOCR2CharacterDatabase context. |
| GetCharacter | Returns a character from the database. |
| GetCharacters | Returns all character from the database. |
| operator= | Copies all the data from another EOCR2CharacterDatabase object into the current EOCR2CharacterDatabase object |

| | |
|---|---|
| RemoveCharacter | Removes a character from the database. |
| Save | Saves the character database to disk. |

O

# CR2CharacterDatabase::AddCharacter

Adds a single character to the database.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddCharacter(
 const EOCR2DatabaseCharacter& character
 )
void AddCharacter(
 const EOCR2Char& character
 )
```

## Parameters

*character*

The EOCR2DatabaseCharacter or EOCR2Char to be added to the database.

# EOCR2CharacterDatabase::AddCharacters

Add characters to the database.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddCharacters(
 const std::vector<Euresys::Open_eVision_2_10::EOCR2DatabaseCharacter>& characters
 )
void AddCharacters(
 const std::string& path
 )
void AddCharacters(
 const std::string& path,
 Euresys::Open_eVision_2_10::EasyOCR2CharacterFilter filter
 )
void AddCharacters(
 const EOCR2Word& word
 )
void AddCharacters(
 const EOCR2Line& line
 )
void AddCharacters(
 const EOCR2Text& text
 )
```

## Parameters

*characters*

A vector of EOCR2DatabaseCharacter objects to be added to this database.

*path*

The path on disk of the character database to be added to this database.

*filter*

An EasyOCR2CharacterFilter that tells the method which subset of the character-set should be loaded.

*word*

An EOCR2Word object to be added to this database.

*line*

An EOCR2Line object to be added to this database.

*text*

An EOCR2Text object to be added to this database.

# EOCR2CharacterDatabase::AddCluster

Adds the characters from an EOCR2CharacterCluster to the database

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void AddCluster(
 const EOCR2CharacterCluster& cluster
 )
```

## Parameters

*cluster*

The EOCR2CharacterCluster to be added to the database.

# EOCR2CharacterDatabase::AddClusters

Adds the characters from a vector of EOCR2CharacterCluster objects to the database

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void AddClusters(
 const std::vector<Euresys::Open_eVision_2_10::EOCR2CharacterCluster>& clusters
 )
```

## Parameters

*clusters*

The vector of EOCR2CharacterCluster objects to be added to the database.

# EOCR2CharacterDatabase::GetCharacters

Returns all character from the database.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::vector<Euresys::Open_eVision_2_10::EOCR2DatabaseCharacter> GetCharacters()
```

# EOCR2CharacterDatabase::ClearDatabase

Clears the database.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ClearDatabase(
 )
```

# EOCR2CharacterDatabase::ClusterDatabase

Performs a clustering on the database.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::vector<Euresys::Open_eVision_2_10::EOCR2CharacterCluster> ClusterDatabase(
 const int nClusters
 )
```

### Parameters

*nClusters*

The amount of clusters to be generated.

# EOCR2CharacterDatabase::EOCR2CharacterDatabase

Constructs an EOCR2CharacterDatabase context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EOCR2CharacterDatabase(
 )
void EOCR2CharacterDatabase(
 const EOCR2CharacterDatabase& other
 )
```

### Parameters

*other*

EOCR2CharacterDatabase object to be copied.

# EOCR2CharacterDatabase::GetCharacter

Returns a character from the database.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EOCR2DatabaseCharacter GetCharacter(
 const int index
 )
```

### Parameters

*index*

The index of the character to be returned.

# EOCR2CharacterDatabase::operator=

Copies all the data from another EOCR2CharacterDatabase object into the current EOCR2CharacterDatabase object

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EOCR2CharacterDatabase& operator=(
 const EOCR2CharacterDatabase& other
 )
```

### Parameters

*other*

EOCR2CharacterDatabase object to be copied

# EOCR2CharacterDatabase::RemoveCharacter

Removes a character from the database.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveCharacter(
 const int index
 )
```

### Parameters

*index*

The index of the character to be removed.

# EOCR2CharacterDatabase::Save

Saves the character database to disk.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Save(
 const std::string& path
 )
```

### Parameters

*path*

The path of the file.

# 4.129. EOCR2DatabaseCharacter Class

Holds all information related to a database character.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EOCR2DatabaseCharacter | Constructs an EOCR2DatabaseCharacter context. |
| GetBitmap | The bitmap associated to this character. |
| GetCharacterCode | The ascii code associated to this character. |
| operator= | Copies all the data from another EOCR2DatabaseCharacter object into the current EOCR2DatabaseCharacter object |
| SetCharacterCode | The ascii code associated to this character. |

# EOCR2DatabaseCharacter::GetBitmap

The bitmap associated to this character.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EImageBW8& GetBitmap()
```

# EOCR2DatabaseCharacter::GetCharacterCode

# EOCR2DatabaseCharacter::SetCharacterCode

The ascii code associated to this character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT16 GetCharacterCode()
void SetCharacterCode(OEV_UINT16 code)
```

# EOCR2DatabaseCharacter::EOCR2DatabaseCharacter

Constructs an EOCR2DatabaseCharacter context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EOCR2DatabaseCharacter(
 )
void EOCR2DatabaseCharacter(
 const EOCR2DatabaseCharacter& other
 )
```

### Parameters

*other*

EOCR2DatabaseCharacter object to be copied.

# EOCR2DatabaseCharacter::operator=

Copies all the data from another EOCR2DatabaseCharacter object into the current EOCR2DatabaseCharacter object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EOCR2DatabaseCharacter& operator=(
 const EOCR2DatabaseCharacter& other
 )
```

### Parameters

*other*

EOCR2DatabaseCharacter object to be copied

# 4.130. EOCR2Line Class

Holds a vector of EOCR2Word objects representing a line.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EOCR2Line | Constructs an EOCR2Line context. |
| GetBoundingBox | The bounding box encapsulating all characters in the line. |
| GetText | The true value of all characters in the line. |

| | |
|---|---|
| GetWords | The vector of EOCR2Word objects representing the line. |
| operator= | Copies all the data from another EOCR2Line object into the current EOCR2Line object |
| SetText | The true value of all characters in the line. |
| SetWords | The vector of EOCR2Word objects representing the line. |

# EOCR2Line::GetBoundingBox

The bounding box encapsulating all characters in the line.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERectangle GetBoundingBox()
```

# EOCR2Line::EOCR2Line

Constructs an EOCR2Line context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EOCR2Line(
 )
void EOCR2Line(
 const EOCR2Line& other
 )
```

### Parameters

*other*

EOCR2Line object to be copied.

# EOCR2Line::operator=

Copies all the data from another EOCR2Line object into the current EOCR2Line object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EOCR2Line& operator=(
 const EOCR2Line& other
 )
```

### Parameters

*other*

EOCR2Line object to be copied

# EOCR2Line::GetText

# EOCR2Line::SetText

The true value of all characters in the line.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::string GetText()
void SetText(const std::string& text)
```

## Remarks

Use a space to separate two words.

# EOCR2Line::GetWords

# EOCR2Line::SetWords

The vector of EOCR2Word objects representing the line.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::vector<Euresys::Open_eVision_2_10::EOCR2Word> GetWords()
void SetWords(std::vector<Euresys::Open_eVision_2_10::EOCR2Word> words)
```

# 4.131. EOCR2Text Class

Holds a vector of EOCR2Line objects representing a text.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EOCR2Text | Constructs an EOCR2Text context. |
| GetBoundingBox | The bounding box encapsulating all characters in the text. |
| GetLines | The vector of EOCR2Line objects representing the text. |
| GetText | The true value of all characters in the text. |
| operator= | Copies all the data from another EOCR2Text object into the current EOCR2Text object |
| SetLines | The vector of EOCR2Line objects representing the text. |
| SetText | The true value of all characters in the text. |

# EOCR2Text::GetBoundingBox

The bounding box encapsulating all characters in the text.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERectangle GetBoundingBox()
```

# EOCR2Text::EOCR2Text

Constructs an EOCR2Text context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EOCR2Text(
 )
void EOCR2Text(
 const EOCR2Text& other
 )
```

## Parameters

*other*

EOCR2Text object to be copied.

## Remarks

Default and copy constructors.

# EOCR2Text::GetLines

# EOCR2Text::SetLines

The vector of EOCR2Line objects representing the text.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::vector<Euresys::Open_eVision_2_10::EOCR2Line> GetLines()
void SetLines(std::vector<Euresys::Open_eVision_2_10::EOCR2Line> lines)
```

# EOCR2Text::operator=

Copies all the data from another EOCR2Text object into the current EOCR2Text object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EOCR2Text& operator=(
 const EOCR2Text& other
 )
```

### Parameters

*other*

EOCR2Text object to be copied

## EOCR2Text::GetText

## EOCR2Text::SetText

The true value of all characters in the text.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::string GetText()
void SetText(const std::string& text)
```

### Remarks

Use a space (" ") to separate two words and a linebreak ("\n") to separate two lines.

# 4.132. EOCR2Word Class

Holds a vector of EOCR2Char objects representing a word.

**Namespace:** Euresys::Open_eVision_2_10

### Methods

| | |
|---|---|
| EOCR2Word | Constructs an EOCR2Word context. |

| | |
|---|---|
| GetBoundingBox | |
| | The bounding box of the word, encapsulating all characters in the word. |
| GetCharacters | The vector of EOCR2Char objects representing the word. |
| GetText | The true value of all characters in the word. |
| operator= | Copies all the data from another EOCR2Word object into the current EOCR2Word object |
| SetCharacters | The vector of EOCR2Char objects representing the word. |
| SetText | The true value of all characters in the word. |

# EOCR2Word::GetBoundingBox

The bounding box of the word, encapsulating all characters in the word.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERectangle GetBoundingBox()
```

# EOCR2Word::GetCharacters

# EOCR2Word::SetCharacters

The vector of EOCR2Char objects representing the word.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::vector<Euresys::Open_eVision_2_10::EOCR2Char> GetCharacters()
void SetCharacters(std::vector<Euresys::Open_eVision_2_10::EOCR2Char> characters)
```

# EOCR2Word::EOCR2Word

Constructs an EOCR2Word context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EOCR2Word(
 )
void EOCR2Word(
 const EOCR2Word& other
 )
```

### Parameters

*other*

EOCR2Word object to be copied.

# EOCR2Word::operator=

Copies all the data from another EOCR2Word object into the current EOCR2Word object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EOCR2Word& operator=(
 const EOCR2Word& other
 )
```

### Parameters

*other*

EOCR2Word object to be copied

# EOCR2Word::GetText

# EOCR2Word::SetText

The true value of all characters in the word.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::string GetText()
void SetText(const std::string& text)
```

# 4.133. EOCV Class

Manages a complete context for the optical character verification tool implemented in EasyOCV.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AdjustCharsLocationRanges | Adjusts the location ranges (i.e tolerances and bias) of the characters using the results of the statistical training (minimum and maximum observed values). |
| AdjustCharsQualityRanges | Adjusts the allowed ranges for the character quality indicators (i.e. tolerances and bias) using the results of the statistical training (average and standard deviation of the observed values). |
| AdjustShiftTolerances | Adjust the shift tolerances on texts from a provided ROI. |
| AdjustTextsLocationRanges | Adjusts the location ranges (i.e tolerances and bias) of texts using the results of the statistical training (minimum and maximum observed values). |
| AdjustTextsQualityRanges | Adjusts the allowed ranges for the text quality indicators (i.e. tolerances and bias) using the results of the statistical training (average and standard deviation of the observed values). |

| | |
|---|---|
| ClearStatistics | Resets the statistics accumulation. |
| ComputeDefaultTolerances | Computes or recomputes the default tolerances of the quality indicators currently in use for all characters of all texts from the given ROI and threshold. |
| CreateTemplateChars | Adds to the OCV context the characters formed from the list of free objects. |
| CreateTemplateObjects | Adds to the OCV context the objects from the list of the associated coded image. |
| CreateTemplateTexts | Adds to the OCV context a piece of text formed from the list of free characters. |
| DeleteTemplateChars | Removes free characters from the OCV context. |
| DeleteTemplateObjects | Removes free objects from the OCV context. |
| DeleteTemplateTexts | Removes template texts from the OCV context. |
| DrawTemplateChars | Draws the free characters as bounding rectangles. |
| DrawTemplateCharsWithCurrentPen | Draws the free characters as bounding rectangles. |
| DrawTemplateObjects | Draws the free objects as blobs. |

| | |
|---|---|
| DrawTemplateObjectsWithCurrentPen | Draws the free objects as blobs. |
| DrawTemplateTexts | Draws the texts as bounding rectangles. |
| DrawTemplateTextsChars | Draws the characters of the texts as bounding rectangles. |
| DrawTemplateTextsCharsWithCurrentPen | Draws the characters of the texts as bounding rectangles. |
| DrawTemplateTextsWithCurrentPen | Draws the texts as bounding rectangles. |
| DrawText | Draws a tight bounding box around a single text, possibly with the main diagonal where diagnostics occur. |
| DrawTextChars | Draws a tight bounding box around all characters of a single text, possibly with the main diagonal where diagnostics occur. |
| DrawTextCharsWithCurrentPen | Draws a tight bounding box around all characters of a single text, possibly with the main diagonal where diagnostics occur. |
| DrawTexts | Draws a tight bounding box around all texts, possibly with the main diagonal where diagnostics occur. |
| DrawTextsChars | Draws a tight bounding box around all characters of all texts. |
| DrawTextsCharsWithCurrentPen | Draws a tight bounding box around all characters of all texts. |

| | |
|---|---|
| DrawTextsWithCurrentPen | Draws a tight bounding box around all texts, possibly with the main diagonal where diagnostics occur. |
| DrawTextWithCurrentPen | Draws a tight bounding box around a single text, possibly with the main diagonal where diagnostics occur. |
| EOCV | Constructs an OCV context. |
| GatherTextsCharsParameters | Copies the parameters from the characters specified by the selection modes to the temporary character. |
| GatherTextsParameters | Copies the parameters from the texts specified by the selection mode to the temporary text. |
| GetAccurateTextsLocationScores | Current texts location scoring mode. |
| GetContrastAverage | Average contrast of last images added to statistics, or **FLOAT32_UNDEFINED** if it cannot be computed. |
| GetContrastDeviation | Standard deviation of the contrast of the last images added to statistics, or **FLOAT32_UNDEFINED** if it cannot be computed. |
| GetContrastTolerance | Allowed tolerance on the global contrast (allowed difference between the sample and template values). |

| | |
|---|---|
| GetDiagnostics | Logical combination (bitwise OR) of the defects found. |
| GetFreeChar | Gets an individual free character of the OCV context. |
| GetFreeCharCount | Return the number of free characters of the OCV context. |
| GetLocationMode | Kind of pre-processing should be applied to the sample image, as defined by ELocationMode. |
| GetNormalizeLocationScore | Current location score normalization mode. |
| GetNumTextChars | Returns the number of characters in this OCV context. |
| GetNumTexts | Number of texts in the current OCV context. |
| GetReduceLocationScore | Current location score reduction mode. |
| GetResampleChars | Character resampling mode. |
| GetSampleBackground | Reference background gray levels determined during inspection of the sample. |

| | |
|---|---|
| GetSampleContrast | Global contrast of the last inspected image, or **FLOAT32_UNDEFINED** if it has not been computed. |
| GetSampleForeground | Reference foreground gray levels determined during inspection of the sample. |
| GetSampleThreshold | Threshold level applied to the sample during inspection. |
| GetStatisticsCount | Number of samples that have been taken into account in the statistics so far. |
| GetTemplateBackground | Reference background gray levels determined during learning of the template. |
| GetTemplateContrast | Global contrast of the template image, or **FLOAT32_UNDEFINED** if it has not been computed. |
| GetTemplateForeground | Reference foreground gray levels determined during learning of the template. |
| GetTemplateImage | Source template image associated to the OCV context during model edition. |
| GetTemplateThreshold | Threshold level applied to the template during learning. |
| GetText | Returns an individual text of the OCV context. |

| | |
|---|---|
| GetTextCharParameters | Copies the parameters from the character of given index to the temporary character. |
| GetTextCharPoint | Computes the coordinates of a point located relatively to a given character. |
| GetTextParameters | Copies the parameters from the text of given index to the temporary text. |
| GetTextPoint | Computes the coordinates of a point located relatively to a given text. |
| GetUsedQualityIndicators | Choice of quality indicators. |
| GetWhiteOnBlack | Flag indicating whether the learning and inspection steps use white characters on a black background, or black characters on a white background. |
| Inspect | Inspection is the process that locates the template in the sample image using all allowed degrees of freedom, compares both images to detects and quantify defects. |
| Learn | Pre-processes the model so that all required internal data structures are set up. |
| Load | Loads the EOCV. The given ESerializer must have been created for reading. |

| | |
|---|---|
| Save | Saves the EOCV. The given ESerializer must have been created for writing. |
| ScatterTextsCharsParameters | Copies the desired parameters from the temporary character to the characters specified by the selection mode. |
| ScatterTextsParameters | Copies the desired parameters from the temporary text to the texts specified by the selection mode. |
| SelectSampleTexts | Toggles selection of the template texts whose bounding box intersects a given rectangle. |
| SelectSampleTextsChars | Toggles selection of the template text characters whose bounding box intersects a given rectangle. |
| SelectTemplateChars | Toggles selection of the free characters whose bounding box intersects a given rectangle. |
| SelectTemplateObjects | Toggles selection of the objects whose bounding box intersects a given rectangle. |
| SelectTemplateTexts | Toggles selection of the template texts whose bounding box intersects a given rectangle. |
| SetAccurateTextsLocationScores | Current texts location scoring mode. |

| | |
|---|---|
| SetContrastTolerance | Allowed tolerance on the global contrast (allowed difference between the sample and template values). |
| SetFreeChar | Sets an individual free character of the OCV context. |
| SetLocationMode | Kind of pre-processing should be applied to the sample image, as defined by ELocationMode. |
| SetNormalizeLocationScore | Current location score normalization mode. |
| SetReduceLocationScore | Current location score reduction mode. |
| SetResampleChars | Character resampling mode. |
| SetSampleBackground | Reference background gray levels determined during inspection of the sample. |
| SetSampleForeground | Reference foreground gray levels determined during inspection of the sample. |
| SetSampleThreshold | Threshold level applied to the sample during inspection. |
| SetTemplateBackground | Reference background gray levels determined during learning of the template. |

| | |
|---|---|
| SetTemplateForeground | Reference foreground gray levels determined during learning of the template. |
| SetTemplateImage | Source template image associated to the OCV context during model edition. |
| SetTemplateThreshold | Threshold level applied to the template during learning. |
| SetText | Modifies the individual text at the given index by the given value. |
| SetTextCharParameters | Copies the desired parameters from the temporary character to the character of given index. |
| SetTextParameters | Copies the desired parameters from the temporary text to the text of given index. |
| SetUsedQualityIndicators | Choice of quality indicators. |
| SetWhiteOnBlack | Flag indicating whether the learning and inspection steps use white characters on a black background, or black characters on a white background. |
| UpdateStatistics | Adds the parameters of the last inspection to the statistics. |

# EOCV::GetAccurateTextsLocationScores

# EOCV::SetAccurateTextsLocationScores

Current texts location scoring mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetAccurateTextsLocationScores()
void SetAccurateTextsLocationScores(BOOL bAccurateTextsLocationScores)
```

### Remarks

During text location, the text location score is computed before any individual character displacement is allowed. This results in lower location scores because the matching between undeformed template and sample texts is less accurate. When turned on, this property allows the score to be corrected, by taking into account the character displacements. This allows more robust diagnostics based on the location score. If no movement freedom is given to the characters (no **ShiftTolerance**), this option is irrelevant.

# EOCV::AdjustCharsLocationRanges

Adjusts the location ranges (i.e tolerances and bias) of the characters using the results of the statistical training (minimum and maximum observed values).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void AdjustCharsLocationRanges(
 float factor,
 Euresys::Open_eVision_2_10::ESelectionFlag textSelection,
 Euresys::Open_eVision_2_10::ESelectionFlag charSelection
 )
```

### Parameters

*factor*

Safety factor to use when re-computing the tolerances.

*textSelection*

Tells on which texts the adjustment should be applied, as defined by ESelectionFlag.

*charSelection*

Tells on which chars the adjustment should be applied, as defined by ESelectionFlag.

### Remarks

The method reassigns bias and tolerances values according to their minimum and maximum observed values in the X and Y directions (reported by statistics). The following adjustments are made on characters whose selection state matches **textSelection** and **charSelection**: * **Shift Bias** is assigned **1/2 *(Shift Max + Shift Min)** * **Shift Tolerance** is assigned **1/2 * factor * (Shift Max - Shift Min)**Defined this way, the allowed range is simply the observed range enlarged by the user-defined factor. The default value for **factor** is **1.2** (+20 % allowance).

# EOCV::AdjustCharsQualityRanges

Adjusts the allowed ranges for the character quality indicators (i.e. tolerances and bias) using the results of the statistical training (average and standard deviation of the observed values).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void AdjustCharsQualityRanges(
 float factor,
 Euresys::Open_eVision_2_10::ESelectionFlag textSelection,
 Euresys::Open_eVision_2_10::ESelectionFlag charSelection
 )
```

### Parameters

*factor*

Safety factor to use when re-computing the tolerances.

*textSelection*

Tells on which texts the adjustment should be applied, as defined by ESelectionFlag.

*charSelection*

Tells on which chars the adjustment should be applied, as defined by ESelectionFlag.

### Remarks

The method reassigns bias and tolerances values according to their average and standard deviation, as reported by statistics. The following adjustments are made on characters whose selection state matches **textSelection** and **charSelection**: * the bias parameter value is assigned the corresponding parameter average * the tolerance parameter value is assigned the product of **factor** by its corresponding standard deviation Please note that **3** is a minimum value for **factor** (3sigma rule).This value should be increased if deviations are computed from a small number of samples.

# EOCV::AdjustShiftTolerances

Adjust the shift tolerances on texts from a provided ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AdjustShiftTolerances(
 EROIBW8* roi
 )
```

### Parameters

*roi*

Pointer on the ROI from which the shift tolerances will be computed.

### Remarks

The method performs the computation of texts shift tolerances, assuming that the specified ROI always contains the whole text (text cannot lie outside of the ROI). Other text location parameters (skew, scale, shear) are not taken in account, and this method should not be called if these parameters tolerances are not set to zero. The method also assumes texts always have the same displacement, and does not affect characters location tolerances.

# EOCV::AdjustTextsLocationRanges

Adjusts the location ranges (i.e tolerances and bias) of texts using the results of the statistical training (minimum and maximum observed values).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AdjustTextsLocationRanges(
 float factor,
 Euresys::Open_eVision_2_10::ESelectionFlag selection
 )
```

### Parameters

*factor*

Safety factor to use when re-computing the tolerances.

*selection*

Tells on which texts the adjustment should be applied, as defined by ESelectionFlag.

### Remarks

The method reassigns bias and tolerances values according to their minimum and maximum observed values in the X and Y directions (reported by statistics). The following adjustments are made on texts whose selection state matches **selection**: * **Shift Bias** is assigned **1/2 * (Shift Max + Shift Min)** * **Shift Tolerance** is assigned **1/2 * factor * (Shift Max - Shift Min)**Defined this way, the allowed range is simply the observed range enlarged by the user-defined factor. The default value for **factor** is **1.2** (+20 % allowance).

# EOCV::AdjustTextsQualityRanges

Adjusts the allowed ranges for the text quality indicators (i.e. tolerances and bias) using the results of the statistical training (average and standard deviation of the observed values).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AdjustTextsQualityRanges(
 float factor,
 Euresys::Open_eVision_2_10::ESelectionFlag selection
 )
```

### Parameters

*factor*

Safety factor to use when re-computing the tolerances.

*selection*

Tells on which texts the adjustment should be applied, as defined by ESelectionFlag.

### Remarks

The method reassigns bias and tolerances values according to their average and standard deviation, as reported by statistics. The following adjustments are made on texts whose selection state matches **selection**: * the bias parameter value is assigned the corresponding parameter average * the tolerance parameter value is assigned the product of **factor** by its corresponding standard deviation Please note that **3** is a minimum value for **factor** (3sigma rule).This value should be increased if deviations are computed from a small number of samples.

# EOCV::ClearStatistics

Resets the statistics accumulation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ClearStatistics(
 )
```

### Remarks

This function must be called before a batch of inspections.

# EOCV::ComputeDefaultTolerances

Computes or recomputes the default tolerances of the quality indicators currently in use for all characters of all texts from the given ROI and threshold.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ComputeDefaultTolerances(
 EROIBW8* roi,
 OEV_UINT32 threshold
 )
```

### Parameters

*roi*

Pointer to the ROI from which the default tolerances should be computed.

*threshold*

Threshold level to use during computation, as defined by EThresholdMode.

### Remarks

An explicit call of this method is performed by method Learn at the end of the learning stage.

# EOCV::GetContrastAverage

Average contrast of last images added to statistics, or **FLOAT32_UNDEFINED** if it cannot be computed.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetContrastAverage()
```

### Remarks

The contrast is defined as the ratio of the reference gray-levels difference over their sum. This parameter is maximum (100 %) for a perfectly contrasted image (white on black).

# EOCV::GetContrastDeviation

Standard deviation of the contrast of the last images added to statistics, or **FLOAT32_UNDEFINED** if it cannot be computed.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetContrastDeviation()
```

### Remarks

The contrast is defined as the ratio of the reference gray-levels difference over their sum. This parameter is maximum (100 %) for a perfectly contrasted image (white on black).

# EOCV::GetContrastTolerance

# EOCV::SetContrastTolerance

Allowed tolerance on the global contrast (allowed difference between the sample and template values).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetContrastTolerance()
void SetContrastTolerance(float contrastTolerance)
```

### Remarks

The contrast is defined as the ratio of the reference gray-levels difference over their sum. This parameter is maximum (100 %) for a perfectly contrasted image (white on black).

# EOCV::CreateTemplateChars

Adds to the OCV context the characters formed from the list of free objects.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void CreateTemplateChars(
 Euresys::Open_eVision_2_10::ESelectionFlag objectsSelectionFlag,
 Euresys::Open_eVision_2_10::ECharCreationMode creationMode
 )
```

### Parameters

*objectsSelectionFlag*

Selection mode for the objects, as defined by ESelectionFlag. By default, only the selected objects are created.

*creationMode*

Free objects grouping criterion, as defined by ECharCreationMode. By default, the free objects whose bounding rectangle overlap are considered as belonging to the same character.

### Remarks

The free characters are sets of blobs taken from the free objects list. One step of the model definition is to specify which free objects will be handled by the OCV context and how they will be grouped to form characters. Grouping can be done automatically by using a overlapping criterion, or explicitly. When an object is added to a character, it is removed from the free objects list.

# EOCV::CreateTemplateObjects

Adds to the OCV context the objects from the list of the associated coded image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void CreateTemplateObjects(
 ECodedImage* codedImage,
 Euresys::Open_eVision_2_10::ESelectionFlag codedObjectsSelectionFlag
 )
```

### Parameters

*codedImage*

An **ECodedImage** object.

*codedObjectsSelectionFlag*

Selection mode for the objects, as defined by ESelectionFlag. By default, only the selected objects are created.

### Remarks

The free objects are blobs built using a separate coded image. One step of the model definition is to specify which objects will be handled by the OCV context.

# EOCV::CreateTemplateTexts

Adds to the OCV context a piece of text formed from the list of free characters.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void CreateTemplateTexts(
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag
 )
```

### Parameters

*charsSelectionFlag*

Selection mode for the free characters, as defined by ESelectionFlag. By default, only the selected free characters are processed.

### Remarks

The template texts are sets of characters taken from the free characters list. One step of the model definition is to specify which free characters will be handled by the OCV context and how they will be grouped to form texts. Grouping must be done explicitly. When a character is added to a text, it is removed from the free characters list.

# EOCV::DeleteTemplateChars

Removes free characters from the OCV context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DeleteTemplateChars(
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag
 )
```

### Parameters

*charsSelectionFlag*

Selection mode for the free characters, as defined by ESelectionFlag. By default, only the selected free characters are deleted.

### Remarks

The free characters are sets of blobs taken from the free objects list. One step of the model definition is to specify which free objects will be handled by the OCV context and how they will be grouped to form characters. Grouping can be done automatically by using a overlapping criterion, or explicitly. When an object is added to a character, it is removed from the free objects list.

# EOCV::DeleteTemplateObjects

Removes free objects from the OCV context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DeleteTemplateObjects(
 Euresys::Open_eVision_2_10::ESelectionFlag objectsSelectionFlag
 )
```

### Parameters

*objectsSelectionFlag*

Selection mode for the objects, as defined by ESelectionFlag. By default, only the selected objects are deleted.

### Remarks

The free objects are blobs built using a separate coded image. One step of the model definition is to specify which objects will be handled by the OCV context.

# EOCV::DeleteTemplateTexts

Removes template texts from the OCV context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DeleteTemplateTexts(
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag
 )
```

### Parameters

*textsSelectionFlag*

Selection mode for the template texts, as defined by ESelectionFlag. By default, only the selected texts are deleted.

### Remarks

The template texts are sets of characters taken from the free characters list. One step of the model definition is to specify which free characters will be handled by the OCV context and how they will be grouped to form texts. Grouping must be done explicitly. When a character is added to a text, it is removed from the free characters list.

# EOCV::GetDiagnostics

Logical combination (bitwise OR) of the defects found.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetDiagnostics()
```

## Remarks

Inspection is the process that locates the template in the sample image, using all allowed degrees of freedom, and that compares both images to detects and quantify defects. After inspection, all defective texts and text characters are selected. Further, the diagnostics binary mask contain a detailed interpretation of the defects found.

# EOCV::DrawTemplateChars

Draws the free characters as bounding rectangles.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawTemplateChars(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

```
void DrawTemplateChars(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )

void DrawTemplateChars(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Device context of the drawing window.

*charsSelectionFlag*

Selection mode for the free characters, as defined by ESelectionFlag. By default, only the selected characters are drawn.

*zoomX*

Zooming parameters.

*zoomY*

Zooming parameters.

*originX*

Panning parameters.

*originY*

Panning parameters.

*color*

-

## Remarks

The template drawing methods are used to graphically represent the free objects, free characters, texts and text characters associated to the OCV context before learning. All drawing operations are done using the current pen attributes.

# EOCV::DrawTemplateCharsWithCurrentPen

Draws the free characters as bounding rectangles.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawTemplateCharsWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Device context of the drawing window.

*charsSelectionFlag*

Selection mode for the free characters, as defined by ESelectionFlag. By default, only the selected characters are drawn.

*zoomX*

Zooming parameters.

*zoomY*

Zooming parameters.

*originX*

Panning parameters.

*originY*

Panning parameters.

## Remarks

The template drawing methods are used to graphically represent the free objects, free characters, texts and text characters associated to the OCV context before learning. All drawing operations are done using the current pen attributes.

# EOCV::DrawTemplateObjects

Draws the free objects as blobs.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawTemplateObjects(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag objectsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
void DrawTemplateObjects(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::ESelectionFlag objectsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
void DrawTemplateObjects(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag objectsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Device context of the drawing window.

*objectsSelectionFlag*

Selection mode for the objects, as defined by ESelectionFlag. By default, only the selected objects are drawn.

*zoomX*

Zooming parameters.

*zoomY*

Zooming parameters.

*originX*

Panning parameters.

*originY*

Panning parameters.

*color*

The color in which to draw the overlay.

### Remarks

The associated coded image must be present. The template drawing methods are used to graphically represent the free objects, free characters, texts and text characters associated to the OCV context before learning. All drawing operations are done using the current pen attributes.

# EOCV::DrawTemplateObjectsWithCurrentPen

Draws the free objects as blobs.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawTemplateObjectsWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag objectsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Device context of the drawing window.

*objectsSelectionFlag*

Selection mode for the objects, as defined by ESelectionFlag. By default, only the selected objects are drawn.

*zoomX*

Zooming parameters.

*zoomY*

Zooming parameters.

*originX*

Panning parameters.

*originY*

Panning parameters.

### Remarks

The associated coded image must be present. The template drawing methods are used to graphically represent the free objects, free characters, texts and text characters associated to the OCV context before learning. All drawing operations are done using the current pen attributes.

# EOCV::DrawTemplateTexts

Draws the texts as bounding rectangles.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawTemplateTexts(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

```
void DrawTemplateTexts(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )

void DrawTemplateTexts(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Device context of the drawing window.

*textsSelectionFlag*

Selection mode for the free texts, as defined by ESelectionFlag. By default, only the selected texts are drawn.

*zoomX*

Zooming parameters.

*zoomY*

Zooming parameters.

*originX*

Panning parameters.

*originY*

Panning parameters.

*color*

The color in which to draw the overlay.

## Remarks

The template drawing methods are used to graphically represent the free objects, free characters, texts and text characters associated to the OCV context before learning. All drawing operations are done using the current pen attributes.

# EOCV::DrawTemplateTextsChars

Draws the characters of the texts as bounding rectangles.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawTemplateTextsChars(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
void DrawTemplateTextsChars(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
void DrawTemplateTextsChars(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Device context of the drawing window.

*textsSelectionFlag*

Selection mode for the texts, as defined by ESelectionFlag. By default, only characters from the selected texts are drawn.

*charsSelectionFlag*

Selection mode for the characters, as defined by ESelectionFlag. By default, only the selected characters of the selected texts are drawn.

*zoomX*

Zooming parameters.

*zoomY*

Zooming parameters.

*originX*

Panning parameters.

*originY*

Panning parameters.

*color*

The color in which to draw the overlay.

### Remarks

The template drawing methods are used to graphically represent the free objects, free characters, texts and text characters associated to the OCV context before learning. All drawing operations are done using the current pen attributes.

# EOCV::DrawTemplateTextsCharsWithCurrentPen

Draws the characters of the texts as bounding rectangles.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawTemplateTextsCharsWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Device context of the drawing window.

*textsSelectionFlag*

Selection mode for the texts, as defined by ESelectionFlag. By default, only characters from the selected texts are drawn.

*charsSelectionFlag*

Selection mode for the characters, as defined by ESelectionFlag. By default, only the selected characters of the selected texts are drawn.

*zoomX*

Zooming parameters.

*zoomY*

Zooming parameters.

*originX*

Panning parameters.

*originY*

Panning parameters.

### Remarks

The template drawing methods are used to graphically represent the free objects, free characters, texts and text characters associated to the OCV context before learning. All drawing operations are done using the current pen attributes.

# EOCV::DrawTemplateTextsWithCurrentPen

Draws the texts as bounding rectangles.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawTemplateTextsWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Device context of the drawing window.

*textsSelectionFlag*

Selection mode for the free texts, as defined by ESelectionFlag. By default, only the selected texts are drawn.

*zoomX*

Zooming parameters.

*zoomY*

Zooming parameters.

*originX*

Panning parameters.

*originY*

Panning parameters.

### Remarks

The template drawing methods are used to graphically represent the free objects, free characters, texts and text characters associated to the OCV context before learning. All drawing operations are done using the current pen attributes.

# EOCV::DrawText

Draws a tight bounding box around a single text, possibly with the main diagonal where diagnostics occur.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawText(
 HDC graphicContext,
 EOCVText& text,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )

void DrawText(
 HDC graphicContext,
 const ERGBColor& color,
 EOCVText& text,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )

void DrawText(
 EDrawAdapter* graphicContext,
 EOCVText& text,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Device context of the drawing window.

*text*

Reference to the desired text can be obtained by using EOCV::GetText.

*zoomX*

Zooming parameters.

*zoomY*

Zooming parameters.

*originX*

Panning parameters.

*originY*

Panning parameters.

*color*

The color in which to draw the overlay.

### Remarks

The sample drawing member functions are used to graphically represent the texts and text characters located in the image during inspection. All drawing operations are done using the current pen attributes. Normally, the characters and texts are drawn as a bounding box. However, when diagnostics have been raised on a character or text, the main diagonal (north-west to south-east) is drawn as well, indicating that the item is rejected.

# EOCV::DrawTextChars

Draws a tight bounding box around all characters of a single text, possibly with the main diagonal where diagnostics occur.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawTextChars(
 HDC graphicContext,
 EOCVText& text,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )

void DrawTextChars(
 HDC graphicContext,
 const ERGBColor& color,
 EOCVText& text,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

```
void DrawTextChars(
 EDrawAdapter* graphicContext,
 EOCVText& text,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination window.

*text*

Reference to the desired text.

*charsSelectionFlag*

Characters selection mode, as defined by ESelectionFlag.

*zoomX*

Zooming parameters.

*zoomY*

Zooming parameters.

*originX*

Panning parameters.

*originY*

Panning parameters.

*color*

The color in which to draw the overlay.

## Remarks

The sample drawing methods are used to graphically represent the texts and text characters located in the image during inspection. All drawing operations are done using the current pen attributes. Normally, the characters and texts are drawn as a bounding box. However, when diagnostics have been raised on a character or text, the main diagonal (north west to south east) is drawn as well, indicating that the item is rejected.

# EOCV::DrawTextCharsWithCurrentPen

Draws a tight bounding box around all characters of a single text, possibly with the main diagonal where diagnostics occur.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawTextCharsWithCurrentPen(
 HDC graphicContext,
 EOCVText& text,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination window.

*text*

Reference to the desired text.

*charsSelectionFlag*

Characters selection mode, as defined by ESelectionFlag.

*zoomX*

Zooming parameters.

*zoomY*

Zooming parameters.

*originX*

Panning parameters.

*originY*

Panning parameters.

### Remarks

The sample drawing methods are used to graphically represent the texts and text characters located in the image during inspection. All drawing operations are done using the current pen attributes. Normally, the characters and texts are drawn as a bounding box. However, when diagnostics have been raised on a character or text, the main diagonal (north west to south east) is drawn as well, indicating that the item is rejected.

# EOCV::DrawTexts

Draws a tight bounding box around all texts, possibly with the main diagonal where diagnostics occur.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawTexts(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )

void DrawTexts(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )

void DrawTexts(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Device context of the drawing window.

*textsSelectionFlag*

Texts selection mode, as defined by ESelectionFlag.

*zoomX*

Zooming parameters.

*zoomY*

Zooming parameters.

*originX*

Panning parameters.

*originY*

Panning parameters.

*color*

The color in which to draw the overlay.

### Remarks

The sample drawing methods are used to graphically represent the texts and text characters located in the image during inspection. All drawing operations are done using the current pen attributes. Normally, the characters and texts are drawn as a bounding box. However, when diagnostics have been raised on a character or text, the main diagonal (north west to south east) is drawn as well, indicating that the item is rejected.

# EOCV::DrawTextsChars

Draws a tight bounding box around all characters of all texts.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawTextsChars(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )

void DrawTextsChars(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )

void DrawTextsChars(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Device context of the drawing window.

*textsSelectionFlag*

Texts selection mode, as defined by ESelectionFlag.

*charsSelectionFlag*

Characters selection mode, as defined by ESelectionFlag.

*zoomX*

Zooming parameters.

*zoomY*

Zooming parameters.

*originX*

Panning parameters.

*originY*

Panning parameters.

*color*

The color in which to draw the overlay.

### Remarks

The sample drawing methods are used to graphically represent the texts and text characters located in the image during inspection. All drawing operations are done using the current pen attributes. Normally, the characters and texts are drawn as a bounding box. However, when diagnostics have been raised on a character or text, the main diagonal (north west to south east) is drawn as well, indicating that the item is rejected.

# EOCV::DrawTextsCharsWithCurrentPen

Draws a tight bounding box around all characters of all texts.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawTextsCharsWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Device context of the drawing window.

*textsSelectionFlag*

Texts selection mode, as defined by ESelectionFlag.

*charsSelectionFlag*

Characters selection mode, as defined by ESelectionFlag.

*zoomX*

Zooming parameters.

*zoomY*

Zooming parameters.

*originX*

Panning parameters.

*originY*

Panning parameters.

### Remarks

The sample drawing methods are used to graphically represent the texts and text characters located in the image during inspection. All drawing operations are done using the current pen attributes. Normally, the characters and texts are drawn as a bounding box. However, when diagnostics have been raised on a character or text, the main diagonal (north west to south east) is drawn as well, indicating that the item is rejected.

# EOCV::DrawTextsWithCurrentPen

Draws a tight bounding box around all texts, possibly with the main diagonal where diagnostics occur.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawTextsWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Device context of the drawing window.

*textsSelectionFlag*

Texts selection mode, as defined by ESelectionFlag.

*zoomX*

Zooming parameters.

*zoomY*

Zooming parameters.

*originX*

Panning parameters.

*originY*

Panning parameters.

### Remarks

The sample drawing methods are used to graphically represent the texts and text characters located in the image during inspection. All drawing operations are done using the current pen attributes. Normally, the characters and texts are drawn as a bounding box. However, when diagnostics have been raised on a character or text, the main diagonal (north west to south east) is drawn as well, indicating that the item is rejected.

# EOCV::DrawTextWithCurrentPen

Draws a tight bounding box around a single text, possibly with the main diagonal where diagnostics occur.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawTextWithCurrentPen(
 HDC graphicContext,
 EOCVText& text,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Device context of the drawing window.

*text*

Reference to the desired text can be obtained by using EOCV::GetText.

*zoomX*

Zooming parameters.

*zoomY*

Zooming parameters.

*originX*

Panning parameters.

*originY*

Panning parameters.

### Remarks

The sample drawing member functions are used to graphically represent the texts and text characters located in the image during inspection. All drawing operations are done using the current pen attributes. Normally, the characters and texts are drawn as a bounding box. However, when diagnostics have been raised on a character or text, the main diagonal (north-west to south-east) is drawn as well, indicating that the item is rejected.

# EOCV::EOCV

Constructs an OCV context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EOCV(
 const EOCV& other
 )
void EOCV(
 )
```

### Parameters

*other*

-

### Remarks

Only the default constructor is needed.

# EOCV::GetFreeCharCount

Return the number of free characters of the OCV context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetFreeCharCount()
```

# EOCV::GatherTextsCharsParameters

Copies the parameters from the characters specified by the selection modes to the temporary character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GatherTextsCharsParameters(
 EOCVChar& ocvChar,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag
 )
```

## Parameters

*ocvChar*

Temporary EOCVChar object.

*textsSelectionFlag*

Selection mode for the texts, as defined by ESelectionFlag.

*charsSelectionFlag*

Selection mode for the text characters, as defined by ESelectionFlag.

### Remarks

For each character parameter that has a unique value among the specified characters, the corresponding value is different from undefined. Always be sure to check that a returned parameter does not have such an undefined value assigned, otherwise its contents will appear meaningless.

# EOCV::GatherTextsParameters

Copies the parameters from the texts specified by the selection mode to the temporary text.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GatherTextsParameters(
 EOCVText& Text,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag
 )
```

### Parameters

*Text*

Selection mode for the texts, as defined by ESelectionFlag.

*textsSelectionFlag*

Temporary EOCVText object.

### Remarks

For each text parameter that has a unique value among the specified texts, the corresponding value is different from undefined. Always be sure to check that a returned parameter does not have such an undefined value assigned, otherwise its contents will appear meaningless.

# EOCV::GetFreeChar

Gets an individual free character of the OCV context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EOCVChar& GetFreeChar(
 OEV_INT32 index
 )
```

### Parameters

*index*

Index of the individual free character.

### Remarks

The free characters are sets of blobs taken from the free objects list. One step of the model definition is to specify which free objects will be handled by the OCV context, and how they will be grouped to form characters. Grouping can be done automatically by using a overlapping criterion, or explicitly. When an object is added to a character, it is removed from the free objects list.

# EOCV::GetNumTextChars

Returns the number of characters in this OCV context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumTextChars(
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index of the desired text, in range **0** to **NumTexts-1**.

### Remarks

The characters are numbered from **0** to **NumTextChars** excluded.

# EOCV::GetText

Returns an individual text of the OCV context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EOCVText& GetText(
 OEV_INT32 index
 )
```

### Parameters

*index*

Index, between **0** and EOCV::NumTexts (excluded) of the text to be accessed.

# EOCV::GetTextCharParameters

Copies the parameters from the character of given index to the temporary character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetTextCharParameters(
 EOCVChar& ocvChar,
 OEV_UINT32 textIndex,
 OEV_UINT32 charIndex
 )
```

### Parameters

*ocvChar*

Temporary EOCVChar object.

*textIndex*

Character index in range **0** to **NumTexts-1**.

*charIndex*

Character index in range **0** to **NumTextChars-1**.

### Remarks

For each character parameter retrieved, the corresponding value is different from undefined. Always be sure to check that a returned parameter does not have such an undefined value assigned, otherwise its contents will appear meaningless.

# EOCV::GetTextCharPoint

Computes the coordinates of a point located relatively to a given character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetTextCharPoint(
 OEV_UINT32 textIndex,
 OEV_UINT32 charIndex,
 OEV_INT32& x,
 OEV_INT32& y,
 float reducedX,
 float reducedY,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*textIndex*

Text index in range **0..NumTexts**, excluded.

*charIndex*

Character index in range **0..NumTextChars**, excluded.

*x*

returned abscissa of the requested point.

*y*

returned ordinate of the requested point.

*reducedX*

X position of the requested point relatively to the character bounding box, as defined on the picture below.

*reducedY*

Y position of the requested point relatively to the character bounding box, as defined on the picture below

*zoomX*

Zooming parameters.

*zoomY*

Zooming parameters.

*panX*

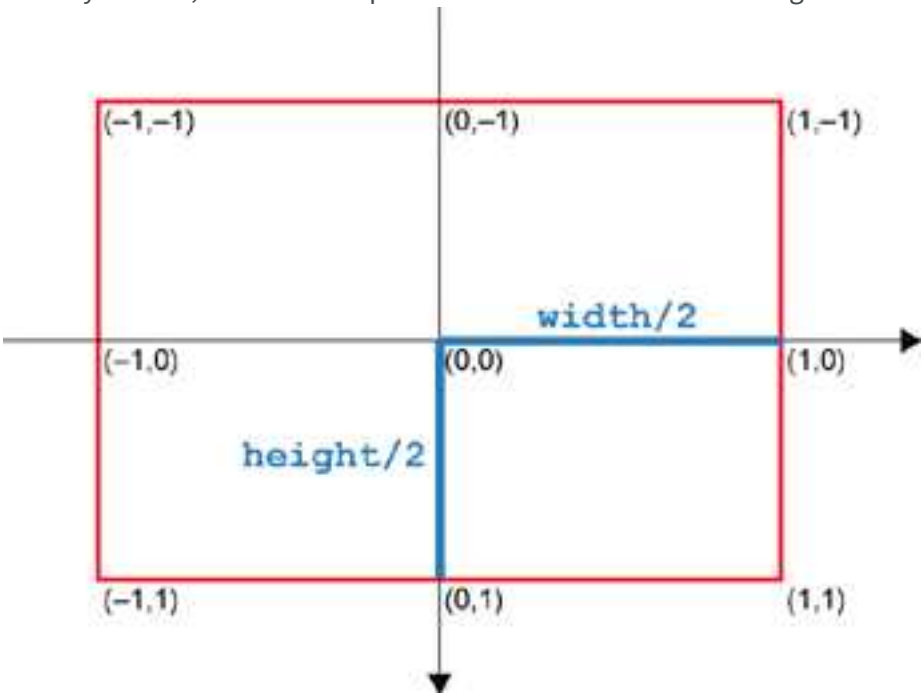
Panning parameters.

*panY*

Panning parameters.

### Remarks

The parameters **reducedX** and **reducedY** are used to indicate the position of the requested point relatively to the bounding box. These parameters can take values from **-1** to **1**, where **1** is the half height or half width of the bounding box. By default, the returned point is the center of the bounding box.




# EOCV::GetTextParameters

Copies the parameters from the text of given index to the temporary text.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetTextParameters(
 EOCVText& Text,
 OEV_UINT32 textIndex
 )
```

### Parameters

*Text*

Text index in range **0** to **NumTexts**, excluded.

*textIndex*

Text index in range **0** to **NumTexts**, excluded.

### Remarks

For each text parameter retrieved, the corresponding value is different from undefined. Always be sure to check that a returned parameter does not have such an undefined value assigned, otherwise its contents will appear meaningless.

# EOCV::GetTextPoint

Computes the coordinates of a point located relatively to a given text.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void GetTextPoint(
 OEV_UINT32 textIndex,
 OEV_INT32& x,
 OEV_INT32& y,
 float reducedX,
 float reducedY,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*textIndex*

Text index in range **0..NumTexts**, excluded.

*x*

Returned abscissa of the requested point.

*y*

Returned ordinate of the requested point.

*reducedX*

X position of the requested point relatively the text bounding box, as defined on the picture below.

*reducedY*

Y position of the requested point relatively to the text bounding box, as defined on the picture below.

*zoomX*

Zooming parameters.

*zoomY*

Zooming parameters.

*panX*

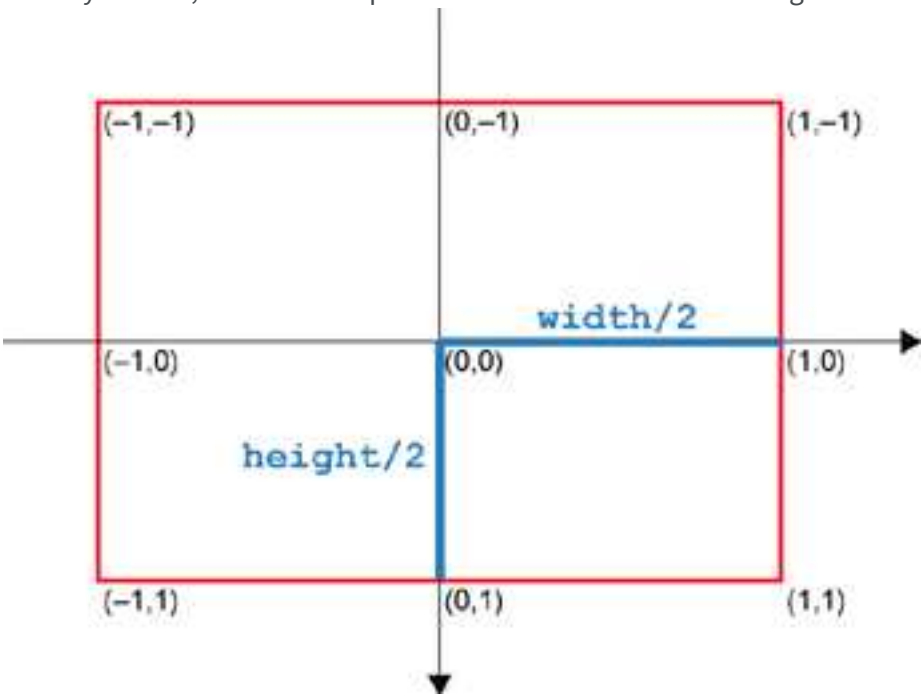
Panning parameters.

*panY*

Panning parameters.

### Remarks

The parameters **reducedX** and **reducedY** are used to indicate the position of the requested point relatively to the bounding box. These parameters can take values from **-1** to **1**, where **1** is the half height or half width of the bounding box. By default, the returned point is the center of the bounding box.




# EOCV::Inspect

Inspection is the process that locates the template in the sample image using all allowed degrees of freedom, compares both images to detects and quantify defects.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Inspect(
 EROIBW8* sampleImage,
 OEV_UINT32 threshold
 )
```

### Parameters

*sampleImage*

Pointer to the image/ROI to be inspected. The center of this image serves as a reference point on which the center of the template image is aligned.

*threshold*

Threshold level suitable for the sample image. The same convention as that of **ImgThreshold** is used for automatic thresholding.

### Remarks

After inspection, all defective texts and text characters are selected. Further, the Diagnostics binary mask contain a detailed interpretation of the defects found. A few options can be tuned to speed up the inspection process (see Location Options and Inspection Options).

# EOCV::Learn

Pre-processes the model so that all required internal data structures are set up.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Learn(
 EROIBW8* image,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag
 )
```

### Parameters

*image*

Pointer to the template image/ROI.

*textsSelectionFlag*

Texts selection mode, as defined by ESelectionFlag.

*charsSelectionFlag*

Characters selection mode, as defined by ESelectionFlag.

### Remarks

After learning, the model may not be edited anymore. It can be saved for later use and becomes ready for the inspection task. Learning is the final step of the model editing.

# EOCV::Load

Loads the EOCV. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from.

# EOCV::GetLocationMode

# EOCV::SetLocationMode

Kind of pre-processing should be applied to the sample image, as defined by ELocationMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::ELocationMode GetLocationMode()
void SetLocationMode(Euresys::Open_eVision_2_10::ELocationMode locationMode)
```

### Remarks

The location process is an important phase of the inspection task. It allows the OCV context to accurately find the template position in the sample image, using all allowed degrees of freedom. This process can be tuned in several ways to trade speed for robustness.

# EOCV::GetNormalizeLocationScore

# EOCV::SetNormalizeLocationScore

Current location score normalization mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetNormalizeLocationScore()
void SetNormalizeLocationScore(BOOL bNormalizeLocationScore)
```

### Remarks

Location score normalization performs a score correction, using the sample and template reference gray levels. It is intended to correct potential contrast or intensity discrepancies between template and sample images. It acts as if the sample image had the same foreground and background reference gray levels than the template image. It is recommended to turn this option on if the acquisition conditions can change from one sample to another. However, the correction may have a bad effect if the image acquisition system (camera) uses gamma correction, or the acquired image is saturated. The *location score normalization* mode is independent of the *location score reduction* mode.

# EOCV::GetNumTexts

Number of texts in the current OCV context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumTexts()
```

### Remarks

The texts are numbered from **0** to **NumTexts-1**.

# EOCV::GetReduceLocationScore

# EOCV::SetReduceLocationScore

Current location score reduction mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetReduceLocationScore()
void SetReduceLocationScore(BOOL bReduceLocationScore)
```

### Remarks

The location score is computed as a sum of gray-level values along the character contours. When the reduction mode is enabled (default), the location score is divided by the number of contour points, giving an average gray level, in range **0..255**. The non-reduction mode is considered obsolete. The *location score reduction* mode is independent of the *location score normalization* mode.

# EOCV::GetResampleChars

# EOCV::SetResampleChars

Character resampling mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetResampleChars()
void SetResampleChars(BOOL bResampleChars)
```

### Remarks

Normally, when text is inspected with rotation, scaling and/or shearing, some resampling must be performed when computing the quality indicators on the separate characters. If the angles remain small and the scale factors remain very close to unity, this resampling can be avoided by setting this property to **FALSE**.

# EOCV::GetSampleBackground

# EOCV::SetSampleBackground

Reference background gray levels determined during inspection of the sample.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSampleBackground()
void SetSampleBackground(float value)
```

### Remarks

Image contrast is an important factor during both learning and inspection. The background and foreground information must be separated using an appropriate threshold, possibly determined automatically. After a threshold is given, the average gray level of the background and foreground areas are computed separately. These serve as reference gray levels to measure the image contrast and normalize the gray level quality indicators. The background and foreground reference gray levels are computed for both the template and sample images. They are used for contrast assessment as well as gray-level normalization.

# EOCV::GetSampleContrast

Global contrast of the last inspected image, or **FLOAT32_UNDEFINED** if it has not been computed.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSampleContrast()
```

## Remarks

The contrast is defined as the ratio of the reference gray-levels difference over their sum. This parameter is maximum (100 %) for a perfectly contrasted image (white on black).

# EOCV::GetSampleForeground

# EOCV::SetSampleForeground

Reference foreground gray levels determined during inspection of the sample.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSampleForeground()
void SetSampleForeground(float value)
```

### Remarks

Image contrast is an important factor during both learning and inspection. The background and foreground information must be separated using an appropriate threshold, possibly determined automatically. After a threshold is given, the average gray level of the background and foreground areas are computed separately. These serve as reference gray levels to measure the image contrast and normalize the gray level quality indicators. The background and foreground reference gray levels are computed for both the template and sample images. They are used for contrast assessment as well as gray-level normalization.

# EOCV::GetSampleThreshold

# EOCV::SetSampleThreshold

Threshold level applied to the sample during inspection.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW8 GetSampleThreshold()
void SetSampleThreshold(EBW8 value)
```

### Remarks

Image contrast is an important factor during both learning and inspection. The background and foreground information must be separated using an appropriate threshold, possibly determined automatically. After a threshold is given, the average gray level of the background and foreground areas are computed separately. These serve as reference gray levels to measure the image contrast and normalize the gray level quality indicators. The background and foreground reference gray levels are computed for both the template and sample images. They are used for contrast assessment as well as gray-level normalization.

# EOCV::Save

Saves the EOCV. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is written to.

# EOCV::ScatterTextsCharsParameters

Copies the desired parameters from the temporary character to the characters specified by the selection mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ScatterTextsCharsParameters(
 EOCVChar& ocvChar,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag
 )
```

### Parameters

*ocvChar*

Temporary EOCVChar object.

*textsSelectionFlag*

Selection mode for the texts, as defined by ESelectionFlag.

*charsSelectionFlag*

Selection mode for the text characters, as defined by ESelectionFlag.

### Remarks

Only the parameters for which the value is not undefined are copied. Before setting parameters, be sure to call the EOCVChar::ResetParameters member of the temporary EOCVChar object. This will reset all parameters to the undefined value, thus avoiding unwanted copy.

# EOCV::ScatterTextsParameters

Copies the desired parameters from the temporary text to the texts specified by the selection mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ScatterTextsParameters(
 EOCVText& Text,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag
 )
```

### Parameters

*Text*

Selection mode for the texts, as defined by ESelectionFlag.

*textsSelectionFlag*

Temporary EOCVText object.

### Remarks

Only the parameters for which the value is not undefined value are copied. Before setting parameters, be sure to call the EOCVText::ResetParameters member of the temporary EOCVText object. This will reset all parameters to the undefined value, thus avoiding unwanted copy.

# EOCV::SelectSampleTexts

Toggles selection of the template texts whose bounding box intersects a given rectangle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void SelectSampleTexts(
 OEV_INT32 originX,
 OEV_INT32 originY,
 OEV_INT32 width,
 OEV_INT32 height,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag
 )
```

### Parameters

*originX*

Absolute coordinates of the selection rectangle (relative to the topmost parent image).

*originY*

Absolute coordinates of the selection rectangle (relative to the topmost parent image).

*width*

Limits of the selection rectangle.

*height*

Limits of the selection rectangle.

*textsSelectionFlag*

Selection mode for the template texts, as defined by ESelectionFlag. By default, all texts are checked for intersection

### Remarks

Unselected become selected and conversely. Reading or changing properties of the sample texts and the characters they contain can be done on basis of selection.

# EOCV::SelectSampleTextsChars

Toggles selection of the template text characters whose bounding box intersects a given rectangle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void SelectSampleTextsChars(
 OEV_INT32 originX,
 OEV_INT32 originY,
 OEV_INT32 width,
 OEV_INT32 height,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag
 )
```

### Parameters

*originX*

Absolute coordinates of the selection rectangle (relative to the topmost parent image).

*originY*

Absolute coordinates of the selection rectangle (relative to the topmost parent image).

*width*

Limits of the selection rectangle.

*height*

Limits of the selection rectangle.

*textsSelectionFlag*

Selection mode for the sample texts, as defined by ESelectionFlag. By default, all texts are checked for intersection

*charsSelectionFlag*

Selection mode for the sample text characters, as defined by ESelectionFlag. By default, all characters of all texts are checked for intersection.

### Remarks

Unselected become selected and conversely. Reading or changing properties of the sample texts and the characters they contain can be done on basis of selection.

# EOCV::SelectTemplateChars

Toggles selection of the free characters whose bounding box intersects a given rectangle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SelectTemplateChars(
 OEV_INT32 originX,
 OEV_INT32 originY,
 OEV_INT32 width,
 OEV_INT32 height,
 Euresys::Open_eVision_2_10::ESelectionFlag charsSelectionFlag
 )
```

### Parameters

*originX*

Absolute coordinates of the selection rectangle (relative to the topmost parent image).

*originY*

Absolute coordinates of the selection rectangle (relative to the topmost parent image).

*width*

Limits of the selection rectangle.

*height*

Limits of the selection rectangle.

*charsSelectionFlag*

Selection mode for the free characters, as defined by ESelectionFlag. By default, all free characters are processed.

### Remarks

Unselected becomes selected and conversely. The free characters are sets of blobs taken from the free objects list. One step of the model definition is to specify which free objects will be handled by the OCV context and how they will be grouped to form characters. Grouping can be done automatically by using a overlapping criterion, or explicitly. When an object is added to a character, it is removed from the free objects list.

# EOCV::SelectTemplateObjects

Toggles selection of the objects whose bounding box intersects a given rectangle.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SelectTemplateObjects(
 OEV_INT32 originX,
 OEV_INT32 originY,
 OEV_INT32 width,
 OEV_INT32 height,
 Euresys::Open_eVision_2_10::ESelectionFlag objectsSelectionFlag
 )
```

### Parameters

*originX*

Absolute coordinates of the selection rectangle (relative to the topmost parent image).

*originY*

Absolute coordinates of the selection rectangle (relative to the topmost parent image).

*width*

Limits of the selection rectangle.

*height*

Limits of the selection rectangle.

*objectsSelectionFlag*

Selection mode for the objects, as defined by ESelectionFlag. By default, all objects are processed.

### Remarks

Unselected becomes selected and conversely. The free objects are blobs built using a separate coded image. One step of the model definition is to specify which objects will be handled by the OCV context.

# EOCV::SelectTemplateTexts

Toggles selection of the template texts whose bounding box intersects a given rectangle.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SelectTemplateTexts(
 OEV_INT32 originX,
 OEV_INT32 originY,
 OEV_INT32 width,
 OEV_INT32 height,
 Euresys::Open_eVision_2_10::ESelectionFlag textsSelectionFlag
 )
```

### Parameters

*originX*

Absolute coordinates of the selection rectangle (relative to the topmost parent image).

*originY*

Absolute coordinates of the selection rectangle (relative to the topmost parent image).

*width*

Limits of the selection rectangle.

*height*

Limits of the selection rectangle.

*textsSelectionFlag*

Selection mode for the template texts, as defined by ESelectionFlag. By default, all texts are processed.

### Remarks

Unselected becomes selected and conversely. The template texts are sets of characters taken from the free characters list. One step of the model definition is to specify which free characters will be handled by the OCV context and how they will be grouped to form texts. Grouping must be done explicitly. When a character is added to a text, it is removed from the free characters list.

# EOCV::SetFreeChar

Sets an individual free character of the OCV context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void SetFreeChar(
 OEV_INT32 index,
 EOCVChar& freeChar
 )
```

### Parameters

*index*

Index of the individual free character.

*freeChar*

New free character.

### Remarks

The free characters are sets of blobs taken from the free objects list. One step of the model definition is to specify which free objects will be handled by the OCV context, and how they will be grouped to form characters. Grouping can be done automatically by using a overlapping criterion, or explicitly. When an object is added to a character, it is removed from the free objects list.

# EOCV::SetText

Modifies the individual text at the given index by the given value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetText(
 OEV_INT32 index,
 EOCVText& text
 )
```

### Parameters

*index*

Index, between **0** and EOCV::NumTexts (excluded) of the individual text to be modified.

*text*

The new text.

# EOCV::SetTextCharParameters

Copies the desired parameters from the temporary character to the character of given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetTextCharParameters(
 EOCVChar& ocvChar,
 OEV_UINT32 textIndex,
 OEV_UINT32 charIndex
 )
```

### Parameters

*ocvChar*

Temporary EOCVChar object.

*textIndex*

Character index in range **0** to **NumTexts-1**.

*charIndex*

Character index in range **0** to **NumTextChars-1**.

### Remarks

Only the parameters for which the value is not undefined are copied. Before setting parameters, be sure to call the EOCVChar::ResetParameters member of the temporary EOCVChar object. This will reset all parameters to the undefined value, thus avoiding unwanted copy.

# EOCV::SetTextParameters

Copies the desired parameters from the temporary text to the text of given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetTextParameters(
 EOCVText& Text,
 OEV_UINT32 textIndex
 )
```

### Parameters

*Text*

Text index in range **0** to **NumTexts**, excluded.

*textIndex*

Text index in range **0** to **NumTexts**, excluded.

### Remarks

Only the parameters for which the value is not undefined value are copied. Before setting parameters, be sure to call the EOCVText::ResetParameters member of the temporary EOCVText object. This will reset all parameters to the undefined value, thus avoiding unwanted copy.

# EOCV::GetStatisticsCount

Number of samples that have been taken into account in the statistics so far.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetStatisticsCount()
```

### Remarks

For averages to be computed, this member must amount to at least **1**. For standard deviations to be computed, this member must amount to at least **2**.

# EOCV::GetTemplateBackground

# EOCV::SetTemplateBackground

Reference background gray levels determined during learning of the template.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetTemplateBackground()
void SetTemplateBackground(float value)
```

### Remarks

Image contrast is an important factor during both learning and inspection. The background and foreground information must be separated using an appropriate threshold, possibly determined automatically. After a threshold is given, the average gray level of the background and foreground areas are computed separately. These serve as reference gray levels to measure the image contrast and normalize the gray level quality indicators. The background and foreground reference gray levels are computed for both the template and sample images. They are used for contrast assessment as well as gray-level normalization.

# EOCV::GetTemplateContrast

Global contrast of the template image, or **FLOAT32_UNDEFINED** if it has not been computed.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetTemplateContrast()
```

### Remarks

The contrast is defined as the ratio of the reference gray-levels difference over their sum. This parameter is maximum (100 %) for a perfectly contrasted image (white on black).

# EOCV::GetTemplateForeground

# EOCV::SetTemplateForeground

Reference foreground gray levels determined during learning of the template.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetTemplateForeground()
void SetTemplateForeground(float value)
```

### Remarks

Image contrast is an important factor during both learning and inspection. The background and foreground information must be separated using an appropriate threshold, possibly determined automatically. After a threshold is given, the average gray level of the background and foreground areas are computed separately. These serve as reference gray levels to measure the image contrast and normalize the gray level quality indicators. The background and foreground reference gray levels are computed for both the template and sample images. They are used for contrast assessment as well as gray-level normalization.

# EOCV::GetTemplateImage

# EOCV::SetTemplateImage

Source template image associated to the OCV context during model edition.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIBW8* GetTemplateImage()
void SetTemplateImage(EROIBW8* templateImage)
```

### Remarks

This property must be set as a first step before any model edition operation. During edition, the source image should not be altered. Before learning, the OCV context must have access to the template image to pre-process it. After learning, the OCV context keeps an internal copy for comparison purposes.

# EOCV::GetTemplateThreshold

# EOCV::SetTemplateThreshold

Threshold level applied to the template during learning.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW8 GetTemplateThreshold()
void SetTemplateThreshold(EBW8 value)
```

### Remarks

Image contrast is an important factor during both learning and inspection. The background and foreground information must be separated using an appropriate threshold, possibly determined automatically. After a threshold is given, the average gray level of the background and foreground areas are computed separately. These serve as reference gray levels to measure the image contrast and normalize the gray level quality indicators. The background and foreground reference gray levels are computed for both the template and sample images. They are used for contrast assessment as well as gray-level normalization.

# EOCV::UpdateStatistics

Adds the parameters of the last inspection to the statistics.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void UpdateStatistics(
 )
```

## Remarks

This function must be called explicitly for the current sample to be taken into account.

# EOCV::GetUsedQualityIndicators

# EOCV::SetUsedQualityIndicators

Choice of quality indicators.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetUsedQualityIndicators()
void SetUsedQualityIndicators(OEV_UINT32 un32UsedQualityIndicators)
```

## Remarks

For efficiency reasons, the quality indicators can be computed on demand only. This property is the logical combination (bitwise OR) of the values in EQualityIndicator. When a bit is set in this mask, the corresponding feature is computed.

## EOCV::GetWhiteOnBlack

## EOCV::SetWhiteOnBlack

Flag indicating whether the learning and inspection steps use white characters on a black background, or black characters on a white background.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetWhiteOnBlack()
void SetWhiteOnBlack(BOOL bWhiteOnBlack)
```

### Remarks

If **TRUE** (default), the learning and inspection steps use white characters on a black background. If **FALSE**, black characters on a white background are taken into account. It is important to note that this parameter affects the learning *and* the inspection steps. Moreover, the use of the "black characters on a white background" mode requires the choice of the correct ECodedImage class.

# 4.134. EOCVChar Class

Holds all information related to a single character, such as nominal and average position, reference quality indicators,... During the learning phases, it also keeps the list of its constituent blobs.

### Remarks

The statistics available for each character are the average, standard deviation (unbiased), minimum and maximum computed on the corresponding parameters for all images for which EOCV::UpdateStatistics has been invoked after inspection. The average is only available when at least one set of results has been accumulated. The standard deviation is only available when at least two sets of results have been accumulated.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EOCVChar | Constructs an OCVChar context. |
| GetBackgroundAreaAverage | Background area average. |
| GetBackgroundAreaDeviation | Background area standard deviation. |
| GetBackgroundAreaTolerance | Maximum allowed difference between the sample and template background areas. |
| GetBackgroundSumAverage | Background sum average. |
| GetBackgroundSumDeviation | Background sum standard deviation. |
| GetBackgroundSumTolerance | Maximum allowed difference between the sample and template background sums. |
| GetCorrelation | Normalized correlation involving the template and sample character images. |
| GetCorrelationAverage | Correlation average. |
| GetCorrelationDeviation | Correlation standard deviation. |

| | |
|---|---|
| GetCorrelationTolerance | Maximum allowed difference between the normalized correlation and unity. |
| GetDiagnostics | Logical combination (bitwise OR) of defects, as defined by EDiagnostic. |
| GetForegroundAreaAverage | Foreground area average. |
| GetForegroundAreaDeviation | Foreground area standard deviation. |
| GetForegroundAreaTolerance | Maximum allowed difference between the sample and template foreground areas. |
| GetForegroundSumAverage | Foreground sum average. |
| GetForegroundSumDeviation | Foreground sum standard deviation. |
| GetForegroundSumTolerance | Maximum allowed difference between the sample and template foreground sums. |
| GetLocationScoreAverage | Location score average. |
| GetLocationScoreDeviation | Location score standard deviation. |
| GetLocationScoreTolerance | Allowed absolute difference between the template and sample scores. |

| | |
|---|---|
| GetMarginWidth | Width of the extra space to be used around the characters bounding box (on all four sides) when computing a quality indicator. |
| GetNumContourPoints | Number of contour points of this character used for location. |
| GetSampleBackgroundArea | Number of background pixels of this character corresponding to a background pixel of the template character. |
| GetSampleBackgroundSum | Sum of the normalized gray-level values of the pixels of this character corresponding to a background pixel of the template character. |
| GetSampleForegroundArea | Number of foreground pixels of this character corresponding to a foreground pixel of the template character. |
| GetSampleForegroundSum | Sum of the normalized gray-level values of the pixels of this character corresponding to a foreground pixel of the template character. |
| GetSampleLocationScore | Value of the location score measured on the sample during inspection. |
| GetSelected | Selection state: **TRUE** when selected. |
| GetShiftX | Measured horizontal translation. |

| | |
|---|---|
| GetShiftXAverage | Shift X average. |
| GetShiftXBias | Horizontal translation bias. |
| GetShiftXDeviation | Shift X standard deviation. |
| GetShiftXMax | Maximum Shift X. |
| GetShiftXMin | Minimum Shift X. |
| GetShiftXStride | First pass stride for the horizontal translation. |
| GetShiftXTolerance | Horizontal translation tolerance. |
| GetShiftY | Measured vertical translation. |
| GetShiftYAverage | Shift Y average. |
| GetShiftYBias | Vertical translation bias. **0** corresponds to the nominal position. |
| GetShiftYDeviation | Shift Y standard deviation. |
| GetShiftYMax | Maximum Shift Y. |

| | |
|---|---|
| GetShiftYMin | Minimum Shift Y. |
| GetShiftYStride | First pass stride for the vertical translation. |
| GetShiftYTolerance | Vertical translation tolerance. |
| GetTemplateBackgroundArea | Number of pixels in the background of this character. |
| GetTemplateBackgroundSum | Sum of the normalized gray-level values of the background pixels of this character. |
| GetTemplateForegroundArea | Number of pixels in the template foreground of this character. |
| GetTemplateForegroundSum | Sum of the normalized gray-level values of the foreground pixels of this character. |
| GetTemplateLocationScore | Value of the location score measured on the template during learning. |
| GetWhiteOnBlack | Character contrast: **TRUE** for light characters on a dark background. |

| | |
|---|---|
| operator= | Copies all the data from another EOCVChar object into the current EOCVChar object |
| ResetParameters | Sets all character parameters to the undefined value for use before a EOCV::SetTextCharParameters or EOCV::ScatterTextsCharsParameters operation. |
| SetBackgroundAreaAverage | Background area average. |
| SetBackgroundAreaDeviation | Background area standard deviation. |
| SetBackgroundAreaTolerance | Maximum allowed difference between the sample and template background areas. |
| SetBackgroundSumAverage | Background sum average. |
| SetBackgroundSumDeviation | Background sum standard deviation. |
| SetBackgroundSumTolerance | Maximum allowed difference between the sample and template background sums. |
| SetCorrelation | Normalized correlation involving the template and sample character images. |
| SetCorrelationAverage | Correlation average. |

| | |
|---|---|
| SetCorrelationDeviation | Correlation standard deviation. |
| SetCorrelationTolerance | Maximum allowed difference between the normalized correlation and unity. |
| SetDiagnostics | Logical combination (bitwise OR) of defects, as defined by EDiagnostic. |
| SetForegroundAreaAverage | Foreground area average. |
| SetForegroundAreaDeviation | Foreground area standard deviation. |
| SetForegroundAreaTolerance | Maximum allowed difference between the sample and template foreground areas. |
| SetForegroundSumAverage | Foreground sum average. |
| SetForegroundSumDeviation | Foreground sum standard deviation. |
| SetForegroundSumTolerance | Maximum allowed difference between the sample and template foreground sums. |
| SetLocationScoreAverage | Location score average. |
| SetLocationScoreDeviation | Location score standard deviation. |

| | |
|---|---|
| SetLocationScoreTolerance | Allowed absolute difference between the template and sample scores. |
| SetMarginWidth | Width of the extra space to be used around the characters bounding box (on all four sides) when computing a quality indicator. |
| SetNumContourPoints | Number of contour points of this character used for location. |
| SetSampleBackgroundArea | Number of background pixels of this character corresponding to a background pixel of the template character. |
| SetSampleBackgroundSum | Sum of the normalized gray-level values of the pixels of this character corresponding to a background pixel of the template character. |
| SetSampleForegroundArea | Number of foreground pixels of this character corresponding to a foreground pixel of the template character. |
| SetSampleForegroundSum | Sum of the normalized gray-level values of the pixels of this character corresponding to a foreground pixel of the template character. |
| SetSampleLocationScore | Value of the location score measured on the sample during inspection. |
| SetSelected | Selection state: **TRUE** when selected. |

| | |
|---|---|
| SetShiftX | Measured horizontal translation. |
| SetShiftXAverage | Shift X average. |
| SetShiftXBias | Horizontal translation bias. |
| SetShiftXDeviation | Shift X standard deviation. |
| SetShiftXMax | Maximum Shift X. |
| SetShiftXMin | Minimum Shift X. |
| SetShiftXStride | First pass stride for the horizontal translation. |
| SetShiftXTolerance | Horizontal translation tolerance. |
| SetShiftY | Measured vertical translation. |
| SetShiftYAverage | Shift Y average. |
| SetShiftYBias | Vertical translation bias. **0** corresponds to the nominal position. |
| SetShiftYDeviation | Shift Y standard deviation. |

| | |
|---|---|
| SetShiftYMax | Maximum Shift Y. |
| SetShiftYMin | Minimum Shift Y. |
| SetShiftYStride | First pass stride for the vertical translation. |
| SetShiftYTolerance | Vertical translation tolerance. |
| SetTemplateBackgroundArea | Number of pixels in the background of this character. |
| SetTemplateBackgroundSum | Sum of the normalized gray-level values of the background pixels of this character. |
| SetTemplateForegroundArea | Number of pixels in the template foreground of this character. |
| SetTemplateForegroundSum | Sum of the normalized gray-level values of the foreground pixels of this character. |
| SetTemplateLocationScore | Value of the location score measured on the template during learning. |
| SetWhiteOnBlack | Character contrast: **TRUE** for light characters on a dark background. |

# EOCVChar::GetBackgroundAreaAverage

# EOCVChar::SetBackgroundAreaAverage

Background area average.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetBackgroundAreaAverage() const
void SetBackgroundAreaAverage(float val)
```

# EOCVChar::GetBackgroundAreaDeviation

# EOCVChar::SetBackgroundAreaDeviation

Background area standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetBackgroundAreaDeviation() const
void SetBackgroundAreaDeviation(float val)
```

# EOCVChar::GetBackgroundAreaTolerance

# EOCVChar::SetBackgroundAreaTolerance

Maximum allowed difference between the sample and template background areas.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetBackgroundAreaTolerance() const
void SetBackgroundAreaTolerance(OEV_UINT32 val)
```

# EOCVChar::GetBackgroundSumAverage

# EOCVChar::SetBackgroundSumAverage

Background sum average.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetBackgroundSumAverage() const
void SetBackgroundSumAverage(float val)
```

# EOCVChar::GetBackgroundSumDeviation

# EOCVChar::SetBackgroundSumDeviation

Background sum standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetBackgroundSumDeviation() const
void SetBackgroundSumDeviation(float val)
```

# EOCVChar::GetBackgroundSumTolerance

# EOCVChar::SetBackgroundSumTolerance

Maximum allowed difference between the sample and template background sums.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetBackgroundSumTolerance() const
void SetBackgroundSumTolerance(float val)
```

# EOCVChar::GetCorrelation

# EOCVChar::SetCorrelation

Normalized correlation involving the template and sample character images.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCorrelation() const
void SetCorrelation(float val)
```

# EOCVChar::GetCorrelationAverage

# EOCVChar::SetCorrelationAverage

Correlation average.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCorrelationAverage() const
void SetCorrelationAverage(float val)
```

# EOCVChar::GetCorrelationDeviation

# EOCVChar::SetCorrelationDeviation

Correlation standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCorrelationDeviation() const
void SetCorrelationDeviation(float val)
```

# EOCVChar::GetCorrelationTolerance

# EOCVChar::SetCorrelationTolerance

Maximum allowed difference between the normalized correlation and unity.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCorrelationTolerance() const
void SetCorrelationTolerance(float val)
```

# EOCVChar::GetDiagnostics

# EOCVChar::SetDiagnostics

Logical combination (bitwise OR) of defects, as defined by EDiagnostic.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetDiagnostics() const
void SetDiagnostics(OEV_UINT32 val)
```

## Remarks

After inspection, each character is tagged with a logical combination of diagnostics, each corresponding to a kind of defect.

# EOCVChar::EOCVChar

Constructs an OCVChar context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EOCVChar(
 )
void EOCVChar(
 const EOCVChar& other
 )
```

### Parameters

*other*

EOCVChar object to be copied.

### Remarks

Default and copy constructors.

# EOCVChar::GetForegroundAreaAverage

# EOCVChar::SetForegroundAreaAverage

Foreground area average.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetForegroundAreaAverage() const
void SetForegroundAreaAverage(float val)
```

# EOCVChar::GetForegroundAreaDeviation

# EOCVChar::SetForegroundAreaDeviation

Foreground area standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
float GetForegroundAreaDeviation() const
void SetForegroundAreaDeviation(float val)
```

# EOCVChar::GetForegroundAreaTolerance

# EOCVChar::SetForegroundAreaTolerance

Maximum allowed difference between the sample and template foreground areas.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetForegroundAreaTolerance() const
void SetForegroundAreaTolerance(OEV_UINT32 val)
```

# EOCVChar::GetForegroundSumAverage

# EOCVChar::SetForegroundSumAverage

Foreground sum average.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetForegroundSumAverage() const
```

```
void SetForegroundSumAverage(float val)
```

# EOCVChar::GetForegroundSumDeviation

# EOCVChar::SetForegroundSumDeviation

Foreground sum standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetForegroundSumDeviation() const
void SetForegroundSumDeviation(float val)
```

# EOCVChar::GetForegroundSumTolerance

# EOCVChar::SetForegroundSumTolerance

Maximum allowed difference between the sample and template foreground sums.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetForegroundSumTolerance() const
void SetForegroundSumTolerance(float val)
```

# EOCVChar::GetLocationScoreAverage

# EOCVChar::SetLocationScoreAverage

Location score average.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetLocationScoreAverage() const
void SetLocationScoreAverage(float val)
```

# EOCVChar::GetLocationScoreDeviation

# EOCVChar::SetLocationScoreDeviation

Location score standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetLocationScoreDeviation() const
void SetLocationScoreDeviation(float val)
```

# EOCVChar::GetLocationScoreTolerance

# EOCVChar::SetLocationScoreTolerance

Allowed absolute difference between the template and sample scores.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetLocationScoreTolerance() const
void SetLocationScoreTolerance(float val)
```

## Remarks

When the sample value lies outside the acceptance interval, a character not found diagnostic is issued. During the location phase, a score is computed on the sample image. During learning, the same score is measured on the template image to serve as a reference. The closer the template and sample scores, the more successful the location. The location score is a first indication on the conformance of the inspected sample. In particular, a very small sample score may indicate that the character is absent.

# EOCVChar::GetMarginWidth

# EOCVChar::SetMarginWidth

Width of the extra space to be used around the characters bounding box (on all four sides) when computing a quality indicator.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetMarginWidth() const
void SetMarginWidth(OEV_UINT32 val)
```

# EOCVChar::GetNumContourPoints

# EOCVChar::SetNumContourPoints

Number of contour points of this character used for location.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumContourPoints() const
void SetNumContourPoints(OEV_UINT32 val)
```

## Remarks

The location process relies on points from the external contours of the character constituent blobs. The number of points to be used can be adjusted. The smaller this value, the faster the location, but a too small value can cause mismatches.

# EOCVChar::operator=

Copies all the data from another EOCVChar object into the current EOCVChar object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EOCVChar& operator=(
 const EOCVChar& other
 )
```

### Parameters

*other*

EOCVChar object to be copied

# EOCVChar::ResetParameters

Sets all character parameters to the undefined value for use before a EOCV::SetTextCharParameters or EOCV::ScatterTextsCharsParameters operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ResetParameters(
 )
```

# EOCVChar::GetSampleBackgroundArea

# EOCVChar::SetSampleBackgroundArea

Number of background pixels of this character corresponding to a background pixel of the template character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetSampleBackgroundArea() const
void SetSampleBackgroundArea(OEV_UINT32 val)
```

# EOCVChar::GetSampleBackgroundSum

# EOCVChar::SetSampleBackgroundSum

Sum of the normalized gray-level values of the pixels of this character corresponding to a background pixel of the template character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSampleBackgroundSum() const
void SetSampleBackgroundSum(float val)
```

# EOCVChar::GetSampleForegroundArea

# EOCVChar::SetSampleForegroundArea

Number of foreground pixels of this character corresponding to a foreground pixel of the template character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
OEV_UINT32 GetSampleForegroundArea() const
void SetSampleForegroundArea(OEV_UINT32 val)
```

# EOCVChar::GetSampleForegroundSum

# EOCVChar::SetSampleForegroundSum

Sum of the normalized gray-level values of the pixels of this character corresponding to a foreground pixel of the template character.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetSampleForegroundSum() const
void SetSampleForegroundSum(float val)
```

# EOCVChar::GetSampleLocationScore

# EOCVChar::SetSampleLocationScore

Value of the location score measured on the sample during inspection.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetSampleLocationScore() const
void SetSampleLocationScore(float val)
```

### Remarks

During the location phase, a score is computed on the sample image. During learning, the same score is measured on the template image to serve as a reference. The closer the template and sample scores, the more successful the location. The location score is a first indication on the conformance of the inspected sample. In particular, a very small sample score may indicate that the character is absent.

# EOCVChar::GetSelected

# EOCVChar::SetSelected

Selection state: **TRUE** when selected.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetSelected() const
void SetSelected(BOOL val)
```

# EOCVChar::GetShiftX

# EOCVChar::SetShiftX

Measured horizontal translation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetShiftX() const
void SetShiftX(float val)
```

# EOCVChar::GetShiftXAverage

# EOCVChar::SetShiftXAverage

Shift X average.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetShiftXAverage() const
void SetShiftXAverage(float val)
```

# EOCVChar::GetShiftXBias

# EOCVChar::SetShiftXBias

Horizontal translation bias.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetShiftXBias() const
void SetShiftXBias(float val)
```

### Remarks

**0** corresponds to the nominal position.

# EOCVChar::GetShiftXDeviation

# EOCVChar::SetShiftXDeviation

Shift X standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShiftXDeviation() const
void SetShiftXDeviation(float val)
```

# EOCVChar::GetShiftXMax

# EOCVChar::SetShiftXMax

Maximum Shift X.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetShiftXMax() const
void SetShiftXMax(float val)
```

# EOCVChar::GetShiftXMin

# EOCVChar::SetShiftXMin

Minimum Shift X.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetShiftXMin() const
void SetShiftXMin(float val)
```

# EOCVChar::GetShiftXStride

# EOCVChar::SetShiftXStride

First pass stride for the horizontal translation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetShiftXStride() const
void SetShiftXStride(OEV_UINT32 val)
```

# EOCVChar::GetShiftXTolerance

# EOCVChar::SetShiftXTolerance

Horizontal translation tolerance.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetShiftXTolerance() const
void SetShiftXTolerance(float val)
```

# EOCVChar::GetShiftY

# EOCVChar::SetShiftY

Measured vertical translation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetShiftY() const
```

```
void SetShiftY(float val)
```

# EOCVChar::GetShiftYAverage

# EOCVChar::SetShiftYAverage

Shift Y average.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShiftYAverage() const
void SetShiftYAverage(float val)
```

# EOCVChar::GetShiftYBias

# EOCVChar::SetShiftYBias

Vertical translation bias. **0** corresponds to the nominal position.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShiftYBias() const
void SetShiftYBias(float val)
```

# EOCVChar::GetShiftYDeviation

# EOCVChar::SetShiftYDeviation

Shift Y standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetShiftYDeviation() const
void SetShiftYDeviation(float val)
```

# EOCVChar::GetShiftYMax

# EOCVChar::SetShiftYMax

Maximum Shift Y.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetShiftYMax() const
void SetShiftYMax(float val)
```

# EOCVChar::GetShiftYMin

# EOCVChar::SetShiftYMin

Minimum Shift Y.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShiftYMin() const
void SetShiftYMin(float val)
```

# EOCVChar::GetShiftYStride

# EOCVChar::SetShiftYStride

First pass stride for the vertical translation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetShiftYStride() const
void SetShiftYStride(OEV_UINT32 val)
```

# EOCVChar::GetShiftYTolerance

# EOCVChar::SetShiftYTolerance

Vertical translation tolerance.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShiftYTolerance() const
void SetShiftYTolerance(float val)
```

# EOCVChar::GetTemplateBackgroundArea

# EOCVChar::SetTemplateBackgroundArea

Number of pixels in the background of this character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetTemplateBackgroundArea() const
void SetTemplateBackgroundArea(OEV_UINT32 val)
```

# EOCVChar::GetTemplateBackgroundSum

# EOCVChar::SetTemplateBackgroundSum

Sum of the normalized gray-level values of the background pixels of this character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetTemplateBackgroundSum() const
void SetTemplateBackgroundSum(float val)
```

# EOCVChar::GetTemplateForegroundArea

# EOCVChar::SetTemplateForegroundArea

Number of pixels in the template foreground of this character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetTemplateForegroundArea() const
void SetTemplateForegroundArea(OEV_UINT32 val)
```

# EOCVChar::GetTemplateForegroundSum

# EOCVChar::SetTemplateForegroundSum

Sum of the normalized gray-level values of the foreground pixels of this character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetTemplateForegroundSum() const
void SetTemplateForegroundSum(float val)
```

# EOCVChar::GetTemplateLocationScore

# EOCVChar::SetTemplateLocationScore

Value of the location score measured on the template during learning.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetTemplateLocationScore() const
void SetTemplateLocationScore(float val)
```

## Remarks

If necessary, this value may be set explicitly. During the location phase, a score is computed on the sample image. During learning, the same score is measured on the template image to serve as a reference. The closer the template and sample scores, the more successful the location. The location score is a first indication on the conformance of the inspected sample. In particular, a very small sample score may indicate that the character is absent.

## EOCVChar::GetWhiteOnBlack

## EOCVChar::SetWhiteOnBlack

Character contrast: **TRUE** for light characters on a dark background.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetWhiteOnBlack() const
void SetWhiteOnBlack(BOOL val)
```

# 4.135. EOCVText Class

Holds all information related to a single piece of text, such as nominal and average position, reference quality indicators,... It also keeps the list of its constituent characters.

### Remarks

Each text can be translated horizontally and vertically, rotated, scaled horizontally and vertically and sheared with respect to its nominal position, to cope with mechanical displacement of the marking device. Location is performed in the range **Bias** +/- **Tolerance** around the nominal position. To speed up location, a two pass search may be performed, first with a large stride, then with a unit stride.

**Note.** The stride parameters apply to the two translation degrees of freedom only.
A set of parameters are computed on demand (see Inspection Options) during inspection. The values of these parameters are used to detect defects of various kinds by checking that they remain in a given tolerance interval. If not, a diagnostic is issued. Note that the quality indicators associated with the texts are the sums of the corresponding parameters for each of the constituent characters. The statistics available for each text are the average, standard deviation (unbiased), minimum and maximum computed on the corresponding parameters for all images for which EOCV::UpdateStatistics has been invoked after inspection. The average is only available when at least one set of results has been accumulated. The standard deviation is only available when at least two sets of results have been accumulated.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EOCVText | Constructs an OCVText context. |
| GetBackgroundAreaAverage | Background area average. |
| GetBackgroundAreaDeviation | Background area standard deviation. |
| GetBackgroundAreaTolerance | Maximum allowed difference between the sample and template background areas. |
| GetBackgroundSumAverage | Background sum average. |
| GetBackgroundSumDeviation | Background sum standard deviation. |
| GetBackgroundSumTolerance | Maximum allowed difference between the sample and template background sums. |
| GetCorrelation | Normalized correlation involving the template and sample images of the characters of this text. |
| GetCorrelationAverage | Correlation average. |
| GetCorrelationDeviation | Correlation standard deviation. |

| | |
|---|---|
| GetCorrelationTolerance | Maximum allowed difference between the normalized correlation and unity. |
| GetDiagnostics | Logical combination (bitwise OR) of defects, as defined by EDiagnostic. |
| GetForegroundAreaAverage | Foreground area average. |
| GetForegroundAreaDeviation | Foreground area standard deviation. |
| GetForegroundAreaTolerance | Maximum allowed difference between the sample and template foreground areas. |
| GetForegroundSumAverage | Foreground sum average. |
| GetForegroundSumDeviation | Foreground sum standard deviation. |
| GetForegroundSumTolerance | Maximum allowed difference between the sample and template foreground sums. |
| GetIsotropicScaling | Flag indicating whether the scaling degree of freedom should be considered isotropic (**ScaleX** and **ScaleY** identical) or not. |
| GetLocationScoreAverage | Location score average. |
| GetLocationScoreDeviation | Location score standard deviation. |

| | |
|---|---|
| GetLocationScoreTolerance | Allowed absolute difference between the template and sample scores. |
| GetMarginWidth | Unused. |
| GetNumContourPoints | Number of contour points used for location of this text. |
| GetSampleBackgroundArea | Number of background pixels of this text corresponding to a background pixel of the characters of the template text. |
| GetSampleBackgroundSum | Sum of the normalized gray-level values of the pixels of this text corresponding to a background pixel of the characters of the template text. |
| GetSampleForegroundArea | Number of foreground pixels of this text corresponding to a foreground pixel of the characters of the template text. |
| GetSampleForegroundSum | Sum of the normalized gray-level values of the pixels of this text corresponding to a foreground pixel of the characters of the template text. |
| GetSampleLocationScore | Value of the location score measured on the sample during inspection. |
| GetScaleX | Measured horizontal scaling, expressed as a dimensionless ratio. |

| | |
|---|---|
| GetScaleXAverage | Scale X average. |
| GetScaleXBias | Horizontal scaling bias. |
| GetScaleXCount | Number of values to be tried in the Bias +/- Tolerance range, bounds included. |
| GetScaleXDeviation | Scale X standard deviation. |
| GetScaleXMax | Maximum Y-scale. |
| GetScaleXMin | Minimum X-scale. |
| GetScaleXTolerance | Horizontal scaling tolerance. |
| GetScaleY | Measured vertical scaling, expressed as a dimensionless ratio. |
| GetScaleYAverage | Scale Y average. |
| GetScaleYBias | Vertical scaling bias. |
| GetScaleYCount | Number of values to be tried in the Bias +/- Tolerance range, bounds included. |

| | |
|---|---|
| GetScaleYDeviation | Scale Y standard deviation. |
| GetScaleYMax | Maximum Y-scale. |
| GetScaleYMin | Minimum Y-scale. |
| GetScaleYTolerance | Vertical scaling tolerance. |
| GetSelected | Selection state. **TRUE** when selected. |
| GetShear | Measured shearing, clockwise from the vertical direction, expressed in the current angle unit. |
| GetShearAverage | Shear average. |
| GetShearBias | Shearing bias. |
| GetShearCount | Number of values to be tried in the Bias +/- Tolerance range, bounds included. |
| GetShearDeviation | Shear standard deviation. |
| GetShearMax | Maximum shear. |

| | |
|---|---|
| GetShearMin | Minimum shear. |
| GetShearTolerance | Shearing tolerance. |
| GetShiftX | Measured horizontal translation, in pixels. |
| GetShiftXAverage | Shift X average. |
| GetShiftXBias | Horizontal translation bias. |
| GetShiftXDeviation | Shift X standard deviation. |
| GetShiftXMax | Maximum Shift Y. |
| GetShiftXMin | Minimum Shift X. |
| GetShiftXStride | First pass stride for the horizontal translation. |
| GetShiftXTolerance | Horizontal translation tolerance. |
| GetShiftY | Measured vertical translation, in pixels. |
| GetShiftYAverage | Shift Y average. |

| | |
|---|---|
| GetShiftYBias | Vertical translation bias. |
| GetShiftYDeviation | Shift Y standard deviation. |
| GetShiftYMax | Maximum Shift Y. |
| GetShiftYMin | Minimum Shift Y. |
| GetShiftYStride | First pass stride for the vertical translation. |
| GetShiftYTolerance | Vertical translation tolerance. |
| GetSkew | Measured rotation, clockwise from the horizontal direction, expressed in the current angle unit. |
| GetSkewAverage | Skew average. |
| GetSkewBias | Rotation bias. |
| GetSkewCount | Number of values to be tried in the Bias +/- Tolerance range, bounds included. |
| GetSkewDeviation | Skew standard deviation. |

| | |
|---|---|
| GetSkewMax | Maximum skew. |
| GetSkewMin | Minimum skew. |
| GetSkewTolerance | Rotation tolerance. |
| GetTemplateBackgroundArea | Number of pixels in the background of the characters of this text. |
| GetTemplateBackgroundSum | Sum of the normalized gray-level values of the background pixels of the characters of this text. |
| GetTemplateForegroundArea | Number of pixels in the foreground of all characters of this text. |
| GetTemplateForegroundSum | Sum of the normalized gray-level values of the foreground pixels of the characters of this text. |
| GetTemplateLocationScore | Value of the location score measured on the template during learning. If necessary, this value may be set explicitly. |
| operator= | Copies all the data from another EOCVText object into the current EOCVText object |

| | |
|---|---|
| ResetParameters | Sets all text parameters to the undefined value for use before a EOCV::SetTextParameters or EOCV::ScatterTextsParameters operation. |
| SetBackgroundAreaAverage | Background area average. |
| SetBackgroundAreaDeviation | Background area standard deviation. |
| SetBackgroundAreaTolerance | Maximum allowed difference between the sample and template background areas. |
| SetBackgroundSumAverage | Background sum average. |
| SetBackgroundSumDeviation | Background sum standard deviation. |
| SetBackgroundSumTolerance | Maximum allowed difference between the sample and template background sums. |
| SetCorrelation | Normalized correlation involving the template and sample images of the characters of this text. |
| SetCorrelationAverage | Correlation average. |
| SetCorrelationDeviation | Correlation standard deviation. |

| | |
|---|---|
| SetCorrelationTolerance | Maximum allowed difference between the normalized correlation and unity. |
| SetDiagnostics | Logical combination (bitwise OR) of defects, as defined by EDiagnostic. |
| SetForegroundAreaAverage | Foreground area average. |
| SetForegroundAreaDeviation | Foreground area standard deviation. |
| SetForegroundAreaTolerance | Maximum allowed difference between the sample and template foreground areas. |
| SetForegroundSumAverage | Foreground sum average. |
| SetForegroundSumDeviation | Foreground sum standard deviation. |
| SetForegroundSumTolerance | Maximum allowed difference between the sample and template foreground sums. |
| SetIsotropicScaling | Flag indicating whether the scaling degree of freedom should be considered isotropic (**ScaleX** and **ScaleY** identical) or not. |
| SetLocationScoreAverage | Location score average. |
| SetLocationScoreDeviation | Location score standard deviation. |

| | |
|---|---|
| SetLocationScoreTolerance | Allowed absolute difference between the template and sample scores. |
| SetMarginWidth | Unused. |
| SetNumContourPoints | Number of contour points used for location of this text. |
| SetSampleBackgroundArea | Number of background pixels of this text corresponding to a background pixel of the characters of the template text. |
| SetSampleBackgroundSum | Sum of the normalized gray-level values of the pixels of this text corresponding to a background pixel of the characters of the template text. |
| SetSampleForegroundArea | Number of foreground pixels of this text corresponding to a foreground pixel of the characters of the template text. |
| SetSampleForegroundSum | Sum of the normalized gray-level values of the pixels of this text corresponding to a foreground pixel of the characters of the template text. |
| SetSampleLocationScore | Value of the location score measured on the sample during inspection. |
| SetScaleX | Measured horizontal scaling, expressed as a dimensionless ratio. |

| | |
|---|---|
| SetScaleXAverage | Scale X average. |
| SetScaleXBias | Horizontal scaling bias. |
| SetScaleXCount | Number of values to be tried in the Bias +/- Tolerance range, bounds included. |
| SetScaleXDeviation | Scale X standard deviation. |
| SetScaleXMax | Maximum Y-scale. |
| SetScaleXMin | Minimum X-scale. |
| SetScaleXTolerance | Horizontal scaling tolerance. |
| SetScaleY | Measured vertical scaling, expressed as a dimensionless ratio. |
| SetScaleYAverage | Scale Y average. |
| SetScaleYBias | Vertical scaling bias. |
| SetScaleYCount | Number of values to be tried in the Bias +/- Tolerance range, bounds included. |

| | |
|---|---|
| SetScaleYDeviation | Scale Y standard deviation. |
| SetScaleYMax | Maximum Y-scale. |
| SetScaleYMin | Minimum Y-scale. |
| SetScaleYTolerance | Vertical scaling tolerance. |
| SetSelected | Selection state. **TRUE** when selected. |
| SetShear | Measured shearing, clockwise from the vertical direction, expressed in the current angle unit. |
| SetShearAverage | Shear average. |
| SetShearBias | Shearing bias. |
| SetShearCount | Number of values to be tried in the Bias +/- Tolerance range, bounds included. |
| SetShearDeviation | Shear standard deviation. |
| SetShearMax | Maximum shear. |

| | |
|---|---|
| SetShearMin | Minimum shear. |
| SetShearTolerance | Shearing tolerance. |
| SetShiftX | Measured horizontal translation, in pixels. |
| SetShiftXAverage | Shift X average. |
| SetShiftXBias | Horizontal translation bias. |
| SetShiftXDeviation | Shift X standard deviation. |
| SetShiftXMax | Maximum Shift Y. |
| SetShiftXMin | Minimum Shift X. |
| SetShiftXStride | First pass stride for the horizontal translation. |
| SetShiftXTolerance | Horizontal translation tolerance. |
| SetShiftY | Measured vertical translation, in pixels. |
| SetShiftYAverage | Shift Y average. |

| | |
|---|---|
| SetShiftYBias | Vertical translation bias. |
| SetShiftYDeviation | Shift Y standard deviation. |
| SetShiftYMax | Maximum Shift Y. |
| SetShiftYMin | Minimum Shift Y. |
| SetShiftYStride | First pass stride for the vertical translation. |
| SetShiftYTolerance | Vertical translation tolerance. |
| SetSkew | Measured rotation, clockwise from the horizontal direction, expressed in the current angle unit. |
| SetSkewAverage | Skew average. |
| SetSkewBias | Rotation bias. |
| SetSkewCount | Number of values to be tried in the Bias +/- Tolerance range, bounds included. |
| SetSkewDeviation | Skew standard deviation. |

| | |
|---|---|
| SetSkewMax | Maximum skew. |
| SetSkewMin | Minimum skew. |
| SetSkewTolerance | Rotation tolerance. |
| SetTemplateBackgroundArea | Number of pixels in the background of the characters of this text. |
| SetTemplateBackgroundSum | Sum of the normalized gray-level values of the background pixels of the characters of this text. |
| SetTemplateForegroundArea | Number of pixels in the foreground of all characters of this text. |
| SetTemplateForegroundSum | Sum of the normalized gray-level values of the foreground pixels of the characters of this text. |
| SetTemplateLocationScore | Value of the location score measured on the template during learning. If necessary, this value may be set explicitly. |

# EOCVText::GetBackgroundAreaAverage

# EOCVText::SetBackgroundAreaAverage

Background area average.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetBackgroundAreaAverage() const
void SetBackgroundAreaAverage(float value)
```

# EOCVText::GetBackgroundAreaDeviation

# EOCVText::SetBackgroundAreaDeviation

Background area standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetBackgroundAreaDeviation() const
void SetBackgroundAreaDeviation(float value)
```

# EOCVText::GetBackgroundAreaTolerance

# EOCVText::SetBackgroundAreaTolerance

Maximum allowed difference between the sample and template background areas.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetBackgroundAreaTolerance() const
void SetBackgroundAreaTolerance(OEV_UINT32 value)
```

# EOCVText::GetBackgroundSumAverage

# EOCVText::SetBackgroundSumAverage

Background sum average.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetBackgroundSumAverage() const
void SetBackgroundSumAverage(float value)
```

# EOCVText::GetBackgroundSumDeviation

# EOCVText::SetBackgroundSumDeviation

Background sum standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetBackgroundSumDeviation() const
void SetBackgroundSumDeviation(float value)
```

# EOCVText::GetBackgroundSumTolerance

# EOCVText::SetBackgroundSumTolerance

Maximum allowed difference between the sample and template background sums.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetBackgroundSumTolerance() const
void SetBackgroundSumTolerance(float value)
```

# EOCVText::GetCorrelation

# EOCVText::SetCorrelation

Normalized correlation involving the template and sample images of the characters of this text.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCorrelation() const
void SetCorrelation(float value)
```

# EOCVText::GetCorrelationAverage

# EOCVText::SetCorrelationAverage

Correlation average.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCorrelationAverage() const
void SetCorrelationAverage(float value)
```

# EOCVText::GetCorrelationDeviation

# EOCVText::SetCorrelationDeviation

Correlation standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCorrelationDeviation() const
void SetCorrelationDeviation(float value)
```

# EOCVText::GetCorrelationTolerance

# EOCVText::SetCorrelationTolerance

Maximum allowed difference between the normalized correlation and unity.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCorrelationTolerance() const
void SetCorrelationTolerance(float value)
```

# EOCVText::GetDiagnostics

# EOCVText::SetDiagnostics

Logical combination (bitwise OR) of defects, as defined by EDiagnostic.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetDiagnostics() const
void SetDiagnostics(OEV_UINT32 value)
```

## Remarks

After inspection, each text is tagged with a logical combination of diagnostics, each corresponding to a kind of defect. The defects of the characters belonging to this texts are also added to this combination.

# EOCVText::EOCVText

Constructs an OCVText context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EOCVText(
 )
void EOCVText(
 const EOCVText& ocvText
 )
```

### Parameters

*ocvText*

EOCVText object to be copied.

### Remarks

Default and copy constructors.

# EOCVText::GetForegroundAreaAverage

# EOCVText::SetForegroundAreaAverage

Foreground area average.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetForegroundAreaAverage() const
void SetForegroundAreaAverage(float value)
```

# EOCVText::GetForegroundAreaDeviation

# EOCVText::SetForegroundAreaDeviation

Foreground area standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
float GetForegroundAreaDeviation() const
void SetForegroundAreaDeviation(float value)
```

# EOCVText::GetForegroundAreaTolerance

# EOCVText::SetForegroundAreaTolerance

Maximum allowed difference between the sample and template foreground areas.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetForegroundAreaTolerance() const
void SetForegroundAreaTolerance(OEV_UINT32 value)
```

# EOCVText::GetForegroundSumAverage

# EOCVText::SetForegroundSumAverage

Foreground sum average.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetForegroundSumAverage() const
```

```
void SetForegroundSumAverage(float value)
```

# EOCVText::GetForegroundSumDeviation

# EOCVText::SetForegroundSumDeviation

Foreground sum standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetForegroundSumDeviation() const
void SetForegroundSumDeviation(float value)
```

# EOCVText::GetForegroundSumTolerance

# EOCVText::SetForegroundSumTolerance

Maximum allowed difference between the sample and template foreground sums.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetForegroundSumTolerance() const
void SetForegroundSumTolerance(float value)
```

# EOCVText::GetIsotropicScaling

# EOCVText::SetIsotropicScaling

Flag indicating whether the scaling degree of freedom should be considered isotropic (**ScaleX** and **ScaleY** identical) or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetIsotropicScaling()
void SetIsotropicScaling(BOOL isotropicScaling)
```

## Remarks

In most cases, isotropic scaling (default mode), is recommended. Isotropic scaling executes faster and is more realistic. In case of isotropic scaling search, the **ScaleY...** values are meaningless.

# EOCVText::GetLocationScoreAverage

# EOCVText::SetLocationScoreAverage

Location score average.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetLocationScoreAverage() const
void SetLocationScoreAverage(float value)
```

# EOCVText::GetLocationScoreDeviation

# EOCVText::SetLocationScoreDeviation

Location score standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetLocationScoreDeviation() const
void SetLocationScoreDeviation(float value)
```

# EOCVText::GetLocationScoreTolerance

# EOCVText::SetLocationScoreTolerance

Allowed absolute difference between the template and sample scores.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetLocationScoreTolerance() const
void SetLocationScoreTolerance(float value)
```

## Remarks

When the sample value lies outside the acceptance interval, a text not found diagnostic is issued.

# EOCVText::GetMarginWidth

# EOCVText::SetMarginWidth

Unused.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetMarginWidth() const
void SetMarginWidth(OEV_UINT32 value)
```

# EOCVText::GetNumContourPoints

# EOCVText::SetNumContourPoints

Number of contour points used for location of this text.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumContourPoints() const
void SetNumContourPoints(OEV_UINT32 value)
```

# EOCVText::operator=

Copies all the data from another EOCVText object into the current EOCVText object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EOCVText& operator=(
 const EOCVText& other
 )
```

### Parameters

*other*

EOCVText object to be copied

# EOCVText::ResetParameters

Sets all text parameters to the undefined value for use before a EOCV::SetTextParameters or EOCV::ScatterTextsParameters operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ResetParameters(
 )
```

# EOCVText::GetSampleBackgroundArea

# EOCVText::SetSampleBackgroundArea

Number of background pixels of this text corresponding to a background pixel of the characters of the template text.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetSampleBackgroundArea() const
void SetSampleBackgroundArea(OEV_UINT32 value)
```

# EOCVText::GetSampleBackgroundSum

# EOCVText::SetSampleBackgroundSum

Sum of the normalized gray-level values of the pixels of this text corresponding to a background pixel of the characters of the template text.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSampleBackgroundSum() const
void SetSampleBackgroundSum(float value)
```

# EOCVText::GetSampleForegroundArea

# EOCVText::SetSampleForegroundArea

Number of foreground pixels of this text corresponding to a foreground pixel of the characters of the template text.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetSampleForegroundArea() const
void SetSampleForegroundArea(OEV_UINT32 value)
```

# EOCVText::GetSampleForegroundSum

# EOCVText::SetSampleForegroundSum

Sum of the normalized gray-level values of the pixels of this text corresponding to a foreground pixel of the characters of the template text.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetSampleForegroundSum() const
void SetSampleForegroundSum(float value)
```

# EOCVText::GetSampleLocationScore

# EOCVText::SetSampleLocationScore

Value of the location score measured on the sample during inspection.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSampleLocationScore() const
void SetSampleLocationScore(float value)
```

# EOCVText::GetScaleX

# EOCVText::SetScaleX

Measured horizontal scaling, expressed as a dimensionless ratio.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScaleX() const
void SetScaleX(float value)
```

# EOCVText::GetScaleXAverage

# EOCVText::SetScaleXAverage

Scale X average.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScaleXAverage() const
void SetScaleXAverage(float value)
```

# EOCVText::GetScaleXBias

# EOCVText::SetScaleXBias

Horizontal scaling bias.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScaleXBias() const
void SetScaleXBias(float value)
```

## Remarks

**0** corresponds to the nominal, true scale factor.

# EOCVText::GetScaleXCount

# EOCVText::SetScaleXCount

Number of values to be tried in the Bias +/- Tolerance range, bounds included.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetScaleXCount() const
void SetScaleXCount(OEV_UINT32 value)
```

# EOCVText::GetScaleXDeviation

# EOCVText::SetScaleXDeviation

Scale X standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScaleXDeviation() const
void SetScaleXDeviation(float value)
```

# EOCVText::GetScaleXMax

# EOCVText::SetScaleXMax

Maximum Y-scale.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScaleXMax() const
void SetScaleXMax(float value)
```

# EOCVText::GetScaleXMin

# EOCVText::SetScaleXMin

Minimum X-scale.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScaleXMin() const
void SetScaleXMin(float value)
```

# EOCVText::GetScaleXTolerance

# EOCVText::SetScaleXTolerance

Horizontal scaling tolerance.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScaleXTolerance() const
void SetScaleXTolerance(float value)
```

# EOCVText::GetScaleY

# EOCVText::SetScaleY

Measured vertical scaling, expressed as a dimensionless ratio.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScaleY() const
void SetScaleY(float value)
```

# EOCVText::GetScaleYAverage

# EOCVText::SetScaleYAverage

Scale Y average.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScaleYAverage() const
void SetScaleYAverage(float value)
```

# EOCVText::GetScaleYBias

# EOCVText::SetScaleYBias

Vertical scaling bias.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScaleYBias() const
void SetScaleYBias(float value)
```

## Remarks

**0** corresponds to the nominal, true scale factor.

# EOCVText::GetScaleYCount

# EOCVText::SetScaleYCount

Number of values to be tried in the Bias +/- Tolerance range, bounds included.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetScaleYCount() const
void SetScaleYCount(OEV_UINT32 value)
```

# EOCVText::GetScaleYDeviation

# EOCVText::SetScaleYDeviation

Scale Y standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScaleYDeviation() const
void SetScaleYDeviation(float value)
```

# EOCVText::GetScaleYMax

# EOCVText::SetScaleYMax

Maximum Y-scale.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetScaleYMax() const
void SetScaleYMax(float value)
```

# EOCVText::GetScaleYMin

# EOCVText::SetScaleYMin

Minimum Y-scale.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetScaleYMin() const
void SetScaleYMin(float value)
```

# EOCVText::GetScaleYTolerance

# EOCVText::SetScaleYTolerance

Vertical scaling tolerance.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetScaleYTolerance() const
void SetScaleYTolerance(float value)
```

# EOCVText::GetSelected

# EOCVText::SetSelected

Selection state. **TRUE** when selected.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetSelected()
void SetSelected(BOOL selected)
```

# EOCVText::GetShear

# EOCVText::SetShear

Measured shearing, clockwise from the vertical direction, expressed in the current angle unit.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShear() const
void SetShear(float value)
```

# EOCVText::GetShearAverage

# EOCVText::SetShearAverage

Shear average.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShearAverage() const
void SetShearAverage(float value)
```

# EOCVText::GetShearBias

# EOCVText::SetShearBias

Shearing bias.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShearBias() const
void SetShearBias(float value)
```

### Remarks

**0** corresponds to the nominal, upright position.

# EOCVText::GetShearCount

# EOCVText::SetShearCount

Number of values to be tried in the Bias +/- Tolerance range, bounds included.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetShearCount() const
void SetShearCount(OEV_UINT32 value)
```

# EOCVText::GetShearDeviation

# EOCVText::SetShearDeviation

Shear standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShearDeviation() const
void SetShearDeviation(float value)
```

# EOCVText::GetShearMax

# EOCVText::SetShearMax

Maximum shear.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShearMax() const
void SetShearMax(float value)
```

# EOCVText::GetShearMin

# EOCVText::SetShearMin

Minimum shear.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShearMin() const
void SetShearMin(float value)
```

# EOCVText::GetShearTolerance

# EOCVText::SetShearTolerance

Shearing tolerance.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShearTolerance() const
void SetShearTolerance(float value)
```

# EOCVText::GetShiftX

# EOCVText::SetShiftX

Measured horizontal translation, in pixels.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetShiftX() const
void SetShiftX(float value)
```

# EOCVText::GetShiftXAverage

# EOCVText::SetShiftXAverage

Shift X average.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetShiftXAverage() const
void SetShiftXAverage(float value)
```

# EOCVText::GetShiftXBias

# EOCVText::SetShiftXBias

Horizontal translation bias.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShiftXBias() const
void SetShiftXBias(float value)
```

## Remarks

**0** corresponds to the nominal position.

# EOCVText::GetShiftXDeviation

# EOCVText::SetShiftXDeviation

Shift X standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShiftXDeviation() const
void SetShiftXDeviation(float value)
```

# EOCVText::GetShiftXMax

# EOCVText::SetShiftXMax

Maximum Shift Y.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetShiftXMax() const
void SetShiftXMax(float value)
```

# EOCVText::GetShiftXMin

# EOCVText::SetShiftXMin

Minimum Shift X.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetShiftXMin() const
void SetShiftXMin(float value)
```

# EOCVText::GetShiftXStride

# EOCVText::SetShiftXStride

First pass stride for the horizontal translation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetShiftXStride() const
void SetShiftXStride(OEV_UINT32 value)
```

# EOCVText::GetShiftXTolerance

# EOCVText::SetShiftXTolerance

Horizontal translation tolerance.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShiftXTolerance() const
void SetShiftXTolerance(float value)
```

# EOCVText::GetShiftY

# EOCVText::SetShiftY

Measured vertical translation, in pixels.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShiftY() const
void SetShiftY(float value)
```

# EOCVText::GetShiftYAverage

# EOCVText::SetShiftYAverage

Shift Y average.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShiftYAverage() const
void SetShiftYAverage(float value)
```

# EOCVText::GetShiftYBias

# EOCVText::SetShiftYBias

Vertical translation bias.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShiftYBias() const
void SetShiftYBias(float value)
```

## Remarks

**0** corresponds to the nominal position.

# EOCVText::GetShiftYDeviation

# EOCVText::SetShiftYDeviation

Shift Y standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShiftYDeviation() const
void SetShiftYDeviation(float value)
```

# EOCVText::GetShiftYMax

# EOCVText::SetShiftYMax

Maximum Shift Y.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetShiftYMax() const
void SetShiftYMax(float value)
```

# EOCVText::GetShiftYMin

# EOCVText::SetShiftYMin

Minimum Shift Y.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetShiftYMin() const
void SetShiftYMin(float value)
```

# EOCVText::GetShiftYStride

# EOCVText::SetShiftYStride

First pass stride for the vertical translation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetShiftYStride() const
void SetShiftYStride(OEV_UINT32 value)
```

# EOCVText::GetShiftYTolerance

# EOCVText::SetShiftYTolerance

Vertical translation tolerance.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetShiftYTolerance() const
void SetShiftYTolerance(float value)
```

# EOCVText::GetSkew

# EOCVText::SetSkew

Measured rotation, clockwise from the horizontal direction, expressed in the current angle unit.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSkew() const
void SetSkew(float value)
```

# EOCVText::GetSkewAverage

# EOCVText::SetSkewAverage

Skew average.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSkewAverage() const
void SetSkewAverage(float value)
```

# EOCVText::GetSkewBias

# EOCVText::SetSkewBias

Rotation bias.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSkewBias() const
void SetSkewBias(float value)
```

## Remarks

**0** corresponds to the nominal, horizontal position.

# EOCVText::GetSkewCount

# EOCVText::SetSkewCount

Number of values to be tried in the Bias +/- Tolerance range, bounds included.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetSkewCount() const
void SetSkewCount(OEV_UINT32 value)
```

# EOCVText::GetSkewDeviation

# EOCVText::SetSkewDeviation

Skew standard deviation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSkewDeviation() const
void SetSkewDeviation(float value)
```

# EOCVText::GetSkewMax

# EOCVText::SetSkewMax

Maximum skew.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSkewMax() const
void SetSkewMax(float value)
```

# EOCVText::GetSkewMin

# EOCVText::SetSkewMin

Minimum skew.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSkewMin() const
void SetSkewMin(float value)
```

# EOCVText::GetSkewTolerance

# EOCVText::SetSkewTolerance

Rotation tolerance.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSkewTolerance() const
void SetSkewTolerance(float value)
```

# EOCVText::GetTemplateBackgroundArea

# EOCVText::SetTemplateBackgroundArea

Number of pixels in the background of the characters of this text.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetTemplateBackgroundArea() const
void SetTemplateBackgroundArea(OEV_UINT32 value)
```

# EOCVText::GetTemplateBackgroundSum

# EOCVText::SetTemplateBackgroundSum

Sum of the normalized gray-level values of the background pixels of the characters of this text.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetTemplateBackgroundSum() const
void SetTemplateBackgroundSum(float value)
```

# EOCVText::GetTemplateForegroundArea

# EOCVText::SetTemplateForegroundArea

Number of pixels in the foreground of all characters of this text.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetTemplateForegroundArea() const
void SetTemplateForegroundArea(OEV_UINT32 value)
```

# EOCVText::GetTemplateForegroundSum

# EOCVText::SetTemplateForegroundSum

Sum of the normalized gray-level values of the foreground pixels of the characters of this text.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetTemplateForegroundSum() const
void SetTemplateForegroundSum(float value)
```

# EOCVText::GetTemplateLocationScore

# EOCVText::SetTemplateLocationScore

Value of the location score measured on the template during learning. If necessary, this value may be set explicitly.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetTemplateLocationScore() const
void SetTemplateLocationScore(float value)
```

# 4.136. EPathVector Class

Vector objects are used to store 1-dimensional data.

### Remarks

Using vectors is very similar to using 1-dimensional arrays, except that the size can vary at runtime. Memory allocation is handled internally. * To create a vector, use its constructor. * To fill a vector with values, first empty it, using the EPathVector member, and then add elements one at time at the tail by calling the EPathVector::AddElement member. * To access a vector element, either for reading or writing, use the [] operator. * To inquire for the current number of elements, use member EPathVector.

**Base Class:** EVector

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddElement | Appends (adds at the tail) an element to the vector. |

| | |
|---|---|
| Draw | Draws a plot of the vector element values. |
| DrawWithCurrentPen | Draws a plot of the vector element values. |
| EPathVector | Constructs a vector. |
| GetClosed | - |
| GetElement | Returns the vector element at the given index. |
| GetRawDataPtr | Pointer to the vector data. |
| operator[] | Gives access to the vector element at the given index. |
| operator= | Copies all the data from another EPathVector object into the current EPathVector object |
| SetClosed | - |
| SetElement | Modifies the vector element at the given index by the given value. |

# EPathVector::AddElement

Appends (adds at the tail) an element to the vector.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void AddElement(
 EPath element
 )
```

### Parameters

*element*

The element to be added.

# EPathVector::GetClosed

# EPathVector::SetClosed

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetClosed()

void SetClosed(BOOL bClosed)
```

# EPathVector::Draw

Draws a plot of the vector element values.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )

void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )

void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*zoomX*

Zooming factor along the X axis (**1.0f** means no zoom).

*zoomY*

Zooming factor along the Y axis (**1.0f** means no zoom).

*originX*

Abscissa of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*originY*

Ordinate of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*color*

The color in which to draw the overlay.

### Remarks

The vector draws line segment between the element coordinates. The drawing appears on the image itself. Drawing is done in the device context associated to the desired window. To draw the curves, the current pen is used.

# EPathVector::DrawWithCurrentPen

Draws a plot of the vector element values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*zoomX*

Zooming factor along the X axis (**1.0f** means no zoom).

*zoomY*

Zooming factor along the Y axis (**1.0f** means no zoom).

*originX*

Abscissa of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

*originY*

Ordinate of the upper left corner of the plot's bounding rectangle, in pixels. By default, the upper left corner of the window is used.

### Remarks

The vector draws line segment between the element coordinates. The drawing appears on the image itself. Drawing is done in the device context associated to the desired window. To draw the curves, the current pen is used.

# EPathVector::EPathVector

Constructs a vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EPathVector(
 )
void EPathVector(
 OEV_UINT32 maxNumberOfElements
 )
void EPathVector(
 const EPathVector& other
 )
```

### Parameters

*maxNumberOfElements*

Optionally, memory can be pre-allocated to accommodate a given number of elements.

*other*

EPathVector object to be copied

# EPathVector::GetElement

Returns the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPath GetElement(
 OEV_INT32 index
 )
```

### Parameters

*index*

Index, between **0** and EPathVector (excluded) of the element to be accessed.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# EPathVector::operator[]

Gives access to the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPath& operator[](
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index, between **0** and EPathVector (excluded) of the element to be accessed.

# EPathVector::operator=

Copies all the data from another EPathVector object into the current EPathVector object

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPathVector& operator=(
 const EPathVector& other
 )
```

### Parameters

*other*

EPathVector object to be copied

# EPathVector::GetRawDataPtr

Pointer to the vector data.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void* GetRawDataPtr() const
```

# EPathVector::SetElement

Modifies the vector element at the given index by the given value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetElement(
 OEV_INT32 index,
 EPath value
 )
```

### Parameters

*index*

Index, between **0** and EPathVector (excluded), of the element to be modified.

*value*

The new value for the element.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# 4.137. EPatternFinder Class

Manages a complete finding context in EasyFind.

**Base Class:** EPointShape

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| CopyLearntPattern | Copies the learnt pattern in the supplied image. If no pattern has been learned, an exception with code EError_NoPatternLearnt will be thrown. |
| DrawModel | Draws the model features with an overlay in image coordinates. |

| | |
|---|---|
| DrawModelWithCurrentPen | Draws the model features with an overlay in image coordinates. |
| EPatternFinder | Constructs a EPatternFinder context. |
| Find | Locates a set of possible occurrences of the learnt pattern in the supplied search field. |
| GetAngleBias | Angle bias, expressed in the current angle unit. |
| GetAngleSearchExtent | The angular extension of the search neighborhood. See the EPatternFinder::LocalSearchMode property description for further details. |
| GetAngleTolerance | Angle tolerance, expressed in the current angle unit. |
| GetAutoTransitionThickness | Indicates whether the EPatternFinder::TransitionThickness property is automatically computed or not. |
| GetContrastMode | Contrast of the instance, as defined in EFindContrastMode. |
| GetFindExtension | Extension value, that is the pattern margin size (in pixels) that is allowed to go out of the search field. |
| GetForcedThreshold | Forced threshold, between [0, 255]. |

| | |
|---|---|
| GetInterpolate | Whether interpolation is performed when searching for a pattern occurrence. |
| GetLearningDone | Indicates whether a pattern has already been learnt. |
| GetLightBalance | Light balance, between **[-1.0, 1.0]**. |
| GetLocalSearchMode | Sets the local search mode. |
| GetMaxFeaturePoints | Maximum number of feature points at the fine stage. |
| GetMaxInstances | Maximum number of instances to be found. |
| GetMinFeaturePoints | Minimum number of feature points at the coarse stage. |
| GetMinScore | Minimum score of found instances, between **[-1.0, 1.0]**. |
| GetPatternType | Pattern type, as defined in EPatternType. |
| GetPivot | Reference point in the model. |
| GetReductionMode | The reduction mode that is to be used when learning the model (automatic or manual), as defined in EReductionMode. |

| | |
|---|---|
| GetReductionStrength | The reduction strength that is to be used when learning the model, between **0** and **1**. |
| GetScaleBias | Scale bias, expressed in units (not in percent). |
| GetScaleSearchExtent | The scaling extension of the search neighborhood. See the EPatternFinder::LocalSearchMode property description for further details. |
| GetScaleTolerance | Scale tolerance, expressed in units (not in percent). |
| GetThinStructureMode | Mode for EPatternType_ThinStructure, as defined in EThinStructureMode. |
| GetTransitionThickness | Transition thickness, expressed in pixels. |
| GetType | Shape type. |
| GetXSearchExtent | The X-axis extension of the search neighborhood. See the EPatternFinder::LocalSearchMode property description for further details. |
| GetYSearchExtent | The Y-axis extension of the search neighborhood. See the EPatternFinder::LocalSearchMode property description for further details. |
| Learn | Learns a given pattern and stores it in the EPatternFinder object. |

| | |
|---|---|
| operator= | Copies all the data from another EPatternFinder object into the current EPatternFinder object |
| SetAngleBias | Angle bias, expressed in the current angle unit. |
| SetAngleSearchExtent | The angular extension of the search neighborhood. See the EPatternFinder::LocalSearchMode property description for further details. |
| SetAngleTolerance | Angle tolerance, expressed in the current angle unit. |
| SetAutoTransitionThickness | Indicates whether the EPatternFinder::TransitionThickness property is automatically computed or not. |
| SetContrastMode | Contrast of the instance, as defined in EFindContrastMode. |
| SetFindExtension | Extension value, that is the pattern margin size (in pixels) that is allowed to go out of the search field. |
| SetForcedThreshold | Forced threshold, between [0, 255]. |
| SetInterpolate | Whether interpolation is performed when searching for a pattern occurrence. |
| SetLightBalance | Light balance, between **[-1.0, 1.0]**. |

| | |
|---|---|
| SetLocalSearchMode | Sets the local search mode. |
| SetMaxFeaturePoints | Maximum number of feature points at the fine stage. |
| SetMaxInstances | Maximum number of instances to be found. |
| SetMinFeaturePoints | Minimum number of feature points at the coarse stage. |
| SetMinScore | Minimum score of found instances, between **[-1.0, 1.0]**. |
| SetPatternType | Pattern type, as defined in EPatternType. |
| SetPivot | Reference point in the model. |
| SetReductionMode | The reduction mode that is to be used when learning the model (automatic or manual), as defined in EReductionMode. |
| SetReductionStrength | The reduction strength that is to be used when learning the model, between **0** and **1**. |
| SetScaleBias | Scale bias, expressed in units (not in percent). |
| SetScaleSearchExtent | The scaling extension of the search neighborhood. See the EPatternFinder::LocalSearchMode property description for further details. |

| | |
|---|---|
| SetScaleTolerance | Scale tolerance, expressed in units (not in percent). |
| SetThinStructureMode | Mode for EPatternType_ThinStructure, as defined in EThinStructureMode. |
| SetTransitionThickness | Transition thickness, expressed in pixels. |
| SetXSearchExtent | The X-axis extension of the search neighborhood. See the EPatternFinder::LocalSearchMode property description for further details. |
| SetYSearchExtent | The Y-axis extension of the search neighborhood. See the EPatternFinder::LocalSearchMode property description for further details. |

# EPatternFinder::GetAngleBias

# EPatternFinder::SetAngleBias

Angle bias, expressed in the current angle unit.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetAngleBias() const
```

```
void SetAngleBias(float f32AngleBias)
```

### Remarks

The **AngleBias** defines the angle offset between the model and the instances. Finding the pattern is performed in range **AngleBias** +/- EPatternFinder::AngleTolerance. This range should not exceed a full turn. Default: **0.0**.

# EPatternFinder::GetAngleSearchExtent

# EPatternFinder::SetAngleSearchExtent

The angular extension of the search neighborhood. See the EPatternFinder::LocalSearchMode property description for further details.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetAngleSearchExtent() const
void SetAngleSearchExtent(OEV_INT32 n32Extent)
```

# EPatternFinder::GetAngleTolerance

# EPatternFinder::SetAngleTolerance

Angle tolerance, expressed in the current angle unit.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetAngleTolerance() const
void SetAngleTolerance(float f32AngleTolerance)
```

### Remarks

The **AngleTolerance** defines the angle allowance of the instances around the EPatternFinder::AngleBias. Finding the pattern is performed in range EPatternFinder::AngleBias +/- **AngleTolerance**. This range should not exceed a full turn. A **NULL** tolerance can be set, in which case the angle bias value is assumed. Default: **0.0**.

# EPatternFinder::GetAutoTransitionThickness

# EPatternFinder::SetAutoTransitionThickness

Indicates whether the EPatternFinder::TransitionThickness property is automatically computed or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool GetAutoTransitionThickness() const
void SetAutoTransitionThickness(bool bAutoTransitionThickness)
```

### Remarks

If set to **TRUE**, EPatternFinder::TransitionThickness is automatically computed during the EPatternFinder::Learn method call. This computed value will be used (by EPatternFinder::Find) until a new EPatternFinder::Learn is done. Even if the user explicitly sets the EPatternFinder::TransitionThickness, it will have no effect, as EPatternFinder::Find will use the computed value. If set to **FALSE**, EPatternFinder::TransitionThickness is set by the user. It is never automatically changed by a EPatternFinder::Learn method call. Default: **TRUE**.

# EPatternFinder::GetContrastMode

# EPatternFinder::SetContrastMode

Contrast of the instance, as defined in EFindContrastMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EFindContrastMode GetContrastMode() const
void SetContrastMode(Euresys::Open_eVision_2_10::EFindContrastMode eContrastMode)
```

### Remarks

This is a EPatternType_ConsistentEdges pattern type property. It defines the contrast of regions. Contrast can be normal (as in the model), inverse (inverse contrast of the model), or any (same or inverse contrast of the model). Default: EFindContrastMode_Normal.

# EPatternFinder::CopyLearntPattern

Copies the learnt pattern in the supplied image. If no pattern has been learned, an exception with code NoPatternLearnt will be thrown.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void CopyLearntPattern(
 EImageBW8& image
 )
```

### Parameters

*image*

Pointer to the image in which the learnt pattern will be returned.

# EPatternFinder::DrawModel

Draws the model features with an overlay in image coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawModel(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
void DrawModel(
 HDC graphicContext,
 const ERGBColor& color,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
void DrawModel(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*graphicContext*

Handle to the device context of the destination windows.

*zoomX*

Horizontal zooming factor.

*zoomY*

Vertical zooming factor. If set to **0**, the horizontal zooming factor will be used for isotropic zooming.

*panX*

Horizontal panning offset.

*panY*

Vertical panning offset.

*color*

The color in which to draw the overlay.

# EPatternFinder::DrawModelWithCurrentPen

Draws the model features with an overlay in image coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawModelWithCurrentPen(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination windows.

*zoomX*

Horizontal zooming factor.

*zoomY*

Vertical zooming factor. If set to **0**, the horizontal zooming factor will be used for isotropic zooming.

*panX*

Horizontal panning offset.

*panY*

Vertical panning offset.

# EPatternFinder::EPatternFinder

Constructs a EPatternFinder context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EPatternFinder(
 )
void EPatternFinder(
 const EPatternFinder& other
 )
```

### Parameters

*other*

Another EPatternFinder object to be copied in the new EPatternFinder object.

### Remarks

All properties are initialized to their respective default values.

# EPatternFinder::Find

Locates a set of possible occurrences of the learnt pattern in the supplied search field.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::vector<Euresys::Open_eVision_2_10::EFoundPattern> Find(
 EROIBW8* source
 )
```

```
std::vector<Euresys::Open_eVision_2_10::EFoundPattern> Find(
 EROIBW8* source,
 const ERegion& region
 )
```

### Parameters

*source*

Image or part of an image in which the learnt model has to be searched for.

*region*

Region into the ROI where the search is performed.

### Remarks

This method will fail if no pattern has been learnt previously. The result is a vector of EFoundPattern objects. EFoundPattern instances can be retrieved by using the **Vector<>::operator[]** method.

# EPatternFinder::GetFindExtension

# EPatternFinder::SetFindExtension

Extension value, that is the pattern margin size (in pixels) that is allowed to go out of the search field.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetFindExtension() const
void SetFindExtension(OEV_INT32 n32Extension)
```

### Remarks

When a non-**NULL** value is attributed to the extension, the detection of instances partially out of the ROI is allowed. The extension value defines how much the ROI is extended. Default: **0**.

# EPatternFinder::GetForcedThreshold

# EPatternFinder::SetForcedThreshold

Forced threshold, between [0, 255].

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetForcedThreshold() const
void SetForcedThreshold(OEV_UINT32 un32ForcedThreshold)
```

## Remarks

This property fixes an absolute gray-level threshold to help the EPatternFinder in the extraction of regions in the EPatternType_ContrastingRegions. Default: **0**, which means that the thresholding is computed automatically. Once this property has been changed, a new learning process has to be done, to take the new value into account. Note that this property will remain to its value even after a new learning process. An efficient way to see the effect of changing this property is to use the EPatternFinder::DrawModel method.

# EPatternFinder::GetInterpolate

# EPatternFinder::SetInterpolate

Whether interpolation is performed when searching for a pattern occurrence.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
BOOL GetInterpolate() const

void SetInterpolate(BOOL bInterpolate)
```

### Remarks

By default, matching is done with a one-pixel precision for all degrees of freedom (translation, rotation and scaling). You can use an additional interpolation process to achieve sub-pixel accuracy. This generally leads to an improvement of the sub-pixel accuracy by a factor larger than 10. This is possible only when the found instances match closely the model. A score higher than 0.99 indicates that the instances are a close match of the model. In other words, the instance is considered to be more accurate when the score is higher. The added computational cost is low. Default: **TRUE**.

# EPatternFinder::Learn

Learns a given pattern and stores it in the EPatternFinder object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Learn(
 const EROIBW8* pattern,
 EROIBW8* dontCare
 )

void Learn(
 const EROIBW8* pattern,
 const ERegion& region
 )
```

### Parameters

*pattern*

Model to be learnt (ROI).

*dontCare*

"Don't care" area mask (ROI).

*region*

Region into the ROI where the learning is performed

### Remarks

Learning another pattern erases the information stored for the first one. A "don't care area" can be set as argument, allowing to mask and not take into account certain parts of the pattern while learning. The "don't care area" mask should have the same size as the model. The mask should be a bilevel image with pixel - values of '0' for ignored areas and '255' otherwise.

# EPatternFinder::GetLearningDone

Indicates whether a pattern has already been learnt.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
bool GetLearningDone() const
```

# EPatternFinder::GetLightBalance

# EPatternFinder::SetLightBalance

Light balance, between **[-1.0, 1.0]**.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetLightBalance() const
void SetLightBalance(float f32LightBalance)
```

### Remarks

Consistent Edges and Thin StructuresIn the EPatternType_ConsistentEdges and EPatternType_ThinStructure modes, the **LightBalance** property governs the selection of the feature points while learning the model. It drives which edge points are eligible as feature points in the model, by defining a criterion for ignoring those edge points that are not sharp enough. As a consequence, this property will influence the spatial distribution of the feature points. In the aforementioned operating modes, the feature points are the places in the image that exhibit a strong variation in the gray level signal. Mathematically, these places are those at which the magnitude of the gradient is significant. The **LightBalance** property defines the way the latter threshold on the magnitude of the gradient is computed, through a careful analysis of the dynamics of gradient. The more the **LightBalance** tends to -1, the more tolerant will be the threshold, and the more edge points will be considered as candidates for becoming feature points. Conversely, as the **LightBalance** property becomes close to 1, only the points with a high gradient magnitude are taken into consideration. In other words, a small **LightBalance** defines a loose criterion for defining what an edge point is, whereas a great value implies a conservative criterion. By default, this property is fixed to **0.0**. This is an appropriate value for most images which are encountered in industrial machine vision. Contrasting RegionsWhen using the EPatternType_ContrastingRegions pattern type, this property allows the user to compensate for poor lighting conditions of the model. The more the **LightBalance** tends to 1, the lower the threshold will be on the model, and the more dark regions will be considered. Conversely, as the **LightBalance** parameter becomes close to -1, only the bright regions are taken into consideration. The **LightBalance** is automatically set to **0.0** after a learning process. Once the **LightBalance** is changed, a new learning process has to be done to take the new value into account. An efficient way to see the effect of changing this property is to use the EPatternFinder::DrawModel method.

# EPatternFinder::GetLocalSearchMode

# EPatternFinder::SetLocalSearchMode

Sets the local search mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::ELocalSearchMode GetLocalSearchMode() const
void SetLocalSearchMode(Euresys::Open_eVision_2_10::ELocalSearchMode eLocalSearchMode)
```

### Remarks

In the multi-stage approach of EasyFind, pattern occurrence candidates are at first found at the coarsest stage. Then, at each of the following stages, their position and score are refined until the last and finest one. This refining is achieved by searching for better candidates in the neighborhood of each of the ones found in the previous stage. The local search mode allows the user to set the extent of this neighborhood. By default, the local search mode is set to ELocalSearchMode_Basic.

# EPatternFinder::GetMaxFeaturePoints

# EPatternFinder::SetMaxFeaturePoints

Maximum number of feature points at the fine stage.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetMaxFeaturePoints() const
void SetMaxFeaturePoints(OEV_UINT32 un32MaxFeaturePoints)
```

### Remarks

Default: **1024**. Reserved use.

# EPatternFinder::GetMaxInstances

# EPatternFinder::SetMaxInstances

Maximum number of instances to be found.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetMaxInstances() const
void SetMaxInstances(OEV_UINT32 un32MaxInstances)
```

### Remarks

Default: **1**.

# EPatternFinder::GetMinFeaturePoints

# EPatternFinder::SetMinFeaturePoints

Minimum number of feature points at the coarse stage.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetMinFeaturePoints() const
void SetMinFeaturePoints(OEV_UINT32 un32MinFeaturePoints)
```

### Remarks

Default: **8**. Reserved use.

# EPatternFinder::GetMinScore

# EPatternFinder::SetMinScore

Minimum score of found instances, between **[-1.0, 1.0]**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetMinScore() const
void SetMinScore(float f32MinScore)
```

### Remarks

Instances with a score under the **MinScore** will not be returned.

# EPatternFinder::operator=

Copies all the data from another EPatternFinder object into the current EPatternFinder object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPatternFinder& operator=(
 const EPatternFinder& other
 )
```

### Parameters

*other*

EPatternFinder object to be copied

# EPatternFinder::GetPatternType

# EPatternFinder::SetPatternType

Pattern type, as defined in EPatternType.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EPatternType GetPatternType() const
void SetPatternType(Euresys::Open_eVision_2_10::EPatternType patternType)
```

### Remarks

This property informs the EPatternFinder of the general nature of the model to be learnt. Default: EPatternType_ConsistentEdges.

# EPatternFinder::GetPivot

# EPatternFinder::SetPivot

Reference point in the model.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetPivot() const
void SetPivot(const EPoint& pivot)
```

### Remarks

The coordinates of the reference point are relative to the upper left corner of the model. The location of an instance (Coordinates (X,Y)) is the location of its reference point defined in the model. By default, the pivot is a EPoint set to the pattern center. EPoint is a structure that contains two x and y float values.

# EPatternFinder::GetReductionMode

# EPatternFinder::SetReductionMode

The reduction mode that is to be used when learning the model (automatic or manual), as defined in EReductionMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EReductionMode GetReductionMode() const
void SetReductionMode(Euresys::Open_eVision_2_10::EReductionMode eReductionMode)
```

## Remarks

Specifies whether the best-guess method should be used to assert the level of reduction that will be used when learning the model. If this property is set to EReductionMode_Manual, it is up to the user to provide a suitable reduction strength. This value is only used when learning the model. Default: EReductionMode_Auto, which means that the best-guess algorithm is used by default.

# EPatternFinder::GetReductionStrength

# EPatternFinder::SetReductionStrength

The reduction strength that is to be used when learning the model, between **0** and **1**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
float GetReductionStrength() const

void SetReductionStrength(float f32ReductionStrength)
```

### Remarks

Specifies the reduction strength for learning the model (encoded as a percentage). Its precise semantics depends on the reduction mode (see the EPatternFinder::ReductionMode property): * In the automatic reduction mode, its value is undefined until a model is learned. When a model is learned (i.e. after a call to EPatternFinder::Learn), the value of this property can be read, in which case it reflects the reduction strength that has been automatically chosen by the best-guess algorithm. * In the manual reduction mode, this property must be set by the user and is kept constant throughout the entire lifetime of the object. The new value of the property is only used at the following call to EPatternFinder::Learn. This value only has an effect when learning the model. Default: The default value depends on the value of the EPatternFinder::ReductionMode property. Allowed values: Floating-point number in the interval **[0..1]**.

# EPatternFinder::GetScaleBias

# EPatternFinder::SetScaleBias

Scale bias, expressed in units (not in percent).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScaleBias() const

void SetScaleBias(float f32ScaleBias)
```

### Remarks

The **ScaleBias** defines the scale factor between the model and the instances. Finding the pattern is performed in range **ScaleBias** +/- **ScaleTolerance**. This range should not exceed **[0.5..2.5]** (50 % to 250 % scaling). Default: **1.0** (100 %).

# EPatternFinder::GetScaleSearchExtent

# EPatternFinder::SetScaleSearchExtent

The scaling extension of the search neighborhood. See the EPatternFinder::LocalSearchMode property description for further details.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetScaleSearchExtent() const
void SetScaleSearchExtent(OEV_INT32 n32Extent)
```

# EPatternFinder::GetScaleTolerance

# EPatternFinder::SetScaleTolerance

Scale tolerance, expressed in units (not in percent).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScaleTolerance() const
void SetScaleTolerance(float f32ScaleTolerance)
```

## Remarks

The **ScaleTolerance** defines the scale allowance of the instances around the EPatternFinder::ScaleBias. Finding the pattern is performed in range EPatternFinder::ScaleBias +/- **ScaleTolerance**. This range should not exceed **[0.5..2]** (50 % to 200 % scaling). A **NULL** tolerance can be set, in which case the scale bias value is assumed. Default: **0.0**.

# EPatternFinder::GetThinStructureMode

# EPatternFinder::SetThinStructureMode

Mode for ThinStructure, as defined in EThinStructureMode.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::EThinStructureMode GetThinStructureMode() const

void SetThinStructureMode(Euresys::Open_eVision_2_10::EThinStructureMode
thinStructureMode)
```

## Remarks

EThinStructureMode informs EasyFind if thin elements in the model are darker or brighter than regions. Default: EThinStructureMode_Auto, which detects the best mode between dark or bright.

# EPatternFinder::GetTransitionThickness

# EPatternFinder::SetTransitionThickness

Transition thickness, expressed in pixels.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetTransitionThickness() const

void SetTransitionThickness(OEV_UINT32 un32TransitionThickness)
```

### Remarks

This property defines the tolerance on the location of transitions between regions in EPatternType_ContrastingRegions. An efficient way to see the effect of changing this property is to use the EPatternFinder::DrawModel method. The **TransitionThickness** value, used by EPatternFinder::Find, is either the automatically computed value (by a EPatternFinder::Learn method call) if EPatternFinder::AutoTransitionThickness is **TRUE**, or the user-defined value if EPatternFinder::AutoTransitionThickness is **FALSE**.

**Note.** If EPatternFinder::AutoTransitionThickness is **TRUE** and no EPatternFinder::Learn has been executed, the **TransitionThickness** value that EPatternFinder::Find will use is undefined, even if it has been manually set in the meantime.

# EPatternFinder::GetType

Shape type.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::EShapeType GetType()
```

# EPatternFinder::GetXSearchExtent

# EPatternFinder::SetXSearchExtent

The X-axis extension of the search neighborhood. See the EPatternFinder::LocalSearchMode property description for further details.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 GetXSearchExtent() const
void SetXSearchExtent(OEV_INT32 n32Extent)
```

## EPatternFinder::GetYSearchExtent

## EPatternFinder::SetYSearchExtent

The Y-axis extension of the search neighborhood. See the EPatternFinder::LocalSearchMode property description for further details.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetYSearchExtent() const
void SetYSearchExtent(OEV_INT32 n32Extent)
```

# 4.138. EPeakVector Class

Represents a vector of profile peaks.

**Base Class:** EVector
**Namespace:** Euresys::Open_eVision_2_10

### Methods

| | |
|---|---|
| AddElement | Appends (adds at the tail) an element to the vector. |

| | |
|---|---|
| EPeakVector | Constructs a vector. |
| GetElement | Returns the vector element at the given index. |
| GetRawDataPtr | Pointer to the vector data. |
| operator[] | Gives access to the vector element at the given index. |
| operator= | Copies all the data from another EPeakVector object into the current EPeakVector object |
| SetElement | Modifies the vector element at the given index by the given value. |

E

P

# eakVector::AddElement

Appends (adds at the tail) an element to the vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddElement(
 EPeak element
 )
```

## Parameters

*element*

The element to be added.

# EPeakVector::EPeakVector

Constructs a vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EPeakVector(
 )
void EPeakVector(
 const EPeakVector& other
 )
void EPeakVector(
 OEV_UINT32 maxNumberOfElements
 )
```

### Parameters

*other*

EPeakVector object to be copied

*maxNumberOfElements*

Optionally, memory can be pre-allocated to accommodate a given number of elements.

# EPeakVector::GetElement

Returns the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPeak GetElement(
 OEV_INT32 index
 )
```

### Parameters

*index*

Index, between **0** and EPeakVector (excluded) of the element to be accessed.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# EPeakVector::operator[]

Gives access to the vector element at the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPeak& operator[](
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index, between **0** and EPeakVector (excluded) of the element to be accessed.

# EPeakVector::operator=

Copies all the data from another EPeakVector object into the current EPeakVector object

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPeakVector& operator=(
 const EPeakVector& other
 )
```

### Parameters

*other*

EPeakVector object to be copied

# EPeakVector::GetRawDataPtr

Pointer to the vector data.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void* GetRawDataPtr() const
```

# EPeakVector::SetElement

Modifies the vector element at the given index by the given value.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetElement(
 OEV_INT32 index,
 EPeak value
 )
```

### Parameters

*index*

Index, between **0** and EPeakVector (excluded), of the element to be modified.

*value*

The new value for the element.

### Remarks

If the given index is outside the bounds of the vector, the error code EError_Parameter1OutOfRange is set.

# 4.139. EPlaneCropper Class

A EPlaneCropper object is used to crop some points of a EPointCloud object.
The points to keep are selected according to their positions with respect to a reference plane.
A EPlaneCropper object is characterized by its reference plane and is used to produce an output EPointCloud object from an input EPointCloud object.
The produced point cloud contains a subset of the input point cloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| Crop | Crops an EPointCloud. An output point cloud is produced from an input point cloud by only keeping the points that satisfy the specified condition.<br>The condition tests the (signed) distance of the points with respect to the reference plane of the cropper. |

| | |
|---|---|
| EPlaneCropper | Creates an EPlaneCropper object using a horizontal plane as a default reference. |
| GetPlane | Sets/gets the new reference E3DPlane of the EPlaneCropper object. |
| Load | Loads the EPlaneCropper configuration. The given ESerializer must have been created for reading. |
| operator= | Assignment operator. |
| Save | Saves the EPlaneCropper configuration. The given ESerializer must have been created for writing. |
| SetPlane | Sets/gets the new reference E3DPlane of the EPlaneCropper object. |

# PlaneCropper::Crop

Crops an EPointCloud. An output point cloud is produced from an input point cloud by only keeping the points that satisfy the specified condition.
The condition tests the (signed) distance of the points with respect to the reference plane of the cropper.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Crop(
 const EPointCloud& cloudIn,
 EPointCloud& cloudOut,
 Euresys::Open_eVision_2_10::Easy3D::EPlaneCropperType type,
 float maxDistance
 )
```

### Parameters

*cloudIn*

The input point cloud.

*cloudOut*

The output point cloud.

*type*

An enum of type EPlaneCropperType that specifies which points of the input point cloud will be copied to the ouput point cloud.

*maxDistance*

Specifies the distance from the plane for the types "EPlaneCropperType_KeepClose" and "EPlaneCropperType_KeepFar".
It should be **0** for the types "EPlaneCropperType_KeepAbove" and "EPlaneCropperType_KeepBelow.

### Remarks

The input point cloud 'cloudIn' should be different from the output point cloud 'cloudOut'. If not a exception will be thrown.

# EPlaneCropper::EPlaneCropper

Creates an EPlaneCropper object using a horizontal plane as a default reference.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EPlaneCropper(
 )
```

```
void EPlaneCropper(
 const E3DPlane& plane
 )
void EPlaneCropper(
 const EPlaneCropper& other
 )
```

### Parameters

*plane*

Reference E3DPlane used for the initialization.

*other*

Reference EPlaneCropper used for the initialization.

# EPlaneCropper::Load

Loads the EPlaneCropper configuration. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from

# EPlaneCropper::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EPlaneCropper& operator=(
 const EPlaneCropper& other
 )
```

### Parameters

*other*

An other EPlaneCropper.

# EPlaneCropper::GetPlane

# EPlaneCropper::SetPlane

Sets/gets the new reference E3DPlane of the EPlaneCropper object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPlane GetPlane() const
void SetPlane(const E3DPlane& plane)
```

# EPlaneCropper::Save

Saves the EPlaneCropper configuration. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is written to

# 4.140. EPlaneFinder Class

A EPlaneFinder object is used to search a E3DPlane in a EPointCloud.
The algorithm searches **the largest plane** in terms of number of "inliers". A point is an "inlier" when its distance to the plane is smaller than a specified threshold (parameter 'maximum distance').
Another parameter specifies the expected ratio of inliers over the total number of points in the point cloud (by default, this is set to **0.3**).
Reducing the value of this parameter makes the search more robust but will potentially consume more time. A decimation is applied by default to accelerate the search.
Furthermore the expected normal to the plane may be specified.
The method EPlaneFinder::Find processes a EPointCloud object and returns a E3DPlane object when a plane is found in the input point cloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| DisableDecimator | Disables the default decimation.<br>The decimation should be disabled when the input point cloud is already decimated. |

| | |
|---|---|
| EnableDecimator | Enables the default decimation which reduces the input point cloud by drawing 10000 points randomly.<br>This decimation is enabled by default. The decimation accelerates the search. |
| EPlaneFinder | Creates an EPlaneFinder object. |
| Find | Searches the biggest plane in the supplied EPointCloud. If the plane is found, a E3DPlane object is returned.<br>Otherwise an exception is thrown. |
| GetExpectedCloudInliersRatio | Sets/gets the expected ratio of inliers in the EPointCloud. |
| GetMaxDeviation | Sets/gets the maximum distance of a inlier to the plane. |
| GetNormal | Returns the expected normal direction (that has been set by EPlaneFinder::SetNormal) |
| GetNormalTolerance | Returns the angle tolerance around the expected normal (that has been set by EPlaneFinder::SetNormal) |
| IsDecimatorEnabled | Returns true if the default decimator is enabled (enabled by default). |
| IsNormalSet | Returns true if the expected normal direction has been set (not set by default). |

| | |
|---|---|
| Load | Loads the plane finder configuration. The given ESerializer must have been created for reading. |
| operator= | Assignment operator. |
| Save | Saves the plane finder configuration. The given ESerializer must have been created for writing. |
| SetExpectedCloudInliersRatio | Sets/gets the expected ratio of inliers in the EPointCloud. |
| SetMaxDeviation | Sets/gets the maximum distance of a inlier to the plane. |
| SetNormal | Sets the expected normal direction and the tolerance around it.<br>These values are used to limit the scope of the plane search.<br>If the angular tolerance is not specified, a default value of **5 degrees** is assumed.<br>If the tolerance is specified, the value should be strictly positive. |
| UnsetNormal | Unsets the normal vector definition. |

E

P

# laneFinder::DisableDecimator

Disables the default decimation.
The decimation should be disabled when the input point cloud is already decimated.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void DisableDecimator(
 )
```

# EPlaneFinder::EnableDecimator

Enables the default decimation which reduces the input point cloud by drawing 10000 points randomly.
This decimation is enabled by default. The decimation accelerates the search.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void EnableDecimator(
 )
```

# EPlaneFinder::EPlaneFinder

Creates an EPlaneFinder object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void EPlaneFinder(
 )
```

```
void EPlaneFinder(
 float maxDeviation,
 float pcExpectedInCloud
 )

void EPlaneFinder(
 const EPlaneFinder& other
 )
```

### Parameters

*maxDeviation*

Maximum distance of a inlier to the plane. This value has to be strictly positive.

*pcExpectedInCloud*

This is an estimation of the ratio of inliers in the point cloud.
This optional parameter has a default value of **0.3**.
The expected ratio of inliers has an influence on the duration and on the robustness of the search:
a smaller value increases the chance of finding a smaller plane while a larger value is faster.

*other*

The EPlaneFinder object that should be copied.

# EPlaneFinder::GetExpectedCloudInliersRatio

# EPlaneFinder::SetExpectedCloudInliersRatio

Sets/gets the expected ratio of inliers in the EPointCloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetExpectedCloudInliersRatio() const

void SetExpectedCloudInliersRatio(float pcExpectedInCloud)
```

### Remarks

It is an estimation of the ratio of inliers in the point cloud.
The value set by default is **0.3**. The expected ratio of inliers has an influence on the duration and on the robustness of the search:
a smaller value increases the chance of finding the plane while a larger value makes it faster.

# EPlaneFinder::Find

Searches the biggest plane in the supplied EPointCloud. If the plane is found, a E3DPlane object is returned. Otherwise an exception is thrown.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPlane Find(
 const EPointCloud& pointCloud
 )

E3DPlane Find(
 const EPointCloud& pointCloud,
 float& effectiveInliersRatio
 )
```

### Parameters

*pointCloud*

The input point cloud in which the plane should be searched (need at least 3 points)

*effectiveInliersRatio*

This passed by reference float will contain the effective ratio of inliers

# EPlaneFinder::GetNormal

Returns the expected normal direction (that has been set by EPlaneFinder::SetNormal)

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPoint GetNormal(
 )
```

# EPlaneFinder::IsDecimatorEnabled

Returns true if the default decimator is enabled (enabled by default).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool IsDecimatorEnabled(
 )
```

# EPlaneFinder::IsNormalSet

Returns true if the expected normal direction has been set (not set by default).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool IsNormalSet(
 )
```

# EPlaneFinder::Load

Loads the plane finder configuration. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from.

# EPlaneFinder::GetMaxDeviation

# EPlaneFinder::SetMaxDeviation

Sets/gets the maximum distance of a inlier to the plane.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetMaxDeviation() const

void SetMaxDeviation(float maxDeviation)
```

# EPlaneFinder::GetNormalTolerance

Returns the angle tolerance around the expected normal (that has been set by EPlaneFinder::SetNormal)

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetNormalTolerance() const
```

# EPlaneFinder::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EPlaneFinder& operator=(
 const EPlaneFinder& other
 )
```

## Parameters

*other*

The EPlaneFinder object that should be copied.

# EPlaneFinder::Save

Saves the plane finder configuration. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is written to.

# EPlaneFinder::SetNormal

Sets the expected normal direction and the tolerance around it. These values are used to limit the scope of the plane search.
If the angular tolerance is not specified, a default value of **5 degrees** is assumed.
If the tolerance is specified, the value should be strictly positive.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetNormal(
 const E3DPoint& normal,
 float angleTolerance
 )
void SetNormal(
 float nx,
 float ny,
 float nz,
 float angleTolerance
 )
```

```
void SetNormal(
 const E3DPlane& referencePlane,
 float angleTolerance
 )
```

### Parameters

*normal*

The normal vector specifies the expected perpendicular direction of the plane.

*angleTolerance*

The angle tolerance is the maximum angular deviation around the expected normal (strictly positive value). It's set to 5 degrees by default.

*nx*

The x component of the normal vector.

*ny*

The y component of the normal vector.

*nz*

The z component of the normal vector.

*referencePlane*

The reference plane specifies the expected perpendicular direction.

# EPlaneFinder::UnsetNormal

Unsets the normal vector definition.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void UnsetNormal(
 )
```

# 4.141. EPlaneFitter Class

A EPlaneFitter object is used to fit an E3DPlane on an EPointCloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| EPlaneFitter | Constructor of an EPlaneFitter object. |
| Fit | Fits an E3DPlane on a given EPointCloud. |
| GetMinSampleCount | Sets/Gets the minimum number of samples required for fitting on each side of the shape.<br>By default, a value of **3** is assumed. |
| Load | Loads the EPlaneFitter configuration. The given ESerializer must have been created for reading. |
| operator= | Assignment operator. |
| Save | Saves the EPlaneFitter configuration. The given ESerializer must have been created for writing. |
| SetMinSampleCount | Sets/Gets the minimum number of samples required for fitting on each side of the shape.<br>By default, a value of **3** is assumed. |

# EPlaneFitter::EPlaneFitter

Constructor of an EPlaneFitter object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void EPlaneFitter(
 )

void EPlaneFitter(
 const EPlaneFitter& other
 )
```

### Parameters

*other*

Reference to the EPlaneFitter used for the initialization.

# EPlaneFitter::Fit

Fits an E3DPlane on a given EPointCloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DPlane Fit(
 const EPointCloud& pc
 )

E3DPlane Fit(
 const EPointCloud& pc,
 float& averageDistance
 )
```

### Parameters

*pc*

The reference to the point cloud.

*averageDistance*

The reference to a float which will store the average distance from this plane to the points that were used for the fit.

# EPlaneFitter::Load

Loads the EPlaneFitter configuration. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from

# EPlaneFitter::GetMinSampleCount

# EPlaneFitter::SetMinSampleCount

Sets/Gets the minimum number of samples required for fitting on each side of the shape.
By default, a value of **3** is assumed.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
int GetMinSampleCount() const

void SetMinSampleCount(int minSamples)
```

# EPlaneFitter::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
EPlaneFitter& operator=(
 const EPlaneFitter& other
 )
```

## Parameters

*other*

The EPlaneFitter object that should be copied.

# EPlaneFitter::Save

Saves the EPlaneFitter configuration. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is written to

# 4.142. EPoint Class

An exact (floating-point) location in the 2D space.

**Derived Class(es):** EFrame

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Area | Compute the oriented area of the parallelogram built on two EPoint. |
| Argument | Compute the polar argument of a EPoint object. |
| CopyTo | Copies all the data of the current EPoint object into another EPoint object and returns it. |
| Distance | Returns the distance between the addressed point and an EPoint object. |

| | |
|---|---|
| Dot | Compute the dot product of two EPoint object. |
| EPoint | Constructs a EPoint object. |
| GetCenter | Center coordinates of a EPoint object. |
| GetX | Abscissa (X coordinate) of the EPoint object |
| GetY | Ordinate (Y coordinate) of the EPoint object |
| MidPoint | Returns the middle coordinate between this EPoint object and another EPoint object. |
| Modulus | Compute the euclidian modulus of a EPoint. |
| operator- | Subtracts from the current EPoint center coordinates the center coordinates of another EPoint object. |
| operator!= | Compares the current EPoint center coordinates with the center coordinates of another EPoint object. |
| operator* | Multiplies the current EPoint center coordinates by a given multiplier. |
| operator/ | Divides the current EPoint center coordinates by a given divisor. |

| | |
|---|---|
| operator+ | Adds to the current EPoint center coordinates the center coordinates of another EPoint object. |
| operator= | Copies all the data from another EPoint object into the current EPoint object. |
| operator== | Compares the current EPoint center coordinates with the center coordinates of another EPoint object. |
| Project | Compute the orthogonal projection of a EPoint on another shape. |
| Rotate | Returns another EPoint object containing the coordinated of the rotated point. |
| SetCenter | Center coordinates of a EPoint object. |
| SetCenterXY | Sets the center coordinates of a EPoint object. |
| Square | Compute the sum of the squared coordinates of a EPoint. |
| SquaredDistance | Compute the squared distance between two EPoint. |

# EPoint::Area

Compute the oriented area of the parallelogram built on two EPoint.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float Area(
 const EPoint& Point
 )
```

### Parameters

*Point*

Second edge of the parallelogram.

### Remarks

Compute the oriented area of the parallelogram built on two EPoint. The area is counted as positive if the oriented vector pair ('first edge', 'second edge') is in the same sense that the axis frame. This oriented area can also be viewed as the z-coordinate of a vector product of two 3D vectors obtained in supplementing each edges with a third z-coordinate (setted to zero).

# EPoint::Argument

Compute the polar argument of a EPoint object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float Argument(
 )
```

### Remarks

Compute the angle (in radians) between the oriented X-axis and the vector going from the axis origin and the EPoint. If the axis frame is orthogonal, this number is also the polar argument of the EPoint.

# EPoint::GetCenter

# EPoint::SetCenter

Center coordinates of a EPoint object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetCenter() const
void SetCenter(const EPoint& center)
```

# EPoint::CopyTo

Copies all the data of the current EPoint object into another EPoint object and returns it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint* CopyTo(
 EPoint* other
 )
```

### Parameters

*other*

Pointer to the EPoint object in which the current EPoint object data have to be copied.

### Remarks

In case of a **NULL** pointer, a new EPoint object will be created and returned.

# EPoint::Distance

Returns the distance between the addressed point and an EPoint object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float Distance(
 const EPoint& point
 )
float Distance(
 const ELine& line,
 BOOL segmentOnly
 )
float Distance(
 const ECircle& circle,
 BOOL arcOnly
 )
```

### Parameters

*point*

EPoint object with which to calculate the distance.

*line*

ELine object with which to calculate the distance.

*segmentOnly*

By default (**FALSE**), the line is not restricted to a segment.

*circle*

ECircle object with which to calculate the distance.

*arcOnly*

By default (**FALSE**), the circle is not restricted to an arc.

### Remarks

Many EasyGauge members provide measurement result as a EPoint object (see EPointGauge::Center, EPointGauge::GetMeasuredPoint,...). The EPoint class has its own members to retrieve all the information pertaining to a point. Among them, the **Distance** method returns the distance between a point pair or between a point and a line segment, a circle arc or a rectangle.

# EPoint::Dot

Compute the dot product of two EPoint object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float Dot(
 const EPoint& Point
 )
```

### Parameters

*Point*

Second factor of the dot product.

# EPoint::EPoint

Constructs a EPoint object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EPoint(
 )
```

```
void EPoint(
 float centerX,
 float centerY
 )

void EPoint(
 const EPoint& other
 )
```

### Parameters

*centerX*

Center coordinates of the EPoint object.

*centerY*

Center coordinates of the EPoint object.

*other*

Another EPoint object to be copied in the new EPoint object.

# EPoint::MidPoint

Returns the middle coordinate between this EPoint object and another EPoint object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint MidPoint(
 const EPoint& Point
 )
```

### Parameters

*Point*

The other EPoint object.

# EPoint::Modulus

Compute the euclidian modulus of a EPoint.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float Modulus(
 )
```

## Remarks

Compute the squared root of the sum of the squared coordinates of an EPoint. If the axis frame is orthogonal, this number is also the euclidian norm of the EPoint.

# EPoint::operator-

Subtracts from the current EPoint center coordinates the center coordinates of another EPoint object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint operator-(
 const EPoint& point
 )
```

## Parameters

*point*

The other EPoint object.

# EPoint::operator!=

Compares the current EPoint center coordinates with the center coordinates of another EPoint object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL operator!=(
 const EPoint& point
 )
```

### Parameters

*point*

The other EPoint object.

### Remarks

Returns **TRUE** if EPoint::X or EPoint::Y are respectively different.

# EPoint::operator*

Multiplies the current EPoint center coordinates by a given multiplier.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint operator*(
 float scalar
 )
```

### Parameters

*scalar*

The multiplier.

# EPoint::operator/

Divides the current EPoint center coordinates by a given divisor.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint operator/(
 float scalar
 )
```

### Parameters

*scalar*

The divisor.

# EPoint::operator+

Adds to the current EPoint center coordinates the center coordinates of another EPoint object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint operator+(
 const EPoint& point
 )
```

### Parameters

*point*

The other EPoint object.

# EPoint::operator=

Copies all the data from another EPoint object into the current EPoint object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint& operator=(
 const EPoint& other
 )
```

## Parameters

*other*

EPoint object to be copied.

# EPoint::operator==

Compares the current EPoint center coordinates with the center coordinates of another EPoint object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL operator==(
 const EPoint& point
 )
```

## Parameters

*point*

The other EPoint object.

## Remarks

Returns **TRUE** if both EPoint::X and EPoint::Y are respectively the same.

# EPoint::Project

Compute the orthogonal projection of a EPoint on another shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint Project(
 const ELine& shape
 )
EPoint Project(
 const ECircle& shape
 )
```

### Parameters

*shape*

Shape object to which point is projected

### Remarks

Compute the orthogonal projection of a EPoint on another shape. This computation is only valid when the axis frame is orthogonal.

# EPoint::Rotate

Returns another EPoint object containing the coordinated of the rotated point.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint Rotate(
 float angle
 )
```

### Parameters

*angle*

Rotation angle (in radians)

### Remarks

Rotates a EPoint around the origin **(0, 0)** by an angle of **angle** radians. By definition, the smallest (in absolute value) rotation of the oriented X-Axis toward the oriented Y-Axis is choosen as the positive sense of rotation. In a direct frame, this is also the trigonometric sense (counter clockwise).

# EPoint::SetCenterXY

Sets the center coordinates of a EPoint object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetCenterXY(
 float centerX,
 float centerY
 )
```

### Parameters

*centerX*

Center coordinates of the EPoint object.

*centerY*

Center coordinates of the EPoint object.

# EPoint::Square

Compute the sum of the squared coordinates of a EPoint.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float Square(
 )
```

### Remarks

Compute the sum of the squared coordinates of a EPoint. If the axis frame is orthogonal, this sum of squares is also the squared euclidian norm of the EPoint object.

# EPoint::SquaredDistance

Compute the squared distance between two EPoint.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float SquaredDistance(
 const EPoint& Point
 )
```

### Parameters

*Point*

Second EPoint.

### Remarks

Compute the sum of squared coordinates differences of two EPoint. If the axis frame is orthogonal, this number is also the squared euclidian distance between two EPoint.

# EPoint::GetX

Abscissa (X coordinate) of the EPoint object

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetX() const
```

## EPoint::GetY

Ordinate (Y coordinate) of the EPoint object

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetY() const
```

# 4.143. EPointCloud Class

Represents a 3D point cloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

### Methods

| | |
|---|---|
| AddPoint | Adds a point to the EPointCloud. |
| AddPointCloud | Adds the points of another point cloud to EPointCloud. |
| AddPoints | Adds a vector of points to the EPointCloud. |

| | |
|---|---|
| Clear | Empties the EPointCloud. |
| EPointCloud | Creates an EPointCloud object. |
| FillPointsBuffer | Copies an external points buffer into the internal points buffer. |
| GetNumPoints | Number of points in the EPointCloud. |
| GetPoint | Retrieves a point from the EPointCloud. |
| GetPointsBuffer | Retrieves a pointer to the internal points buffer. |
| Load | Loads the EPointCloud. The given ESerializer must have been created for reading. |
| LoadPCD | Loads an EPointCloud stored in the PCD (Point Cloud Library) file format.<br>ASCII and binary formats are compatible. |
| operator= | Assignment operator. |
| Save | Saves the EPointCloud. The given ESerializer must have been created for writing. |
| SavePCD | Saves the EPointCloud in the PCD (Point Cloud Library) file format.<br>ASCII and binary formats are available. |

| | |
|---|---|
| Serialize | Serializes the EPointCloud. |

# PointCloud::AddPoint

Adds a point to the EPointCloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void AddPoint(
 const E3DPoint& point
 )
```

### Parameters

*point*

Point to add to the point cloud.

# EPointCloud::AddPointCloud

Adds the points of another point cloud to EPointCloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void AddPointCloud(
 const EPointCloud& cloud
 )
```

### Parameters

*cloud*

Point cloud whose points will be added to the point cloud.

# EPointCloud::AddPoints

Adds a vector of points to the EPointCloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void AddPoints(
 const std::vector<Euresys::Open_eVision_2_10::Easy3D::E3DPoint>& points
 )
```

### Parameters

*points*

Vector of points to add to the point cloud.

# EPointCloud::Clear

Empties the EPointCloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Clear(
 )
```

# EPointCloud::EPointCloud

Creates an EPointCloud object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EPointCloud(
 )
void EPointCloud(
 const EPointCloud& other
 )
```

### Parameters

*other*

Reference to the EPointCloud used for the initialization.

# EPointCloud::FillPointsBuffer

Copies an external points buffer into the internal points buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void FillPointsBuffer(
 void* pointsBuffer,
 int numPoints
 )
```

### Parameters

*pointsBuffer*

Address of the external points buffer.

*numPoints*

Number of points in the external points buffer.

### Remarks

The buffer must contain points in the form of triplets of 32bits floats stored in the (X,Y,Z) order.

# EPointCloud::GetPoint

Retrieves a point from the EPointCloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPoint GetPoint(
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index of the point to be retrieved.

# EPointCloud::Load

Loads the EPointCloud. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from.

# EPointCloud::LoadPCD

Loads an EPointCloud stored in the PCD (Point Cloud Library) file format.
ASCII and binary formats are compatible.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void LoadPCD(
 const std::string& path
 )
```

### Parameters

*path*

The full path of the destination file.

### Remarks

The PCD file format is documented here: http://pointclouds.org/documentation/tutorials/pcd_file_format.php.
Use EPointCloud::SavePCD to save to PCD file.

# EPointCloud::GetNumPoints

Number of points in the EPointCloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
int GetNumPoints() const
```

# EPointCloud::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EPointCloud& operator=(
 const EPointCloud& other
 )
```

## Parameters

*other*

The EPointCloud object that should be copied.

# EPointCloud::GetPointsBuffer

Retrieves a pointer to the internal points buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
void* GetPointsBuffer()
```

# EPointCloud::Save

Saves the EPointCloud. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Save(
 ESerializer* serializer
 )
```

## Parameters

*serializer*

The ESerializer object that is written to.

# EPointCloud::SavePCD

Saves the EPointCloud in the PCD (Point Cloud Library) file format.
ASCII and binary formats are available.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SavePCD(
 const std::string& path,
 bool binary
 )
```

### Parameters

*path*

The full path of the destination file.

*binary*

Optional parameter, activates the binary file format (default is false).

### Remarks

The PCD file format is documented here: http://pointclouds.org/documentation/tutorials/pcd_file_format.php.
Use EPointCloud::LoadPCD to load from PCD file.

# EPointCloud::Serialize

Serialized the EPointCloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or written to.

# 4.144. EPointCloudFactory Class

Manages a context for creating point clouds of specific shapes.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| CreateCubicPointCloud | Creates a point cloud in the shape of a cube. |
| CreateRectangularPointCloud | Creates a point cloud in the shape of a rectangular parallelepiped. |
| CreateSphericPointCloud | Creates a point cloud in the shape of a sphere. |

E

P

# ointCloudFactory::CreateCubicPointCloud

Creates a point cloud in the shape of a cube.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
EPointCloud CreateCubicPointCloud(
 E3DPoint center,
 float size,
 float roll,
 float pitch,
 float yaw,
 OEV_UINT32 numSamples
 )
```

### Parameters

*center*

Center of the cube.

*size*

Edge size of the cube.

*roll*

Roll (rotation along the X axis) of the cube.

*pitch*

Pitch (rotation along the Y axis) of the cube.

*yaw*

Yaw (rotation along the Z axis) of the cube.

*numSamples*

Number of points along each edge of the cube.

# EPointCloudFactory::CreateRectangularPointCloud

Creates a point cloud in the shape of a rectangular parallelepiped.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
EPointCloud CreateRectangularPointCloud(
 E3DPoint center,
 float width,
 float height,
 float depth,
 float roll,
 float pitch,
 float yaw,
 OEV_UINT32 numSamples
 )
```

### Parameters

*center*

Center of the rectangular parallelepiped.

*width*

Width (size along the X axis before rotation) of the rectangular parallelepiped.

*height*

Height (size along the Y axis before rotation) of the rectangular parallelepiped.

*depth*

Depth (size along the Z axis before rotation) of the rectangular parallelepiped.

*roll*

Roll (rotation along the X axis) of the rectangular parallelepiped.

*pitch*

Pitch (rotation along the Y axis) of the rectangular parallelepiped.

*yaw*

Yaw (rotation along the Z axis) of the rectangular parallelepiped.

*numSamples*

Number of points along each edge of the rectangular parallelepiped.

# EPointCloudFactory::CreateSphericPointCloud

Creates a point cloud in the shape of a sphere.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EPointCloud CreateSphericPointCloud(
 E3DPoint& center,
 float radius,
 int numCircles,
 int numSamples
 )
```

### Parameters

*center*

Center of the sphere.

*radius*

Radius of the sphere.

*numCircles*

Number of parallels and meridians to be rendered.

*numSamples*

Number of points along each meridian and parallel.

# 4.145. EPointCloudStatistics Class

Manages a context for retrieving statistics on an EPointCloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| GetPointCloudBounds | Retrieves the bounds of an EPointCloud. |

GetPointCloudCentroid

# ointCloudStatistics::GetPointCloudBounds

E Retrieves the centroid (arithmetic mean position of all the points, also known as center of gravity or barycenter) of an EPointCloud.
P It is possible to get the centroid of a sphere or a rectangle (rectangular parallelepiped) shape inside the point cloud.
The sphere is defined by it's center and radius, in the EPointCloud coordinate system.
The 3D rectangle is defined by 3 ranges, in X, Y and Z axis, in the EPointCloud coordinate system.
An exception will be thrown if no point is present in the shape or the point cloud.

Retrieves the bounds of an EPointCloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void GetPointCloudBounds(
 const EPointCloud& cloud,
 EFloatRange& rangeX,
 EFloatRange& rangeY,
 EFloatRange& rangeZ
 )
```

## Parameters

*cloud*

Point cloud.

*rangeX*

Bounds of the point cloud along the X direction.

*rangeY*

Bounds of the point cloud along the Y direction.

*rangeZ*

Bounds of the point cloud along the Z direction.

# EPointCloudStatistics::GetPointCloudCentroid

Retrieves the centroid (arithmetic mean position of all the points, also known as center of gravity or barycenter) of an EPointCloud.
It is possible to get the centroid of a sphere or a rectangle (rectangular parallelepiped) shape inside the point cloud.
The sphere is defined by it's center and radius, in the EPointCloud coordinate system.
The 3D rectangle is defined by 3 ranges, in X, Y and Z axis, in the EPointCloud coordinate system.
An exception will be thrown if no point is present in the shape or the point cloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPoint GetPointCloudCentroid(
 const EPointCloud& cloud
 )

E3DPoint GetPointCloudCentroid(
 const EPointCloud& cloud,
 const E3DPoint& sphereCenter,
 float sphereRadius
 )

E3DPoint GetPointCloudCentroid(
 const EPointCloud& cloud,
 const EFloatRange& rangeX,
 const EFloatRange& rangeY,
 const EFloatRange& rangeZ
 )
```

## Parameters

*cloud*

Point cloud.

*sphereCenter*

The position of the center of the sphere.

*sphereRadius*

The radius of the sphere.

*rangeX*

The bounds along the X direction.

*rangeY*

The bounds along the Y direction.

*rangeZ*

The bounds along the Z direction.

# 4.146. EPointCloudToZMapConverter Class

Computes a EZMap from a EPointCloud. The value of the pixels of the ZMap are the distance between the 3D points and the reference plane.
All 3D points under the reference plane are discarded.
Various options can be set with methods EPointCloudToZMapConverter, EPointCloudToZMapConverter::SetFillMode, EPointCloudToZMapConverter, EPointCloudToZMapConverter, EPointCloudToZMapConverter...
When the conversion is called without defining specific parameters, the algorithm uses the following options:
- The reference plane is the horizontal plane.
- The orientation vector is selected automatically.
- The origin is set as the lowest left position of the projected point cloud on the reference plane.
- The resolution (the dimensions of the Z map) is estimated to have approximately one Point Cloud point per ZMap pixels.
- The scale is calculated from the point cloud ranges and the estimated resolution.
- The fill mode is enabled and the method is set to 'EFillUndefinedPixelsDirection_Local' (see method EDepthMap8::FillUndefinedPixels).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| Convert | Computes a EZMap from a world space EPointCloud. The value of the pixels of the ZMap are the distance between the 3D points and the reference plane. Various options can be set with methods EPointCloudToZMapConverter, EPointCloudToZMapConverter, EPointCloudToZMapConverter::SetMapSize, ... |
| EnableFillMode | Enables or disables fill mode. Fill mode parameters are defined by method EPointCloudToZMapConverter::SetFillMode.<br>Fill mode is enabled by default. If fill mode is disable, undefined pixels may remain in the EZMap. |

| | |
|---|---|
| EPointCloudToZMapConverter | Creates a EPointCloudToZMapConverter object. |
| GetExtension | Sets a metric value used to enlarge the point cloud 3D domain. That value affects X,Y and Z directions and can be used to generate an EZMap with borders of undefined pixels.<br>Default value is **0**, which means a ZMap without border. |
| GetFillUndefinedPixelsDirection | Gets the undefined pixel fill direction (see EFillUndefinedPixelsDirection). |
| GetFillUndefinedPixelsMethod | Gets the undefined pixel fill method (see EFillUndefinedPixelsMethod). |
| GetMapHeight | Gets the required height (number of pixels) of the generated EZMap.<br>By default, the required size is not set. |
| GetMapWidth | Gets the required width (number of pixels) of the generated EZMap.<br>By default, the required size is not set. |
| GetMapXResolution | Gets the resolution of the EZMap pixels along the X axis. |
| GetMapYResolution | Gets the resolution of the EZMap pixels along the Y axis. |
| GetMapZResolution | Gets/sets the EZMap Z resolution, in world space units per gray value.<br>The resolution is used to compute the transformation of the world Z position to an integer 8, 16 or 32 bits pixel value. |

| | |
|---|---|
| GetOrientationVector | Sets an explicit orientation for the EZMap.<br>Overides the orientation mode given by the method EPointCloudToZMapConverter. |
| GetOrientationVectorMode | Chooses the EZMap orientation from a list of pre-defined axis, automatic mode or user defined vector.<br>Use EPointCloudToZMapConverter to set an explicit orientation vector for the ZMap. |
| GetOrigin | Chooses the EZMap origin. It is the 3D world position used as origin of the ZMap upper left pixel (0,0). |
| GetReferencePlane | Sets the E3DPlane reference plane.<br>The resulting EZMap is the distance of the 3D points above that plane.<br>3D points below the reference plane are discarded. |
| GetReferencePlaneMode | Sets an axis aligned reference plane.<br>Overides the explicit reference plane given by the method EPointCloudToZMapConverter. |
| GetWorldToZMapTransform | Explicitly sets the world to ZMap transformation.<br>"SetWorldToZMapTransform" overides the settings done by EPointCloudToZMapConverter, EPointCloudToZMapConverter and EPointCloudToZMapConverter.<br>That E3DTransformMatrix transform expresses how the world positions are transformed to the EZMap space.<br>The matrix must be a rigid transformation (translation and rotation only).<br>The resolution and scales of the ZMap are defined by EPointCloudToZMapConverter, EPointCloudToZMapConverter and EPointCloudToZMapConverter methods. |

| | |
|---|---|
| GetZMaptoWorldTransform | Explicitly sets the ZMap to World transformation.<br>"SetZMaptoWorldTransform" overides the settings done by EPointCloudToZMapConverter, EPointCloudToZMapConverter and EPointCloudToZMapConverter.<br>That E3DTransformMatrix transform expresses how the EZMap positions are transformed to the World space.<br>The matrix must be a rigid transformation (translation and rotation only).<br>The resolution and scales of the World are defined by EPointCloudToZMapConverter, EPointCloudToZMapConverter and EPointCloudToZMapConverter methods. |
| IsFillModeEnabled | Tells if the fill mode is enabled or not.<br>Use EPointCloudToZMapConverter::EnableFillMode to toggle the fill mode and EPointCloudToZMapConverter::SetFillMode to set the filling parameters. |
| Load | Loads the converter configuration. The given ESerializer must have been created for reading. |
| operator= | Assignment operator |
| operator== | Comparison operator |
| Save | Saves the converter configuration. The given ESerializer must have been created for writing. |

| | |
|---|---|
| SetExtension | Sets a metric value used to enlarge the point cloud 3D domain. That value affects X,Y and Z directions and can be used to generate an EZMap with borders of undefined pixels.<br>Default value is **0**, which means a ZMap without border. |
| SetFillMode | Inpainting options used to fill the "holes" in the EZMap. A hole exists when no 3D point is projected at that pixel position in the ZMap. |
| SetMapSize | Sets the required size of the generated EZMap; expressed in number of pixels for width and height dimensions.<br>By default, the required size is not set. |
| SetMapXYResolution | Sets the resolution (possibly anisotropic) of the EZMap pixels along the X and Y axes, in world space units per pixel (e.g mm/pixel).<br>The resolution is used to compute the ZMap size (width and height), depending on the projected point cloud on the reference plane. |
| SetMapZResolution | Gets/sets the EZMap Z resolution, in world space units per gray value.<br>The resolution is used to compute the transformation of the world Z position to an integer 8, 16 or 32 bits pixel value. |
| SetOrientationVector | Sets an explicit orientation for the EZMap.<br>Overides the orientation mode given by the method EPointCloudToZMapConverter. |

| | |
|---|---|
| SetOrientationVectorMode | Chooses the EZMap orientation from a list of pre-defined axis, automatic mode or user defined vector.<br>Use EPointCloudToZMapConverter to set an explicit orientation vector for the ZMap. |
| SetOrigin | Chooses the EZMap origin. It is the 3D world position used as origin of the ZMap upper left pixel (0,0). |
| SetReferencePlane | Sets the E3DPlane reference plane.<br>The resulting EZMap is the distance of the 3D points above that plane.<br>3D points below the reference plane are discarded. |
| SetReferencePlaneMode | Sets an axis aligned reference plane.<br>Overides the explicit reference plane given by the method EPointCloudToZMapConverter. |
| SetWorldToZMapTransform | Explicitly sets the world to ZMap transformation.<br>"SetWorldToZMapTransform" overides the settings done by EPointCloudToZMapConverter, EPointCloudToZMapConverter and EPointCloudToZMapConverter.<br>That E3DTransformMatrix transform expresses how the world positions are transformed to the EZMap space.<br>The matrix must be a rigid transformation (translation and rotation only).<br>The resolution and scales of the ZMap are defined by EPointCloudToZMapConverter, EPointCloudToZMapConverter and EPointCloudToZMapConverter methods. |

| | |
|---|---|
| SetZMaptoWorldTransform | Explicitly sets the ZMap to World transformation.<br>"SetZMaptoWorldTransform" overides the settings done by EPointCloudToZMapConverter, EPointCloudToZMapConverter and EPointCloudToZMapConverter.<br>That E3DTransformMatrix transform expresses how the EZMap positions are transformed to the World space.<br>The matrix must be a rigid transformation (translation and rotation only).<br>The resolution and scales of the World are defined by EPointCloudToZMapConverter, EPointCloudToZMapConverter and EPointCloudToZMapConverter methods. |
| UnsetMapSize | Unsets the resolution. Lets the conversion decides for the optimum resolution, depending on the projected point cloud and pixel scale. |
| UnsetMapXYResolution | Unsets the X and Y resolutions. Lets the conversion decide the optimal resolution, depending on the projected point cloud and EZMap size. |
| UnsetMapZResolution | Unsets the ZMap Z resolution. Lets the conversion decide the optimal Z resolution. |
| UnsetOrigin | Lets the conversion process decides for the EZMap origin position (based on projected point cloud on the reference plane).<br>Use EPointCloudToZMapConverter to enable and choose the ZMap origin. |
| UnsetWorldToZMapTransform | Disables the explicit world to ZMap transformation, set with EPointCloudToZMapConverter. |

# EPointCloudToZMapConverter::Convert

Computes a EZMap from a world space EPointCloud. The value of the pixels of the ZMap are the distance between the 3D points and the reference plane. Various options can be set with methods EPointCloudToZMapConverter, EPointCloudToZMapConverter, EPointCloudToZMapConverter::SetMapSize, ...

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Convert(
 const EPointCloud& cloud,
 EZMap8& zmap
 )
void Convert(
 const EPointCloud& cloud,
 EZMap16& zmap
 )
void Convert(
 const EPointCloud& cloud,
 EZMap32f& zmap
 )
```

### Parameters

*cloud*

The input 3D point cloud.

*zmap*

The generated ZMap in 8, 16 or 32 bits format.

# EPointCloudToZMapConverter::EnableFillMode

Enables or disables fill mode. Fill mode parameters are defined by method EPointCloudToZMapConverter::SetFillMode.
Fill mode is enabled by default. If fill mode is disable, undefined pixels may remain in the EZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void EnableFillMode(
 bool state
 )
```

### Parameters

*state*

Set to true to enable fill mode.

# EPointCloudToZMapConverter::EPointCloudToZMapConverter

Creates a EPointCloudToZMapConverter object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void EPointCloudToZMapConverter(
 )

void EPointCloudToZMapConverter(
 const EPointCloudToZMapConverter& other
 )
```

### Parameters

*other*

Reference to the EPointCloudToZMapConverter object used for the initialization.

# EPointCloudToZMapConverter::GetExtension

# EPointCloudToZMapConverter::SetExtension

Sets a metric value used to enlarge the point cloud 3D domain.
That value affects X,Y and Z directions and can be used to generate an EZMap with borders of undefined pixels.
Default value is **0**, which means a ZMap without border.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetExtension() const
void SetExtension(float size)
```

# EPointCloudToZMapConverter::GetFillUndefinedPixelsDirection

Gets the undefined pixel fill direction (see EFillUndefinedPixelsDirection).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsDirection
GetFillUndefinedPixelsDirection() const
```

# EPointCloudToZMapConverter::GetFillUndefinedPixelsMethod

Gets the undefined pixel fill method (see EFillUndefinedPixelsMethod).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsMethod
GetFillUndefinedPixelsMethod() const
```

# EPointCloudToZMapConverter::IsFillModeEnabled

Tells if the fill mode is enabled or not.
Use EPointCloudToZMapConverter::EnableFillMode to toggle the fill mode and
EPointCloudToZMapConverter::SetFillMode to set the filling parameters.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool IsFillModeEnabled(
 )
```

# EPointCloudToZMapConverter::Load

Loads the converter configuration. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

Pointer to the ESerializer created for reading.

# EPointCloudToZMapConverter::GetMapHeight

Gets the required height (number of pixels) of the generated EZMap.
By default, the required size is not set.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
int GetMapHeight() const
```

# EPointCloudToZMapConverter::GetMapWidth

Gets the required width (number of pixels) of the generated EZMap.
By default, the required size is not set.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
int GetMapWidth() const
```

# EPointCloudToZMapConverter::GetMapXResolution

Gets the resolution of the EZMap pixels along the X axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetMapXResolution() const
```

# EPointCloudToZMapConverter::GetMapYResolution

Gets the resolution of the EZMap pixels along the Y axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetMapYResolution() const
```

# EPointCloudToZMapConverter::GetMapZResolution

# EPointCloudToZMapConverter::SetMapZResolution

Gets/sets the EZMap Z resolution, in world space units per gray value.
The resolution is used to compute the transformation of the world Z position to an integer 8, 16 or 32 bits pixel value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetMapZResolution() const

void SetMapZResolution(float scale)
```

# EPointCloudToZMapConverter::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
EPointCloudToZMapConverter& operator=(
 const EPointCloudToZMapConverter& other
 )
```

## Parameters

*other*

Reference to the EPointCloudToZMapConverter object used for the assignment.

# EPointCloudToZMapConverter::operator==

Comparison operator

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
bool operator==(
 const EPointCloudToZMapConverter& other
 )
```

### Parameters

*other*

Reference to the EPointCloudToZMapConverter object used for the comparison.

# EPointCloudToZMapConverter::GetOrientationVector

# EPointCloudToZMapConverter::SetOrientationVector

Sets an explicit orientation for the EZMap.
Overides the orientation mode given by the method EPointCloudToZMapConverter.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPoint GetOrientationVector() const
void SetOrientationVector(const E3DPoint& direction)
```

### Remarks

The direction should be an E3DPoint representing the expected direction of the X (width) axis of the ZMap.
That direction will be used after projection on the reference plane normal.
That direction must NOT be aligned with the reference plane normal.

# EPointCloudToZMapConverter::GetOrientationVectorMode

# EPointCloudToZMapConverter::SetOrientationVectorMode

Chooses the EZMap orientation from a list of pre-defined axis, automatic mode or user defined vector.
Use EPointCloudToZMapConverter to set an explicit orientation vector for the ZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
Euresys::Open_eVision_2_10::Easy3D::EZMapOrientationVectorMode GetOrientationVectorMode
() const

void SetOrientationVectorMode(Euresys::Open_eVision_2_
10::Easy3D::EZMapOrientationVectorMode mode)
```

## Remarks

Choose between Automatic mode (default), world space axis or explicit user defined vector (see EZMapOrientationVectorMode).

# EPointCloudToZMapConverter::GetOrigin

# EPointCloudToZMapConverter::SetOrigin

Chooses the EZMap origin. It is the 3D world position used as origin of the ZMap upper left pixel (0,0).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DPoint GetOrigin() const
void SetOrigin(const E3DPoint& position)
```

### Remarks

That position will be projected on the reference plane.
To let the conversion chooses for the origin, call EPointCloudToZMapConverter::UnsetOrigin.

# EPointCloudToZMapConverter::GetReferencePlane

# EPointCloudToZMapConverter::SetReferencePlane

Sets the E3DPlane reference plane.
The resulting EZMap is the distance of the 3D points above that plane.
3D points below the reference plane are discarded.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPlane GetReferencePlane() const
void SetReferencePlane(const E3DPlane& plane)
```

# EPointCloudToZMapConverter::GetReferencePlaneMode

# EPointCloudToZMapConverter::SetReferencePlaneMode

Sets an axis aligned reference plane.
Overides the explicit reference plane given by the method EPointCloudToZMapConverter.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
Euresys::Open_eVision_2_10::Easy3D::EZMapReferencePlaneMode GetReferencePlaneMode()
const

void SetReferencePlaneMode(Euresys::Open_eVision_2_10::Easy3D::EZMapReferencePlaneMode
mode)
```

### Remarks

Choose between X, Y or Z reference plane (see EZMapReferencePlaneMode).
The plane offset is set automatically on the point cloud lowest 3D point.

# EPointCloudToZMapConverter::Save

Saves the converter configuration. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

Pointer to the ESerializer created for writing.

# EPointCloudToZMapConverter::SetFillMode

Inpainting options used to fill the "holes" in the EZMap. A hole exists when no 3D point is projected at that pixel position in the ZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetFillMode(
 Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsDirection direction,
 Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsMethod method
 )
```

### Parameters

*direction*

Direction in which the undefined pixels are filled in a depthmap from EFillUndefinedPixelsDirection

*method*

Which values used to fill the undefined pixels in a depthmap from EFillUndefinedPixelsMethod

# EPointCloudToZMapConverter::SetMapSize

Sets the required size of the generated EZMap; expressed in number of pixels for width and height dimensions.
By default, the required size is not set.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetMapSize(
 int width,
 int height
 )
```

### Parameters

*width*

The required width for the Generated ZMap.

*height*

The required height for the Generated ZMap.

# EPointCloudToZMapConverter::SetMapXYResolution

Sets the resolution (possibly anisotropic) of the EZMap pixels along the X and Y axes, in world space units per pixel (e.g mm/pixel).
The resolution is used to compute the ZMap size (width and height), depending on the projected point cloud on the reference plane.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetMapXYResolution(
 float resolution
 )
void SetMapXYResolution(
 float resolutionX,
 float resolutionY
 )
```

## Parameters

*resolution*

The resolution for the isotropic case.

*resolutionX*

The resolution for the X axis.

*resolutionY*

The resolution for the Y axis.

## Remarks

The isotropic scale, for X and Y axis is in metric world units.

# EPointCloudToZMapConverter::UnsetMapSize

Unsets the resolution. Lets the conversion decides for the optimum resolution, depending on the projected point cloud and pixel scale.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void UnsetMapSize(
 )
```

# EPointCloudToZMapConverter::UnsetMapXYResolution

Unsets the X and Y resolutions. Lets the conversion decide the optimal resolution, depending on the projected point cloud and EZMap size.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void UnsetMapXYResolution(
 )
```

# EPointCloudToZMapConverter::UnsetMapZResolution

Unsets the ZMap Z resolution. Lets the conversion decide the optimal Z resolution.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void UnsetMapZResolution(
 )
```

# EPointCloudToZMapConverter::UnsetOrigin

Lets the conversion process decides for the EZMap origin position (based on projected point cloud on the reference plane).
Use EPointCloudToZMapConverter to enable and choose the ZMap origin.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void UnsetOrigin(
 )
```

# EPointCloudToZMapConverter::UnsetWorldToZMapTransform

Disables the explicit world to ZMap transformation, set with EPointCloudToZMapConverter.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void UnsetWorldToZMapTransform(
 )
```

# EPointCloudToZMapConverter::GetWorldToZMapTransform

# EPointCloudToZMapConverter::SetWorldToZMapTransform

Explicitly sets the world to ZMap transformation.
"SetWorldToZMapTransform" overides the settings done by EPointCloudToZMapConverter, EPointCloudToZMapConverter and EPointCloudToZMapConverter.
That E3DTransformMatrix transform expresses how the world positions are transformed to the EZMap space.
The matrix must be a rigid transformation (translation and rotation only).
The resolution and scales of the ZMap are defined by EPointCloudToZMapConverter, EPointCloudToZMapConverter and EPointCloudToZMapConverter methods.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix GetWorldToZMapTransform() const
void SetWorldToZMapTransform(const E3DTransformMatrix& matrix)
```

# EPointCloudToZMapConverter::GetZMaptoWorldTransform

# EPointCloudToZMapConverter::SetZMaptoWorldTransform

Explicitly sets the ZMap to World transformation.
"SetZMaptoWorldTransform" overides the settings done by EPointCloudToZMapConverter, EPointCloudToZMapConverter and EPointCloudToZMapConverter.
That E3DTransformMatrix transform expresses how the EZMap positions are transformed to the World space.
The matrix must be a rigid transformation (translation and rotation only).
The resolution and scales of the World are defined by EPointCloudToZMapConverter, EPointCloudToZMapConverter and EPointCloudToZMapConverter methods.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix GetZMaptoWorldTransform() const
void SetZMaptoWorldTransform(const E3DTransformMatrix& matrix)
```

# 4.147. EPointGauge Class

Manages a point location gauge.

**Base Class:** EPointShape
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| CopyTo | Copies all the data of the current EPointGauge object into another EPointGauge object, and returns it. |
| Drag | Moves a handle to a new position and updates the position parameters of the gauge. |
| Draw | Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode. |
| DrawWithCurrentPen | Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode. |

| | |
|---|---|
| EPointGauge | Constructs a point measurement context. |
| GetHVConstraint | Status of the restriction on the orientation of the point location gauge or model fitting sample paths. |
| GetMeasuredPeak | Returns information pertaining to the derivative peak associated with the specified edge-crossing point, such as its area, amplitude, start, length and center. |
| GetMeasuredPoint | Returns the coordinates of an edge-crossing point, measured along the unique sample path of the point location gauge. |
| GetMinAmplitude | Offset added to the **Threshold** when a peak is to be detected. |
| GetMinArea | Minimum area value. |
| GetNumMeasuredPoints | Number of edge-crossing points along the point location gauge. |
| GetRectangularSamplingArea | Flag indicating whether the sampling area remains a rectangle when rotated, instead of becoming a parallelogram. |
| GetSmoothing | Number of pixels used for the low-pass filtering operation. |
| GetThickness | Number of parallel segments used to extract the data profile. |

| | |
|---|---|
| GetThreshold | Threshold level used to delimit significant peaks in the data profile. |
| GetTolerance | Half length of the point location gauge. |
| GetToleranceAngle | Rotation angle of the point location gauge. |
| GetTransitionChoice | Transition choice. |
| GetTransitionIndex | Index (from **0** on) of the transition to be retained when the transition choice parameter is set to ETransitionChoice_NthFromBegin or ETransitionChoice_NthFromEnd. |
| GetTransitionType | Transition type. |
| GetType | Shape type. |
| GetValid | Flag indicating if at least one valid transition has been found. |
| HitTest | Checks whether the cursor is positioned over a handle (**TRUE**) or not (**FALSE**). |
| Measure | Triggers the point location or the model fitting operation. |
| operator= | Copies all the data from another EPointGauge object into the current EPointGauge object |

| | |
|---|---|
| Plot | Draws the profile that is crossed by a point location gauge, as defined by EPlotItem. |
| PlotWithCurrentPen | Draws the profile that is crossed by a point location gauge, as defined by EPlotItem. |
| Process | Triggers the process pertaining to a shape or gauge and all the daughter gauges attach to it. |
| SetActive | Sets the flag indicating whether the gauge is active or not. |
| SetCenter | Center coordinates of a EPointGauge object. |
| SetCenterXY | Sets the center coordinates of a EPointGauge object. |
| SetHVConstraint | Status of the restriction on the orientation of the point location gauge or model fitting sample paths. |
| SetMinAmplitude | Offset added to the **Threshold** when a peak is to be detected. |
| SetMinArea | Minimum area value. |
| SetRectangularSamplingArea | Flag indicating whether the sampling area remains a rectangle when rotated, instead of becoming a parallelogram. |
| SetSmoothing | Number of pixels used for the low-pass filtering operation. |

| | |
|---|---|
| SetThickness | Number of parallel segments used to extract the data profile. |
| SetThreshold | Threshold level used to delimit significant peaks in the data profile. |
| SetTolerance | Half length of the point location gauge. |
| SetToleranceAngle | Rotation angle of the point location gauge. |
| SetTolerances | Sets the half length and the rotation angle of the point location gauge. |
| SetTransitionChoice | Transition choice. |
| SetTransitionIndex | Index (from **0** on) of the transition to be retained when the transition choice parameter is set to ETransitionChoice_NthFromBegin or ETransitionChoice_NthFromEnd. |
| SetTransitionType | Transition type. |

# EPointGauge::SetActive

Sets the flag indicating whether the gauge is active or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetActive(BOOL active)
```

### Remarks

When complex gauging is required, several gauges can be grouped together. Applying EPointGauge::Process to the mother gauge or shape triggers the measurement of the whole. Only the active gauges will participate in the process. By default, the gauge is active (**TRUE**).

# EPointGauge::SetCenter

Center coordinates of a EPointGauge object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetCenter(const EPoint& center)
```

# EPointGauge::CopyTo

Copies all the data of the current EPointGauge object into another EPointGauge object, and returns it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
EPointGauge* CopyTo(
 EPointGauge* other,
 BOOL recursive
 )
```

### Parameters

*other*

Pointer to the EPointGauge object in which the current EPointGauge object data have to be copied.

*recursive*

**TRUE** if the children gauges have to be copied as well, **FALSE** otherwise.

### Remarks

In case of a **NULL** pointer, a new EPointGauge object will be created and returned.

# EPointGauge::Drag

Moves a handle to a new position and updates the position parameters of the gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Cursor current X coordinate.

*y*

Cursor current Y coordinate.

# EPointGauge::Draw

Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )

void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )

void Draw(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

*color*

The color in which to draw the overlay.

# EPointGauge::DrawWithCurrentPen

Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

# EPointGauge::EPointGauge

Constructs a point measurement context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EPointGauge(
 )
```

```
void EPointGauge(
 float centerX,
 float centerY
 )

void EPointGauge(
 const EPointGauge& other
 )
```

### Parameters

*centerX*

Point X coordinate.

*centerY*

Point Y coordinate.

*other*

Another EPointGauge object to be copied in the new EPointGauge object.

### Remarks

With the default constructor, all the parameters are initialized to their respective default values. With the copy constructor, the constructed point measurement context is based on a pre-existing EPointGauge object. The gauge children are also copied. Hierarchy copying through a copy constructor is always recursive. To disable this recursion, use instead the EPointGauge::CopyTo method.

# EPointGauge::GetMeasuredPeak

Returns information pertaining to the derivative peak associated with the specified edge-crossing point, such as its area, amplitude, start, length and center.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPeak GetMeasuredPeak(
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index of the edge-crossing point along the probed line segment, between **0** and EPointGauge::NumMeasuredPoints (excluded).

### Remarks

If **index** is left unchanged from its default value (i.e. **~0 = 0xFFFFFFFF**), the peak associated to the default edge-crossing point is inspected. This default point is chosen according to the transition choice parameter that is managed by the EPointGauge::TransitionChoice property.

**Note.** For this method to succeed, it is necessary to previously call EPointGauge::Measure.

# EPointGauge::GetMeasuredPoint

Returns the coordinates of an edge-crossing point, measured along the unique sample path of the point location gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetMeasuredPoint(
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index of the edge-crossing point along the probed line segment, between **0** and EPointGauge::NumMeasuredPoints (excluded).

### Remarks

These coordinates pertain to the World space; they are expressed in the reference frame to which the current EPointGauge object belongs. An EPointGauge object features only one sample path, which contrasts with the other kinds of gauges. The argument **index** specifies the index of the edge-crossing point that is considered along this unique sample path. If **index** is left unchanged from its default value (i.e. **~0= 0xFFFFFFFF**), the default edge-crossing point is inspected. This default point is chosen according to the transition choice parameter that is managed by the EPointGauge::TransitionChoice property.

**Note.** For this method to succeed, it is necessary to previously call EPointGauge::Measure.

# EPointGauge::HitTest

Checks whether the cursor is positioned over a handle (**TRUE**) or not (**FALSE**).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL HitTest(
 BOOL daughters
 )
```

## Parameters

*daughters*

**TRUE** if the daughters gauges handles have to be considered as well.

# EPointGauge::GetHVConstraint

# EPointGauge::SetHVConstraint

Status of the restriction on the orientation of the point location gauge or model fitting sample paths.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetHVConstraint()
void SetHVConstraint(BOOL bHVConstraint)
```

## Remarks

*Sample paths* are the point location gauges placed along the model to be fitted.

# EPointGauge::Measure

Triggers the point location or the model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Measure(
 EROIBW8* sourceImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image.

### Remarks

When this method is called, and if not enough valid points are detected, then the method returns directly, and the measured gauge is set to the nominal parameters.

# EPointGauge::GetMinAmplitude

# EPointGauge::SetMinAmplitude

Offset added to the **Threshold** when a peak is to be detected.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetMinAmplitude()
void SetMinAmplitude(OEV_UINT32 un32MinAmplitude)
```

### Remarks

When analyzing a derivative profile, a peak is made up of consecutive pixel values above **Threshold**. To detect weak [strong] transitions, lower [raise] the **Threshold** value. To avoid interference of noise, an additional parameter is provided. The **MinAmplitude** parameter is an offset added to **Threshold** when a peak is to be detected. When the pixel values of the derivative profile do not reach **Threshold + MinAmplitude**, the peak is not taken into account. Anyway, when a peak is taken into account, all the pixels with values above **Threshold** are considered (for more accuracy). Setting the **MinAmplitude** value to **0** merely cancels its effect.

# EPointGauge::GetMinArea

# EPointGauge::SetMinArea

Minimum area value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetMinArea()
void SetMinArea(OEV_UINT32 un32MinArea)
```

### Remarks

A transition is detected if its derivative peak reaches **Threshold** + **MinAmplitude** value, and then declared valid if the area between the peak curve and the horizontal at level **Threshold** reaches the **MinArea** value.

# EPointGauge::GetNumMeasuredPoints

Number of edge-crossing points along the point location gauge.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumMeasuredPoints()
```

# EPointGauge::operator=

Copies all the data from another EPointGauge object into the current EPointGauge object

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPointGauge& operator=(
 const EPointGauge& other
 )
```

## Parameters

*other*

EPointGauge object to be copied

# EPointGauge::Plot

Draws the profile that is crossed by a point location gauge, as defined by EPlotItem.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Plot(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )

void Plot(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )

void Plot(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawItems*

Boolean combination of EPlotItem members, that indicates which items must be displayed.

*width*

Width of the plot.

*height*

Height of the plot.

*originX*

Origin point coordinates of the plot along the X axis.

*originY*

Origin point coordinates of the plot along the Y axis.

*color*

The color in which to draw the overlay.

# EPointGauge::PlotWithCurrentPen

Draws the profile that is crossed by a point location gauge, as defined by EPlotItem.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void PlotWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawItems*

Boolean combination of EPlotItem members, that indicates which items must be displayed.

*width*

Width of the plot.

*height*

Height of the plot.

*originX*

Origin point coordinates of the plot along the X axis.

*originY*

Origin point coordinates of the plot along the Y axis.

# EPointGauge::Process

Triggers the process pertaining to a shape or gauge and all the daughter gauges attach to it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Process(
 EROIBW8* sourceImage,
 BOOL daughters
 )
```

### Parameters

*sourceImage*

Pointer to the source image.

*daughters*

Flag indicating whether the daughters shapes inherit of the same behavior.

### Remarks

When complex gauging is required, several gauges can be grouped together. Applying **Process** to the mother gauge or shape triggers the measurement of the whole. Only the active gauges will participate in the process.

# EPointGauge::GetRectangularSamplingArea

# EPointGauge::SetRectangularSamplingArea

Flag indicating whether the sampling area remains a rectangle when rotated, instead of becoming a parallelogram.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetRectangularSamplingArea()
void SetRectangularSamplingArea(BOOL bRectangularSamplingArea)
```

### Remarks

By default, this flag is set to **TRUE**: the sampling area always remains a rectangle. Setting this property is only useful when the thickness transition parameter is greater than 1. In fact, when thickness transition parameter is equal to 1, rectangle and parallelogram reduce to the same segment.

# EPointGauge::SetCenterXY

Sets the center coordinates of a EPointGauge object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetCenterXY(
 float centerX,
 float centerY
 )
```

### Parameters

*centerX*

Center coordinates of the EPointGauge object.

*centerY*

Center coordinates of the EPointGauge object.

# EPointGauge::SetTolerances

Sets the half length and the rotation angle of the point location gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetTolerances(
 float tolerance,
 float angle
 )
```

### Parameters

*tolerance*

Half length of the point location gauge. The default value is **10**.

*angle*

Rotation angle of the point location gauge. The default value is **0**.

### Remarks

By default, the point location gauge length value is 20 (2x10), which means 20 pixels when the field of view is not calibrated and 20 "units" in case of a calibrated field of view. The sign of the rotation angle depends whether the field of view is calibrated or not. * When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value. * When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# EPointGauge::GetSmoothing

# EPointGauge::SetSmoothing

Number of pixels used for the low-pass filtering operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetSmoothing()
```

```
void SetSmoothing(OEV_UINT32 un32Smoothing)
```

### Remarks

To reduce the effect of noise, the profile data can be low-pass filtered along the point location gauge direction.

# EPointGauge::GetThickness

# EPointGauge::SetThickness

Number of parallel segments used to extract the data profile.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetThickness()
void SetThickness(OEV_UINT32 un32Thickness)
```

### Remarks

To reduce the effect of noise and/or strengthen a transition, several parallel profiles can be accumulated.

# EPointGauge::GetThreshold

# EPointGauge::SetThreshold

Threshold level used to delimit significant peaks in the data profile.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetThreshold()
void SetThreshold(OEV_UINT32 un32Threshold)
```

### Remarks

When analyzing a derivative profile, a peak is made up of consecutive pixel values above **Threshold**. To detect weak [strong] transitions, lower [raise] the **Threshold** value. To avoid interference of noise, an additional parameter is provided. The **MinAmplitude** parameter is an offset added to **Threshold** when a peak is to be detected. When the pixel values of the derivative profile do not reach **Threshold + MinAmplitude**, the peak is not taken into account. Anyway, when a peak is taken into account, all the pixels with values above **Threshold** are considered (for more accuracy). Setting the **MinAmplitude** value to **0** merely cancels its effect.

# EPointGauge::GetTolerance

# EPointGauge::SetTolerance

Half length of the point location gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetTolerance()
void SetTolerance(float tolerance)
```

### Remarks

By default, the length of the point location gauge is 20 (2x10), which means 20 pixels when the field of view is not calibrated, and 20 physical units in case of a calibrated field of view.

# EPointGauge::GetToleranceAngle

# EPointGauge::SetToleranceAngle

Rotation angle of the point location gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetToleranceAngle()
void SetToleranceAngle(float toleranceAngle)
```

## Remarks

By default, the rotation angle of the point location gauge is **0**. The sign of the rotation angle depends whether the field of view is calibrated or not. * When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value. * When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# EPointGauge::GetTransitionChoice

# EPointGauge::SetTransitionChoice

Transition choice.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::ETransitionChoice GetTransitionChoice()

void SetTransitionChoice(Euresys::Open_eVision_2_10::ETransitionChoice
eTransitionChoice)
```

### Remarks

Several peaks may be detected along a point location gauge. This parameter helps to select the desired transition. In case of ETransitionChoice_NthFromBegin or ETransitionChoice_NthFromEnd transition choice, set EPointGauge::TransitionIndex to specify the desired transition. By default, the selected transition corresponds to the one with the largest amplitude (ETransitionChoice_LargestAmplitude).

# EPointGauge::GetTransitionIndex

# EPointGauge::SetTransitionIndex

Index (from **0** on) of the transition to be retained when the transition choice parameter is set to NthFromBegin or NthFromEnd.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetTransitionIndex()

void SetTransitionIndex(OEV_UINT32 un32TransitionIndex)
```

### Remarks

Several peaks may be detected along a point location gauge. This parameter helps to select the desired transition. By default, the first transition is retained (the index value is **0**).

# EPointGauge::GetTransitionType

# EPointGauge::SetTransitionType

Transition type.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::ETransitionType GetTransitionType()
void SetTransitionType(Euresys::Open_eVision_2_10::ETransitionType eTransitionType)
```

## Remarks

The type of a transition tells whether it crosses increasing or decreasing gray-level values. This helps discriminate between nearby edges of an object. By default, the searched transition type is indifferently a black to white or a white to black transition (ETransitionType_BwOrWb).

# EPointGauge::GetType

Shape type.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::EShapeType GetType()
```

# EPointGauge::GetValid

Flag indicating if at least one valid transition has been found.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetValid()
```

### Remarks

A **FALSE** value means that no measurement has been performed. A **TRUE** value means that a transition was found along the sample path defined by the EPointGauge, and thus a point was measured.

# 4.148. EPointShape Class

Manages a point shape context.

**Base Class:** EShape
**Derived Class(es):** EPatternFinder EPointGauge
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Closest | Find the daughter shape that is the closest to this shape. To retrieve the closest shape, use EShape::ClosestShape. |

| | |
|---|---|
| CopyTo | |
| | Copies all the data of the current EPointShape object into another EPointShape object, and returns it. |
| Drag | Moves a handle to a new position and updates the position parameters of the gauge. |
| Draw | Draws a graphical representation of a shape, as defined by EDrawingMode. |
| DrawWithCurrentPen | Draws a graphical representation of a shape, as defined by EDrawingMode. |
| GetCenter | Center coordinates of a EPointShape object. |
| GetCenterX | Abscissa of the origin point of the frame. |
| GetCenterY | Ordinate of the origin point of the frame. |
| GetType | Shape type. |
| HitTest | Checks if there is a handle under the cursor. |
| operator!= | Compares the instance another EPointShape object and returns TRUE if they are not identical. |

| | |
|---|---|
| operator= | Copies all the data from another EPointShape object into the current EPointShape object |
| operator== | Compares the instance another EPointShape object and returns TRUE if they are identical. |
| SetCenter | Center coordinates of a EPointShape object. |
| SetCenterXY | Sets the center coordinates of a EPointShape object. |

# EPointShape::GetCenter

# EPointShape::SetCenter

Center coordinates of a EPointShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetCenter() const
void SetCenter(const EPoint& point)
```

# EPointShape::GetCenterX

Abscissa of the origin point of the frame.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCenterX() const
```

# EPointShape::GetCenterY

Ordinate of the origin point of the frame.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCenterY() const
```

# EPointShape::Closest

Find the daughter shape that is the closest to this shape. To retrieve the closest shape, use EShape::ClosestShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Closest(
 )
```

# EPointShape::CopyTo

Copies all the data of the current EPointShape object into another EPointShape object, and returns it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPointShape* CopyTo(
 EPointShape* other,
 BOOL recursive
 )
```

## Parameters

*other*

Pointer to the EPointShape object in which the current EPointShape object data have to be copied.

*recursive*

**TRUE** if the children gauges have to be copied as well, **FALSE** otherwise.

## Remarks

In case of a **NULL** pointer, a new EPointShape object will be created and returned.

# EPointShape::Drag

Moves a handle to a new position and updates the position parameters of the gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 OEV_INT32 n32CursorX,
 OEV_INT32 n32CursorY
 )
```

### Parameters

*n32CursorX*

-

*n32CursorY*

-

# EPointShape::Draw

Draws a graphical representation of a shape, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
void Draw(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

*color*

-

# EPointShape::DrawWithCurrentPen

Draws a graphical representation of a shape, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

# EPointShape::HitTest

Checks if there is a handle under the cursor.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL HitTest(
 BOOL bDaughters
 )
```

### Parameters

*bDaughters*

Indicates if the check must be done in the whole hierarchy or just this object.

# EPointShape::operator!=

Compares the instance another EPointShape object and returns TRUE if they are not identical.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL operator!=(
 const EPointShape& other
 )
```

### Parameters

*other*

EPointShape object to be compared

# EPointShape::operator=

Copies all the data from another EPointShape object into the current EPointShape object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPointShape& operator=(
 const EPointShape& other
 )
```

### Parameters

*other*

EPointShape object to be copied

# EPointShape::operator==

Compares the instance another EPointShape object and returns TRUE if they are identical.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL operator==(
 const EPointShape& other
 )
```

### Parameters

*other*

EPointShape object to be compared

# EPointShape::SetCenterXY

Sets the center coordinates of a EPointShape object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetCenterXY(
 float centerX,
 float centerY
 )
```

### Parameters

*centerX*

Center coordinates of the EPointShape object.

*centerY*

Center coordinates of the EPointShape object.

## EPointShape::GetType

Shape type.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::EShapeType GetType()
```

# 4.149. EPolygonRegion Class

Manages a complete context for an ERegion shaped like a polygon.

**Base Class:** ERegion

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Drag | Moves the specified handle to a new position and updates all placement parameters of the region. |
| EPolygonRegion | Constructs an EPolygonRegion context. |
| GetPoints | List of vertices of the region |
| HitTest | Detects if the cursor is placed over one of the dragging handles. |
| InsertPoint | Insert a vertice between two existing ones |
| Load | Loads the EPolygonRegion. The given ESerializer must have been created for reading. |
| operator!= | Checks if this EPolygonRegion instance is not strictly equal to another |
| operator= | Assignment operator. |
| operator== | Checks if this EPolygonRegion instance is strictly equal to another |
| RemovePoint | Remove a vertice |

| | |
|---|---|
| Rotate | Creates a new EPolygonRegion by rotating the EPolygonRegion. |
| Save | Saves the EPolygonRegion. The given ESerializer must have been created for writing. |
| Scale | Creates a new EPolygonRegion by scaling the region. |
| SetPoints | List of vertices of the region |
| Translate | Creates a new EPolygonRegion by translating the EPolygonRegion. |

E

P

# olygonRegion::Drag

Moves the specified handle to a new position and updates all placement parameters of the region.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Drag(
 int x,
 int y,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

### Parameters

*x*

x-coordinate of the mouse cursor.

*y*

y-coordinate of the mouse cursor.

*zoomX*

Horizontal zoom factor. By default, true scale is used.

*zoomY*

Vertical zoom factor. By default, true scale is used.

*panX*

Horizontal pan offset. By default, no pan is added.

*panY*

Vertical pan offset. By default, no pan is added.

### Remarks

If zooming and/or panning were used when drawing the region, the same values must be used with EPolygonRegion::HitTest and EPolygonRegion::Drag.

# EPolygonRegion::EPolygonRegion

Constructs an EPolygonRegion context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EPolygonRegion(
 )
void EPolygonRegion(
 const std::vector<Euresys::Open_eVision_2_10::EPoint>& points
 )
void EPolygonRegion(
 const EPolygonRegion& other
 )
```

### Parameters

*points*

The list of vertices of the EPolygonRegion.

*other*

EPolygonRegion context to copy.

# EPolygonRegion::HitTest

Detects if the cursor is placed over one of the dragging handles.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EEditionMode HitTest(
 int x,
 int y,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*x*

x-coordinate of the mouse cursor.

*y*

y-coordinate of the mouse cursor.

*zoomX*

Horizontal zoom factor. By default, true scale is used.

*zoomY*

Vertical zoom factor. By default, true scale is used.

*panX*

Horizontal pan offset. By default, no pan is added.

*panY*

Vertical pan offset. By default, no pan is added.

### Remarks

Returns a handle identifier, as defined by EEditionMode.
If zooming and/or panning were used when drawing the region, the same values must be used with EPolygonRegion::HitTest and EPolygonRegion::Drag.

# EPolygonRegion::InsertPoint

Insert a vertice between two existing ones

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool InsertPoint(
 int x,
 int y,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

### Parameters

*x*

x-coordinate of the mouse cursor.

*y*

y-coordinate of the mouse cursor.

*zoomX*

Horizontal zoom factor. By default, true scale is used.

*zoomY*

Vertical zoom factor. By default, true scale is used.

*panX*

Horizontal pan offset. By default, no pan is added.

*panY*

Vertical pan offset. By default, no pan is added.

### Remarks

If zooming and/or panning were used when drawing the region, the same values must be used with EPolygonRegion::HitTest and EPolygonRegion::Drag.

# EPolygonRegion::Load

Loads the EPolygonRegion. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# EPolygonRegion::operator!=

Checks if this EPolygonRegion instance is not strictly equal to another

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool operator!=(
 const EPolygonRegion& other
 )
```

### Parameters

*other*

Reference to the other EPolygonRegion instance

# EPolygonRegion::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPolygonRegion& operator=(
 const EPolygonRegion& other
 )
```

### Parameters

*other*

Reference to the EPolygonRegion used for the assignment

# EPolygonRegion::operator==

Checks if this EPolygonRegion instance is strictly equal to another

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool operator==(
 const EPolygonRegion& other
 )
```

### Parameters

*other*

Reference to the other EPolygonRegion instance

# EPolygonRegion::GetPoints

# EPolygonRegion::SetPoints

List of vertices of the region

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::vector<Euresys::Open_eVision_2_10::EPoint> GetPoints() const
void SetPoints(const std::vector<Euresys::Open_eVision_2_10::EPoint>& points)
```

# EPolygonRegion::RemovePoint

Remove a vertice

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
bool RemovePoint(
 int x,
 int y,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

### Parameters

*x*

x-coordinate of the mouse cursor.

*y*

y-coordinate of the mouse cursor.

*zoomX*

Horizontal zoom factor. By default, true scale is used.

*zoomY*

Vertical zoom factor. By default, true scale is used.

*panX*

Horizontal pan offset. By default, no pan is added.

*panY*

Vertical pan offset. By default, no pan is added.

### Remarks

If zooming and/or panning were used when drawing the region, the same values must be used with EPolygonRegion::HitTest and EPolygonRegion::Drag.

# EPolygonRegion::Rotate

Creates a new EPolygonRegion by rotating the EPolygonRegion.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPolygonRegion Rotate(
 float angle
 )
```

### Parameters

*angle*

rotation angle

# EPolygonRegion::Save

Saves the EPolygonRegion. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# EPolygonRegion::Scale

Creates a new EPolygonRegion by scaling the region.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPolygonRegion Scale(
 float scale
 )
EPolygonRegion Scale(
 float scaleX,
 float scaleY
 )
```

### Parameters

*scale*

Isotropic scale

*scaleX*

Horizontal scale

*scaleY*

Vertical scale

# EPolygonRegion::Translate

Creates a new EPolygonRegion by translating the EPolygonRegion.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPolygonRegion Translate(
 float dx,
 float dy
 )
```

### Parameters

*dx*

Horizontal translation in pixel value

*dy*

Vertical translation in pixel value

# 4.150. EPrincipalAxisExtractor Class

A EPrincipalAxisExtractor object computes the principal axis analysis (PCA) on an EPointCloud and produces a E3DTransformMatrix as a result.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| EPrincipalAxisExtractor | Creates an EPrincipalAxisExtractor object. |
| Extract | Computes the E3DTransformMatrix for a given EPointCloud.<br>This will compute the PCA, then select the direction of each axis of the basis so that this basis is as close as possible as the reference transform.<br>Optionally returns the standard deviation along the 3 axis. |
| GetReferenceTransform | Sets/Gets the reference transform (E3DTransformMatrix). If not set, it will use the main basis as reference to select the direction of each axis of the new basis. |
| HasReferenceTransformSet | Returns 'true' if a reference transform (E3DTransformMatrix) has been set explicitly. |
| Load | Loads the EPrincipalAxisExtractor object configuration. The given ESerializer must have been created for reading. |
| operator= | Assignment operator. |

| | |
|---|---|
| Save | Saves the EPrincipalAxisExtractor object configuration. The given ESerializer must have been created for writing. |
| SetReferenceTransform | Sets/Gets the reference transform (E3DTransformMatrix). If not set, it will use the main basis as reference to select the direction of each axis of the new basis. |
| UnsetReferenceTransform | Unset the reference transform (E3DTransformMatrix). |

E

P

# rincipalAxisExtractor::EPrincipalAxisExtractor

Creates an EPrincipalAxisExtractor object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EPrincipalAxisExtractor(
 )
void EPrincipalAxisExtractor(
 const EPrincipalAxisExtractor& other
 )
```

## Parameters

*other*

The object used for the initialization

# EPrincipalAxisExtractor::Extract

Computes the E3DTransformMatrix for a given EPointCloud.
This will compute the PCA, then select the direction of each axis of the basis so that this basis is as close as possible as the reference transform.
Optionally returns the standard deviation along the 3 axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DTransformMatrix Extract(
 const EPointCloud& pc
 )
E3DTransformMatrix Extract(
 const EPointCloud& pc,
 float& stdDevX,
 float& stdDevY,
 float& stdDevZ
 )
```

### Parameters

*pc*

Input point cloud.

*stdDevX*

Variable to store the X component of the standard deviation.

*stdDevY*

Variable to store the Y component of the standard deviation.

*stdDevZ*

Variable to store the Z component of the standard deviation.

# EPrincipalAxisExtractor::HasReferenceTransformSet

Returns 'true' if a reference transform (E3DTransformMatrix) has been set explicitly.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool HasReferenceTransformSet(
 )
```

# EPrincipalAxisExtractor::Load

Loads the EPrincipalAxisExtractor object configuration. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 ESerializer* serializer
 )
```

## Parameters

*serializer*

The ESerializer object that is read from.

# EPrincipalAxisExtractor::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
EPrincipalAxisExtractor& operator=(
 const EPrincipalAxisExtractor& other
 )
```

### Parameters

*other*

The EPrincipalAxisExtractor object that should be copied.

# EPrincipalAxisExtractor::GetReferenceTransform

# EPrincipalAxisExtractor::SetReferenceTransform

Sets/Gets the reference transform (E3DTransformMatrix). If not set, it will use the main basis as reference to select the direction of each axis of the new basis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DTransformMatrix GetReferenceTransform() const

void SetReferenceTransform(const E3DTransformMatrix& refTransform)
```

# EPrincipalAxisExtractor::Save

Saves the EPrincipalAxisExtractor object configuration. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is written to.

# EPrincipalAxisExtractor::UnsetReferenceTransform

Unset the reference transform (E3DTransformMatrix).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void UnsetReferenceTransform(
 )
```

# 4.151. EPseudoColorLookup Class

Describes a lookup table, that is used to for pseudo-coloring (i.e. for assigning colors to gray-level images).

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EPseudoColorLookup | Default constructor of EPseudoColorLookup objects. |

SetShading

Sets up a pseudo-color mapping such that gray level **0** corresponds to color **c24Black**, gray level **255** corresponds to color **c24White**, and intermediate values are interpolated linearly between these two extremes.

# seudoColorLookup::EPseudoColorLookup

Default constructor of EPseudoColorLookup objects.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EPseudoColorLookup(
 const EPseudoColorLookup& other
 )

void EPseudoColorLookup(
 )
```

### Parameters

*other*

-

# EPseudoColorLookup::SetShading

Sets up a pseudo-color mapping such that gray level **0** corresponds to color **c24Black**, gray level **255** corresponds to color **c24White**, and intermediate values are interpolated linearly between these two extremes.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetShading(
 EC24 black,
 EC24 white,
 Euresys::Open_eVision_2_10::EColorSystem colorSystem,
 BOOL wrap
 )
```

### Parameters

*black*

Color to be mapped on a black (value **0**) pixel.

*white*

Color to be mapped on a white (value **255**) pixel.

*colorSystem*

Color system in which interpolation takes place.

*wrap*

If the color system supports a hue component, indicates whether hue wrap around must be applied.

### Remarks

Furthermore, interpolation is performed in the designated color system. Even though interpolation is performed in an arbitrary color system, the extreme colors are specified in the RGB space. To obtain interesting shades of colors, it is recommended to interpolate on the hue component alone.

# 4.152. EQRCode Class

Represents a QR code found in the search field.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Draw | Draws the QR code using a pre-defined pen. |

| | |
|---|---|
| DrawWithCurrentPen | Draws the QR code using the pen currently set in the graphical context. |
| EQRCode | Creates an EQRCode object. |
| GetDecodedStream | Decoded stream. |
| GetGeometry | Geometry of the QR code. |
| GetIsDecodingReliable | Decoding reliabililty. |
| GetLevel | Level of the QR code. |
| GetModel | Model of the QR code. |
| GetUnusedErrorCorrection | Unused error correction. |
| GetVersion | Version of the QR code. |
| operator= | - |

# EQRCode::GetDecodedStream

Decoded stream.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EQRCodeDecodedStream& GetDecodedStream()
```

# EQRCode::Draw

Draws the QR code using a pre-defined pen.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 HDC hDC,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*hDC*

-

*zoomX*

-

*zoomY*

-

*panX*

-

*panY*

-

# EQRCode::DrawWithCurrentPen

Draws the QR code using the pen currently set in the graphical context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC hDC,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*hDC*

-

*zoomX*

-

*zoomY*

-

*panX*

-

*panY*

-

# EQRCode::EQRCode

Creates an EQRCode object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EQRCode(
 )
void EQRCode(
 const EQRCode& other
 )
```

### Parameters

*other*

-

# EQRCode::GetGeometry

Geometry of the QR code.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EQRCodeGeometry& GetGeometry()
```

# EQRCode::GetIsDecodingReliable

Decoding reliabililty.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool GetIsDecodingReliable()
```

# EQRCode::GetLevel

Level of the QR code.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EQRCodeLevel GetLevel()
```

# EQRCode::GetModel

Model of the QR code.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EQRCodeModel GetModel()
```

# EQRCode::operator=

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EQRCode& operator=(
 const EQRCode& other
 )
```

### Parameters

*other*

-

# EQRCode::GetUnusedErrorCorrection

Unused error correction.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetUnusedErrorCorrection()
```

### Remarks

Returns the amount of unused error correction as a percentage. This parameter ranges from 0 to 1. Returns -1 if error correction failed (too many errors).

# EQRCode::GetVersion

Version of the QR code.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetVersion()
```

# 4.153. EQRCodeDecodedStream Class

Represents the complete decoded stream extracted from a QR code.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EQRCodeDecodedStream | Creates an EQRCodeDecodedStream object. |
| GetApplicationIndicator | Application indicator. |
| GetCodingMode | Coding mode. |
| GetDecodedStreamParts | Decoded stream parts. |

| | |
|---|---|
| GetRawBitstream | Raw bit stream. |
| operator= | - |

# EQRCodeDecodedStream::GetApplicationIndicator

Application indicator.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetApplicationIndicator()
```

## Remarks

The application indicator is relevant if the coding mode of the QR code is FNC1/AIM only.

# EQRCodeDecodedStream::GetCodingMode

Coding mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EQRCodeCodingMode GetCodingMode()
```

# EQRCodeDecodedStream::GetDecodedStreamParts

Decoded stream parts.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::vector<Euresys::Open_eVision_2_10::EQRCodeDecodedStreamPart> GetDecodedStreamParts
()
```

# EQRCodeDecodedStream::EQRCodeDecodedStream

Creates an EQRCodeDecodedStream object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EQRCodeDecodedStream(
 )

void EQRCodeDecodedStream(
 const EQRCodeDecodedStream& other
 )
```

## Parameters

*other*

-

# EQRCodeDecodedStream::operator=

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EQRCodeDecodedStream& operator=(
 const EQRCodeDecodedStream& other
 )
```

### Parameters

*other*

-

# EQRCodeDecodedStream::GetRawBitstream

Raw bit stream.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::vector<OEV_UINT8> GetRawBitstream()
```

### Remarks

The raw bit stream is the bit stream of the QR code after unmasking and error correction, but before decoding.

# 4.154. EQRCodeDecodedStreamPart Class

Represents part of a decoded stream extracted from a QR code.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EQRCodeDecodedStreamPart | Creates an EQRCodeDecodedStreamPart object. |
| GetDecodedData | Decoded data. |
| GetEncoding | Encoding. |
| operator= | - |

# EQRCodeDecodedStreamPart::GetDecodedData

Decoded data.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::vector<OEV_UINT8> GetDecodedData()
```

# EQRCodeDecodedStreamPart::GetEncoding

Encoding.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::EQRCodeEncoding GetEncoding()
```

# EQRCodeDecodedStreamPart::EQRCodeDecodedStreamPart

Creates an EQRCodeDecodedStreamPart object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EQRCodeDecodedStreamPart(
 )

void EQRCodeDecodedStreamPart(
 const EQRCodeDecodedStreamPart& other
 )
```

## Parameters

*other*

-

# EQRCodeDecodedStreamPart::operator=

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EQRCodeDecodedStreamPart& operator=(
 const EQRCodeDecodedStreamPart& other
 )
```

### Parameters

*other*

-

# 4.155. EQRCodeGeometry Class

Represents the geometry of a QR code.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Draw | Draws the QR code geometry using a pre-defined pen. |
| DrawWithCurrentPen | Draws the QR code geometry using the pen currently set in the graphical context. |

| | |
|---|---|
| EQRCodeGeometry | Creates an EQRCodeGeometry object. |
| GetFinderPatternCenters | Finder patterns centers. |
| GetPosition | Position of the QR code. |
| operator= | E-Q |

# RCodeGeometry::Draw

Draws the QR code geometry using a pre-defined pen.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 HDC hDC,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

### Parameters

*hDC*

-

*zoomX*

-

*zoomY*

-

*panX*

-

*panY*

-

# EQRCodeGeometry::DrawWithCurrentPen

Draws the QR code geometry using the pen currently set in the graphical context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC hDC,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

## Parameters

*hDC*

-

*zoomX*

-

*zoomY*

-

*panX*

-

*panY*

-

# EQRCodeGeometry::EQRCodeGeometry

Creates an EQRCodeGeometry object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EQRCodeGeometry(
 )
void EQRCodeGeometry(
 const EQRCodeGeometry& other
 )
void EQRCodeGeometry(
 const EQuadrilateral& position,
 const std::vector<Euresys::Open_eVision_2_10::EPoint>& finderPatternCenters
 )
```

### Parameters

*other*

-

*position*

-

*finderPatternCenters*

-

# EQRCodeGeometry::GetFinderPatternCenters

Finder patterns centers.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::vector<Euresys::Open_eVision_2_10::EPoint> GetFinderPatternCenters()
```

### Remarks

In case of a Micro QR code, there is only one finder pattern center. In case of another QR code, there are three finder pattern centers, returned in the following order: bottom left, top left, top right.

# EQRCodeGeometry::operator=

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EQRCodeGeometry& operator=(
 const EQRCodeGeometry& other
 )
```

### Parameters

*other*

-

# EQRCodeGeometry::GetPosition

Position of the QR code.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EQuadrilateral GetPosition()
```

# 4.156. EQRCodeReader Class

Represents the QR code reader, that is a context for the detection and decoding of QR codes.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Decode | Decodes a QR code candidate, represented as a geometry. |
| Detect | Detects all QR code candidates in the search field, and returns them as a vector of geometries. |
| EQRCodeReader | Creates an EQRCodeReader object. |
| GetCellPolarityConfidenceThreshold | Sets the minimun cell polarity confidence threshold. When the cell confidence is under the threshold, additional processing is attempted to improve the polarity detection. |
| GetDetectionMethod | Sets the detection method for finding QR codes. The method can be any combination of the EQRDetectionMethod enums. |

| | |
|---|---|
| GetDetectionTradeOff | This setting controls the trade-off between computation speed versus reliability of the detection methods. Setting the trade-off will overwrite the current settings for EQRCodeScanPrecision and EQRDetectionMethod. |
| GetFilterOutUnreliablyDecodedQRCodes | Activate or deactivate the filtering of unreliably decoded QR codes. |
| GetForegroundDetectionThreshold | Foreground detection threshold. This parameter determines how many grayscale-values a pixel should deviate from it's local background to be considered part of the foreground. |
| GetMaximumVersion | Maximum version of QR codes to be searched for. |
| GetMinimumIsotropy | QR code minimum isotropy. |
| GetMinimumScore | Minimum pattern finder score that must be reached to consider that a finder pattern has been found. |
| GetMinimumVersion | Minimum version of QR codes to be searched for. |
| GetPerspectiveMode | Sets the perspective mode. |
| GetScanPrecision | Precision of the QR code reader when scanning the search field. |
| GetSearchedModels | QR code models to be searched for. |

| | |
|---|---|
| GetTimeOut | Time-out for the EQRCodeReader::Detect, EQRCodeReader::Decode and EQRCodeReader::Read methods. |
| Read | Detects and decodes all the QR codes in the search field. |
| SetCellPolarityConfidenceThreshold | Sets the minimun cell polarity confidence threshold. When the cell confidence is under the threshold, additional processing is attempted to improve the polarity detection. |
| SetDetectionMethod | Sets the detection method for finding QR codes. The method can be any combination of the EQRDetectionMethod enums. |
| SetDetectionTradeOff | This setting controls the trade-off between computation speed versus reliability of the detection methods. Setting the trade-off will overwrite the current settings for EQRCodeScanPrecision and EQRDetectionMethod. |
| SetFilterOutUnreliablyDecodedQRCodes | Activate or deactivate the filtering of unreliably decoded QR codes. |
| SetForegroundDetectionThreshold | Foreground detection threshold. This parameter determines how many grayscale-values a pixel should deviate from it's local background to be considered part of the foreground. |
| SetMaximumVersion | Maximum version of QR codes to be searched for. |
| SetMinimumIsotropy | QR code minimum isotropy. |

| | |
|---|---|
| SetMinimumScore | Minimum pattern finder score that must be reached to consider that a finder pattern has been found. |
| SetMinimumVersion | Minimum version of QR codes to be searched for. |
| SetPerspectiveMode | Sets the perspective mode. |
| SetScanPrecision | Precision of the QR code reader when scanning the search field. |
| SetSearchedModels | QR code models to be searched for. |
| SetSearchField | Search field for the QR code reader. |
| SetTimeOut | Time-out for the EQRCodeReader::Detect, EQRCodeReader::Decode and EQRCodeReader::Read methods. |

# EQRCodeReader::GetCellPolarityConfidenceThreshold

# EQRCodeReader::SetCellPolarityConfidenceThreshold

Sets the minimun cell polarity confidence threshold. When the cell confidence is under the threshold, additional processing is attempted to improve the polarity detection.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCellPolarityConfidenceThreshold()
void SetCellPolarityConfidenceThreshold(float threshold)
```

# EQRCodeReader::Decode

Decodes a QR code candidate, represented as a geometry.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EQRCode Decode(
 const EQRCodeGeometry& geometries
 )
```

### Parameters

*geometries*

-

### Remarks

The geometry argument can either be custom-built or retrieved after a EQRCodeReader::Detect.

# EQRCodeReader::Detect

Detects all QR code candidates in the search field, and returns them as a vector of geometries.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::vector<Euresys::Open_eVision_2_10::EQRCodeGeometry> Detect(
 )
```

### Remarks

EQRCodeReader::Detect only returns candidate QR codes. These candidates can only be confirmed as actual QR codes after a successful decoding.

# EQRCodeReader::GetDetectionMethod

# EQRCodeReader::SetDetectionMethod

Sets the detection method for finding QR codes. The method can be any combination of the EQRDetectionMethod enums.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetDetectionMethod()
void SetDetectionMethod(int method)
```

### Remarks

The default value is: 'EQRDetectionMethod_Gradient|EQRDetectionMethod_AdaptiveThreshold'.

# EQRCodeReader::GetDetectionTradeOff

# EQRCodeReader::SetDetectionTradeOff

This setting controls the trade-off between computation speed versus reliability of the detection methods. Setting the trade-off will overwrite the current settings for EQRCodeScanPrecision and EQRDetectionMethod.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EQRDetectionTradeOff GetDetectionTradeOff()
void SetDetectionTradeOff(Euresys::Open_eVision_2_10::EQRDetectionTradeOff tradeOff)
```

### Remarks

The default value is **EQRCodeDetectionTradeOff_Balanced**.

# EQRCodeReader::EQRCodeReader

Creates an EQRCodeReader object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EQRCodeReader(
 )
void EQRCodeReader(
 const EQRCodeReader& other
 )
```

### Parameters

*other*

-

# EQRCodeReader::GetFilterOutUnreliablyDecodedQRCodes

# EQRCodeReader::SetFilterOutUnreliablyDecodedQRCodes

Activate or deactivate the filtering of unreliably decoded QR codes.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool GetFilterOutUnreliablyDecodedQRCodes()
void SetFilterOutUnreliablyDecodedQRCodes(bool filter)
```

### Remarks

By default, the QR code reader does not return unreliably decoded QR codes.

# EQRCodeReader::GetForegroundDetectionThreshold

# EQRCodeReader::SetForegroundDetectionThreshold

Foreground detection threshold. This parameter determines how many grayscale-values a pixel should deviate from it's local background to be considered part of the foreground.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetForegroundDetectionThreshold()
void SetForegroundDetectionThreshold(int offset)
```

### Remarks

The default value for this parameter is 10.

# EQRCodeReader::GetMaximumVersion

# EQRCodeReader::SetMaximumVersion

Maximum version of QR codes to be searched for.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetMaximumVersion()
void SetMaximumVersion(OEV_UINT32 version)
```

### Remarks

This parameter value ranges from **1** to **40**. Default value: **40**.

# EQRCodeReader::GetMinimumIsotropy

# EQRCodeReader::SetMinimumIsotropy

QR code minimum isotropy.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetMinimumIsotropy()
void SetMinimumIsotropy(float isotropy)
```

### Remarks

The isotropy of a QR code is defined as its short side divided by its long side. This parameter value ranges from 0 to 1. default value: 0.8.

# EQRCodeReader::GetMinimumScore

# EQRCodeReader::SetMinimumScore

Minimum pattern finder score that must be reached to consider that a finder pattern has been found.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetMinimumScore()
void SetMinimumScore(float minScore)
```

### Remarks

The pattern finder score is based on a normalized correlation with a perfect finder pattern model. A perfect match with the model would return a score of 1. This parameter value ranges from **0** to **1**. Default value: **0.65**.

# EQRCodeReader::GetMinimumVersion

# EQRCodeReader::SetMinimumVersion

Minimum version of QR codes to be searched for.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetMinimumVersion()
void SetMinimumVersion(OEV_UINT32 version)
```

### Remarks

This parameter value ranges from **1** to **40**. Default value: **1**.

# EQRCodeReader::GetPerspectiveMode

# EQRCodeReader::SetPerspectiveMode

Sets the perspective mode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EQRCodePerspectiveMode GetPerspectiveMode()
void SetPerspectiveMode(Euresys::Open_eVision_2_10::EQRCodePerspectiveMode mode)
```

### Remarks

The default value is Basic. This setting is deprecated as per Open eVision release 2.0, setting this variable will have no effect. The EQRCodePerspectiveMode_Basic option has been superceded by EQRDetectionMethod_GradientLegacy, the EQRCodePerspectiveMode_Improved option has been superceded by EQRDetectionMethod_PerspectiveLegacy.

# EQRCodeReader::Read

Detects and decodes all the QR codes in the search field.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::vector<Euresys::Open_eVision_2_10::EQRCode> Read(
 )
```

# EQRCodeReader::GetScanPrecision

# EQRCodeReader::SetScanPrecision

Precision of the QR code reader when scanning the search field.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EQRCodeScanPrecision GetScanPrecision()
void SetScanPrecision(Euresys::Open_eVision_2_10::EQRCodeScanPrecision precision)
```

## Remarks

The default value is **EQRCodeScanPrecision_Automatic**.

# EQRCodeReader::GetSearchedModels

# EQRCodeReader::SetSearchedModels

QR code models to be searched for.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::vector<Euresys::Open_eVision_2_10::EQRCodeModel> GetSearchedModels()

void SetSearchedModels(const std::vector<Euresys::Open_eVision_2_10::EQRCodeModel>&
models)
```

### Remarks

By default, the QR code reader searches for all models of QR codes.

# EQRCodeReader::SetSearchField

Search field for the QR code reader.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetSearchField(EROIBW8& field)
```

# EQRCodeReader::GetTimeOut

# EQRCodeReader::SetTimeOut

Time-out for the EQRCodeReader::Detect, EQRCodeReader::Decode and EQRCodeReader::Read methods.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT64 GetTimeOut()
void SetTimeOut(OEV_UINT64 value)
```

### Remarks

If the processing time of one of these functions becomes longer than the set time-out, the processing is stopped and an exception is thrown. In that case, the error code of the exception is EError_TimeoutReached. The time-out is set in microseconds. This time-out is not a real time-out. The processing is stopped as soon as possible after the time-out has been reached. This means that the time elapsed effectively in the method can be greater than the time-out in itself.

# 4.157. EQuadrangle Class

This class represents a polygon with four corners (with sub-pixel accuracy).

### Remarks

A quadrangle especially arises when representing the corners of a rotated bounding box.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Draw | Draws the quadrangle, by drawing lines between its corners. |
| DrawWithCurrentPen | Draws the quadrangle, by drawing lines between its corners. |
| EQuadrangle | Constructs a EQuadrangle. The default constructor initializes all points to 0. |
| GetGravityCenter | Returns the gravity center of the quadrangle. |

| | |
|---|---|
| GetPoint | Returns the coordinate of a given corner of the quadrangle. |
| GetSideAngle | Returns the angle of a given side of the quadrangle. |
| IsInside | Tests if a point is inside the quadrangle. |
| operator= | Assignment operator. |
| OverLaps | Tests if the quadrangles overlap. |
| SetPoint | Sets the coordinates of a given corner of the quadrangle. |

E

Q

# uadrangle::Draw

Draws the quadrangle, by drawing lines between its corners.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )

void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
```

## Parameters

*graphicContext*

Graphic context on which to draw.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*drawDiagonals*

Specifies whether or not lines are to be drawn between the 1st and 3rd corners, as well as between the 2nd and 4th corners.

*color*

The color in which to draw the overlay.

### Remarks

Drawing is done in the device context associated to the desired window.

# EQuadrangle::DrawWithCurrentPen

Draws the quadrangle, by drawing lines between its corners.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 BOOL drawDiagonals
 )
```

### Parameters

*graphicContext*

Graphic context on which to draw.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*drawDiagonals*

Specifies whether or not lines are to be drawn between the 1st and 3rd corners, as well as between the 2nd and 4th corners.

### Remarks

Drawing is done in the device context associated to the desired window.

# EQuadrangle::EQuadrangle

Constructs a EQuadrangle. The default constructor initializes all points to 0.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EQuadrangle(
 )
void EQuadrangle(
 const EQuadrangle& other
 )
```

### Parameters

*other*

The source EQuadrangle.

# EQuadrangle::GetPoint

Returns the coordinate of a given corner of the quadrangle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
EPoint GetPoint(
 const OEV_UINT32 index
 )
```

### Parameters

*index*

The index of the corner of interest (must lie in the range between 0 and 3, inclusive).

# EQuadrangle::GetSideAngle

Returns the angle of a given side of the quadrangle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSideAngle(
 OEV_UINT32 sideIndex
 )
```

### Parameters

*sideIndex*

-

# EQuadrangle::GetGravityCenter

Returns the gravity center of the quadrangle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetGravityCenter() const
```

# EQuadrangle::IsInside

Tests if a point is inside the quadrangle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool IsInside(
 const float& x,
 const float& y,
 bool includeEdges
 )
```

### Parameters

*x*

-

*y*

-

*includeEdges*

-

# EQuadrangle::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EQuadrangle& operator=(
 const EQuadrangle& other
 )
```

### Parameters

*other*

The source EQuadrangle.

# EQuadrangle::OverLaps

Tests if the quadrangles overlap.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool OverLaps(
 const EQuadrangle& other
 )
```

### Parameters

*other*

-

# EQuadrangle::SetPoint

Sets the coordinates of a given corner of the quadrangle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetPoint(
 const OEV_UINT32& index,
 const EPoint& location
 )
```

### Parameters

*index*

The index of the corner of interest (must lie in the range between 0 and 3, inclusive).

*location*

The coordinate.

# 4.158. EQuadrilateral Class

Represents a quadrilateral.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EQuadrilateral | Creates an EQuadrilateral object. |
| GetCorners | The corners of the quadrilateral. |
| operator= | - |

# EQuadrilateral::GetCorners

The corners of the quadrilateral.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::vector<Euresys::Open_eVision_2_10::EPoint> GetCorners()
```

## Remarks

The corners are returned in the following order: bottom left, top left, top right, bottom right.

# EQuadrilateral::EQuadrilateral

Creates an EQuadrilateral object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EQuadrilateral(
 )

void EQuadrilateral(
 const std::vector<Euresys::Open_eVision_2_10::EPoint>& corners
 )

void EQuadrilateral(
 const EQuadrilateral& other
 )
```

## Parameters

*corners*

-

*other*
-

# EQuadrilateral::operator=

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EQuadrilateral& operator=(
 const EQuadrilateral& other
 )
```

### Parameters

*other*
-

# 4.159. ERandomDecimator Class

Decimation of an EPointCloud.
The random decimator decimates a point cloud by copying a specified number of points, randomly selected, to a new point cloud.

**Base Class:** EDecimator

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| Decimate | Decimates a given EPointCloud and writes the result into a new point cloud. |

| | |
|---|---|
| ERandomDecimator | Creates an ERandomDecimator object. If the required number of points (after decimation) is not specified, the default value is **10000**. |
| GetNumberOfPoints | Sets/gets the parameter "number of points" (number of points after decimation). |
| operator= | Assignment operator. |
| Serialize | Serializer |
| SetNumberOfPoints | Sets/gets the parameter "number of points" (number of points after decimation). |

E

R

# andomDecimator::Decimate

Decimates a given EPointCloud and writes the result into a new point cloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Decimate(
 const EPointCloud& cloudIn,
 EPointCloud& cloudOut
 )
```

## Parameters

*cloudIn*

The input point cloud.

*cloudOut*

The output point cloud.

### Remarks

The input point cloud 'cloudIn' should be different from the output point cloud 'cloudOut'. If not a exception will be thrown.

# ERandomDecimator::ERandomDecimator

Creates an ERandomDecimator object. If the required number of points (after decimation) is not specified, the default value is **10000**.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ERandomDecimator(
 )

void ERandomDecimator(
 int numberOfPoints
 )

void ERandomDecimator(
 const ERandomDecimator& other
 )
```

### Parameters

*numberOfPoints*

Number of points after decimation.

*other*

Reference to the ERandomDecimator object used for the initialization.

# ERandomDecimator::GetNumberOfPoints

# ERandomDecimator::SetNumberOfPoints

Sets/gets the parameter "number of points" (number of points after decimation).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
int GetNumberOfPoints() const

void SetNumberOfPoints(int numberOfPoints)
```

# ERandomDecimator::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
ERandomDecimator& operator=(
 const ERandomDecimator& other
 )
```

## Parameters

*other*

The ERandomDecimator object that should be copied.

# ERandomDecimator::Serialize

Serializer

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or written to.

# 4.160. ERectangle Class

Represents a model of a rectangle in EasyGauge.

**Base Class:** EFrame

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| CopyTo | Copies all the data of the current ERectangle object into another ERectangle object, and returns it. |
| ERectangle | Constructs a ERectangle |

| | |
|---|---|
| GetCorners | Retrieves the coordinates of each corner of a ERectangle object. |
| GetEdges | Retrieves each edge of a ERectangle object. |
| GetMidEdges | Retrieves the center coordinates of each edge of a ERectangle object. |
| GetPoint | Returns the coordinates of a particular point, specified by its location in the ERectangle area. |
| GetSizeX | X size of the ERectangle |
| GetSizeY | Y size of the ERectangle |
| operator= | Copies all the data from another ERectangle object into the current ERectangle object |
| SetFromOppositeCorners | Sets the geometric parameters (center coordinates, size, and rotation angle) of an ERectangle object. |
| SetFromOriginMiddleEnd | DEPRECATED (you should use ERectangle::SetFromThreeCorners): Sets the geometric parameters (center coordinates, size, and rotation angle) of an ERectangle object. |
| SetFromThreeCorners | Sets the geometric parameters (center coordinates, size, and rotation angle) of an ERectangle object. |

| | |
|---|---|
| SetFromTwoPoints | DEPRECATED (you should use ERectangle::SetFromOppositeCorners): Sets the geometric parameters (center coordinates, size, and rotation angle) of an ERectangle object. |
| SetSize | Sets the size of a ERectangle object. |

E

R

# ectangle::CopyTo

Copies all the data of the current ERectangle object into another ERectangle object, and returns it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERectangle* CopyTo(
 ERectangle* other
 )
```

## Parameters

*other*

Pointer to the ERectangle object in which the current ERectangle object data have to be copied.

## Remarks

In case of a **NULL** pointer, a new ERectangle object will be created and returned.

# ERectangle::ERectangle

Constructs a ERectangle

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ERectangle(
 )
void ERectangle(
 const EPoint& center,
 float sizeX,
 float sizeY,
 float angle
 )
void ERectangle(
 const EPoint& origin,
 const EPoint& end
 )
void ERectangle(
 const EPoint& origin,
 const EPoint& middle,
 const EPoint& end
 )
void ERectangle(
 const ERectangle& other
 )
```

## Parameters

*center*

Center coordinates of the rectangle at its nominal position. The default value is **(0,0)**.

*sizeX*

Nominal size X/Y of the rectangle. Both default values are **100**.

*sizeY*

Nominal size X/Y of the rectangle. Both default values are **100**.

*angle*

Nominal rotation angle of the rectangle. The default value is **0**.

*origin*

Upper left point coordinates of the rectangle.

*end*

Lower right point coordinates of the rectangle.

*middle*

A third corner point coordinates.

*other*

Another ERectangle object to be copied in the new ERectangle object.

# ERectangle::GetCorners

Retrieves the coordinates of each corner of a ERectangle object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetCorners(
 EPoint& xy,
 EPoint& XXy,
 EPoint& xYY,
 EPoint& XXYY
 )
```

## Parameters

*xy*

Coordinates of the lower leftmost corner of the ERectangle object.

*XXy*

Coordinates of the lower rightmost corner of the ERectangle object.

*xYY*

Coordinates of the upper leftmost corner of the ERectangle object.

*XXYY*

Coordinates of the upper rightmost corner of the ERectangle object.

# ERectangle::GetEdges

Retrieves each edge of a ERectangle object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetEdges(
 ELine& x,
 ELine& XX,
 ELine& y,
 ELine& YY
 )
```

### Parameters

*x*

Leftmost edge of the ERectangle object.

*XX*

Rightmost edge of the ERectangle object.

*y*

Lower edge of the ERectangle object.

*YY*

Upper edge of the ERectangle object.

# ERectangle::GetMidEdges

Retrieves the center coordinates of each edge of a ERectangle object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetMidEdges(
 EPoint& x,
 EPoint& XX,
 EPoint& y,
 EPoint& YY
 )
```

### Parameters

*x*

Center coordinates of the leftmost edge of the ERectangle object.

*XX*

Center coordinates of the rightmost edge of the ERectangle object.

*Y*

Center coordinates of the lower edge of the ERectangle object.

*YY*

Center coordinates of the upper edge of the ERectangle object.

# ERectangle::GetPoint

Returns the coordinates of a particular point, specified by its location in the ERectangle area.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetPoint(
 float fractionX,
 float fractionY
 )
```

## Parameters

*fractionX*

Point location expressed as a fraction of the ERectangle vertical edges (range -1, +1).

*fractionY*

Point location expressed as a fraction of the ERectangle horizontal edges (range -1, +1).

# ERectangle::operator=

Copies all the data from another ERectangle object into the current ERectangle object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERectangle& operator=(
 const ERectangle& other
 )
```

### Parameters

*other*

ERectangle object to be copied

# ERectangle::SetFromOppositeCorners

Sets the geometric parameters (center coordinates, size, and rotation angle) of an ERectangle object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromOppositeCorners(
 const EPoint& origin,
 const EPoint& end
 )
```

### Parameters

*origin*

Upper left point coordinates of the rectangle.

*end*

Lower right point coordinates of the rectangle.

# ERectangle::SetFromOriginMiddleEnd

DEPRECATED (you should use ERectangle::SetFromThreeCorners): Sets the geometric parameters (center coordinates, size, and rotation angle) of an ERectangle object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetFromOriginMiddleEnd(
 const EPoint& origin,
 const EPoint& middle,
 const EPoint& end
 )
```

### Parameters

*origin*

Upper left point coordinates of the rectangle.

*middle*

A third corner point coordinates.

*end*

Lower right point coordinates of the rectangle.

# ERectangle::SetFromThreeCorners

Sets the geometric parameters (center coordinates, size, and rotation angle) of an ERectangle object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetFromThreeCorners(
 const EPoint& origin,
 const EPoint& middle,
 const EPoint& end
 )
```

### Parameters

*origin*

Upper left point coordinates of the rectangle.

*middle*

A third corner point coordinates.

*end*

Lower right point coordinates of the rectangle.

# ERectangle::SetFromTwoPoints

DEPRECATED (you should use ERectangle::SetFromOppositeCorners): Sets the geometric parameters (center coordinates, size, and rotation angle) of an ERectangle object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetFromTwoPoints(
 const EPoint& origin,
 const EPoint& end
 )
```

### Parameters

*origin*

Upper left point coordinates of the rectangle.

*end*

Lower right point coordinates of the rectangle.

# ERectangle::SetSize

Sets the size of a ERectangle object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetSize(
 float sizeX,
 float sizeY
 )
```

### Parameters

*sizeX*

Nominal size X of the ERectangle object. Default values is **100**.

*sizeY*

Nominal size Y of the ERectangle object. Default values is **100**.

### Remarks

A ERectangle object is fully defined knowing its nominal position (given by the coordinates of its center), its nominal size, its rotation angle and its outline tolerance. By default, the width and height values are **100**, which means 100 pixels when the field of view is not calibrated and 100 "units" in case of a calibrated field of view.

# ERectangle::GetSizeX

X size of the ERectangle

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSizeX() const
```

# ERectangle::GetSizeY

Y size of the ERectangle

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSizeY() const
```

# 4.161. ERectangleGauge Class

Manages a rectangle fitting gauge.

**Base Class:** ERectangleShape
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddSkipRange | Adds an item to the set of skip ranges and returns the index of the newly added range. |
| CopyTo | Copies all the data of the current ERectangleGauge object into another ERectangleGauge object, and returns it. |
| DisableInnerFiltering | Disables inner sampled point filtering. |

| | |
|---|---|
| Drag | Moves a handle to a new position and updates the position parameters of the gauge. |
| Draw | Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode. |
| DrawWithCurrentPen | Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode. |
| ERectangleGauge | Constructs a rectangle measurement context. |
| GetActiveEdges | Active edges as defined in EDragHandle. |
| GetAverageDistance | Average distance between the sampled points and the fitted model. |
| GetFilteringThreshold | Relative filtering threshold, that is the fraction of the average distance between the sampled points and the fitted model above which a point is filtered out. |
| GetHVConstraint | Status of the restriction on the orientation of the point location gauge or model fitting sample paths. |
| GetInnerFilteringEnabled | Getter method for the **GetInnerFilteringEnabled** property. This property is the flag indicating if the inner sampled point filtering is enabled (**TRUE**). |

| | |
|---|---|
| GetInnerFilteringThreshold | Sampled point inner filtering threshold. |
| GetKnownAngle | Flag indicating whether the rotation angle of the rectangle to be fitted is known or not. |
| GetMeasuredPoint | Returns the coordinates of a sample point, measured along one of the sample paths of the gauge. |
| GetMeasuredRectangle | Information pertaining to the fitted rectangle. |
| GetMinAmplitude | Offset added to the **Threshold** when a peak is to be detected. |
| GetMinArea | Minimum area value. |
| GetMinNumFitSamples | Returns the minimum number of samples required for fitting on each side of the shape. |
| GetNumFilteringPasses | Number of filtering passes for a model fitting operation. |
| GetNumSamples | Number of sampled points during the model fitting operation. |
| GetNumSamplesx | Number of sampled points found on edge x during the measure operation. |
| GetNumSamplesX | Number of sampled points found on edge X during the measure operation. |

| | |
|---|---|
| GetNumSamplesy | Number of sampled points found on edge y during the measure operation. |
| GetNumSamplesY | Number of sampled points found on edge Y during the measure operation. |
| GetNumSkipRanges | Number of skip ranges in the gauge after a call to ERectangleGauge::AddSkipRange. |
| GetNumValidSamples | Number of valid sample points remaining after a model fitting operation. |
| GetRectangularSamplingArea | Flag indicating whether the sampling area remains a rectangle when rotated, instead of becoming a parallelogram. |
| GetSamplex | Allows to retrieve information on the samples found along the x edge. |
| GetSampleX | Allows to retrieve information on the samples found along the X edge. |
| GetSampley | Allows to retrieve information on the samples found along the y edge. |
| GetSampleY | Allows to retrieve information on the samples found along the Y edge. |

| | |
|---|---|
| GetSamplingStep | Approximate distance between sampled points during a model fitting operation. |
| GetSkipRange | Allows to retrieve the start and end values of the skip range corresponding to the given index, if it is valid (i.e. if it has previously been created by a call to the ERectangleGauge::AddSkipRange method. |
| GetSmoothing | Number of pixels used for the low-pass filtering operation. |
| GetThickness | Number of parallel segments used to extract the data profile. |
| GetThreshold | Threshold level used to delimit significant peaks in the data profile. |
| GetTolerance | Searching area half thickness of the rectangle fitting gauge. |
| GetTransitionChoice | Transition choice. |
| GetTransitionIndex | Index (from **0** on) of the transition to be retained when the transition choice parameter is set to ETransitionChoice_NthFromBegin or ETransitionChoice_NthFromEnd. |
| GetTransitionType | Transition type. |
| GetType | Shape type. |

| | |
|---|---|
| GetValid | Flag indicating if at least one valid transition has been found. |
| HitTest | Checks whether the cursor is positioned over a handle (**TRUE**) or not (**FALSE**). |
| Measure | Triggers the point location or the model fitting operation. |
| MeasureSample | Computes the sample points along the sample path whose index in the list is given by the **pathIndex** parameter. |
| MeasureWithoutFitting | Triggers the point location without rectangle fitting operation. |
| operator= | Copies all the data from another ERectangleGauge object into the current ERectangleGauge object |
| Plot | Draws the profile that is crossed by one of the sample paths of the gauge, as defined by EPlotItem. |
| PlotWithCurrentPen | Draws the profile that is crossed by one of the sample paths of the gauge, as defined by EPlotItem. |
| Process | Triggers the process pertaining to a shape or gauge and all the daughter gauges attach to it. |
| RemoveAllSkipRanges | Removes all the skip ranges previously created by a call to ERectangleGauge::AddSkipRange. |

| | |
|---|---|
| RemoveSkipRange | After a call to ERectangleGauge::AddSkipRange, removes the skip range with the given index. |
| SetActive | Sets the flag indicating whether the gauge is active or not. |
| SetActiveEdges | Active edges as defined in EDragHandle. |
| SetFilteringThreshold | Relative filtering threshold, that is the fraction of the average distance between the sampled points and the fitted model above which a point is filtered out. |
| SetHVConstraint | Status of the restriction on the orientation of the point location gauge or model fitting sample paths. |
| SetInnerFilteringThreshold | Sampled point inner filtering threshold. |
| SetKnownAngle | Flag indicating whether the rotation angle of the rectangle to be fitted is known or not. |
| SetMinAmplitude | Offset added to the **Threshold** when a peak is to be detected. |
| SetMinArea | Minimum area value. |
| SetMinNumFitSamples | Sets the minimum number of samples required for fitting on each side of the shape. |

| | |
|---|---|
| SetNumFilteringPasses | Number of filtering passes for a model fitting operation. |
| SetRectangularSamplingArea | Flag indicating whether the sampling area remains a rectangle when rotated, instead of becoming a parallelogram. |
| SetSamplingStep | Approximate distance between sampled points during a model fitting operation. |
| SetSmoothing | Number of pixels used for the low-pass filtering operation. |
| SetThickness | Number of parallel segments used to extract the data profile. |
| SetThreshold | Threshold level used to delimit significant peaks in the data profile. |
| SetTolerance | Searching area half thickness of the rectangle fitting gauge. |
| SetTransitionChoice | Transition choice. |
| SetTransitionIndex | Index (from **0** on) of the transition to be retained when the transition choice parameter is set to ETransitionChoice_NthFromBegin or ETransitionChoice_NthFromEnd. |
| SetTransitionType | Transition type. |

# ERectangleGauge::SetActive

Sets the flag indicating whether the gauge is active or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetActive(BOOL active)
```

## Remarks

When complex gauging is required, several gauges can be grouped together. Applying ERectangleGauge::Process to the mother gauge or shape triggers the measurement of the whole. Only the active gauges will participate in the process. By default, the gauge is active (**TRUE**).

# ERectangleGauge::GetActiveEdges

# ERectangleGauge::SetActiveEdges

Active edges as defined in EDragHandle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetActiveEdges()
void SetActiveEdges(OEV_UINT32 un32ActiveEdges)
```

## Remarks

In the case of a rectangle fitting gauge, each edge can have its own transition detection parameters. Updating the transition parameters only affect the current active edges. By default, all edges are active.

# ERectangleGauge::AddSkipRange

Adds an item to the set of skip ranges and returns the index of the newly added range.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 AddSkipRange(
 const OEV_UINT32 start,
 const OEV_UINT32 end
 )
```

### Parameters

*start*

Beginning of the skip range.

*end*

End of the skip range.

### Remarks

The samples indices between start and end (including the boundaries) will be discarded during the measurement process.
The ERectangleGauge::AddSkipRange method allows to define skip ranges in an ERectangleGauge. This means that, at measure time, samples belonging to these ranges will not be taken into account.
A sample may belong to more than one skip range; to be discarded, a sample has to pertain to at least one range.
Moreover, the skip ranges are allowed to overlap one another.
The range is allowed to be reversed (i.e. end is not required to be greater than start).
Also, start and end are not required to reference valid indices at the time of the call (i.e. the range may lie outside of the current return value for ERectangleGauge::NumSamples).

# ERectangleGauge::GetAverageDistance

Average distance between the sampled points and the fitted model.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetAverageDistance()
```

### Remarks

Irrelevant in case of a point location operation.

# ERectangleGauge::CopyTo

Copies all the data of the current ERectangleGauge object into another ERectangleGauge object, and returns it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERectangleGauge* CopyTo(
 ERectangleGauge* other,
 BOOL recursive
 )
```

### Parameters

*other*

Pointer to the ERectangleGauge object in which the current ERectangleGauge object data have to be copied.

*recursive*

**TRUE** if the children gauges have to be copied as well, **FALSE** otherwise.

### Remarks

In case of a **NULL** pointer, a new ERectangleGauge object will be created and returned.

# ERectangleGauge::DisableInnerFiltering

Disables inner sampled point filtering.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DisableInnerFiltering(
 )
```

### Remarks

The inner sampled point filtering is activated as soon as the corresponding ERectangleGauge::InnerFilteringThreshold is set.

# ERectangleGauge::Drag

Moves a handle to a new position and updates the position parameters of the gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Cursor current X coordinate.

*y*

Cursor current Y coordinate.

# ERectangleGauge::Draw

Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )

void Draw(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )

void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

*color*

The color in which to draw the overlay.

# ERectangleGauge::DrawWithCurrentPen

Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

# ERectangleGauge::ERectangleGauge

Constructs a rectangle measurement context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ERectangleGauge(
 )
void ERectangleGauge(
 const ERectangleGauge& other
 )
```

### Parameters

*other*

Another ERectangleGauge object to be copied in the new ERectangleGauge object.

### Remarks

With the default constructor, all the parameters are initialized to their respective default values.
With the copy constructor, the constructed rectangle measurement context is based on a pre-existing ERectangleGauge object. The gauge children are also copied. Hierarchy copying through a copy constructor is always recursive. To disable this recursion, use instead the ERectangleGauge::CopyTo method.

# ERectangleGauge::GetFilteringThreshold

# ERectangleGauge::SetFilteringThreshold

Relative filtering threshold, that is the fraction of the average distance between the sampled points and the fitted model above which a point is filtered out.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetFilteringThreshold()
void SetFilteringThreshold(float f32FilteringThreshold)
```

### Remarks

Irrelevant in case of a point location operation.
During a model fitting operation, the "filtering" process can be invoked to remove outliers, i.e. points that were located significantly far away from the fitted model (so that their position is dubious).

# ERectangleGauge::GetMeasuredPoint

Returns the coordinates of a sample point, measured along one of the sample paths of the gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetMeasuredPoint(
 OEV_UINT32 index
 )
```

### Parameters

*index*

This argument must be left unchanged from its default value, i.e. **~0** (**= 0xFFFFFFFF**).

### Remarks

These coordinates pertain to the world space; they are expressed in the reference frame to which the current ERectangleGauge object belongs. The gauging process uses a list of sample points to find the shape position and size that best fit a given image. These sample points are measured along the sample paths defined by the gauge geometry. ERectangleGauge::GetMeasuredPoint returns the coordinates of the sample point that meets the following two requirements: 1. It lies on the sample path inspected with the last call to ERectangleGauge::MeasureSample, and 1. Among all the sample points along the latter sample path, it is the one selected by the ERectangleGauge::TransitionChoice property.

**Note.** For this method to succeed, it is necessary to previously call ERectangleGauge::MeasureSample.

# ERectangleGauge::GetMinNumFitSamples

Returns the minimum number of samples required for fitting on each side of the shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetMinNumFitSamples(
 OEV_INT32& side0,
 OEV_INT32& side1,
 OEV_INT32& side2,
 OEV_INT32& side3
 )
```

### Parameters

*side0*

Minimum number of samples on the top side of the rectangle.

*side1*

Minimum number of samples on the left side of the rectangle.

*side2*

Minimum number of samples on the bottom side of the rectangle.

*side3*

Minimum number of samples on the right side of the rectangle.

### Remarks

Irrelevant in case of a point location operation.

# ERectangleGauge::GetSampleX

Allows to retrieve information on the samples found along the X edge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetSampleX(
 EPoint& pt,
 OEV_UINT32 index
 )
void GetSampleX(
 ESamplePoint& sp,
 OEV_UINT32 index
 )
BOOL GetSampleX(
 EPeak& pk,
 OEV_UINT32 index
 )
```

### Parameters

*pt*

EPoint structure that will receive the sample position.

*index*

The sample index

*sp*

ESamplePoint structure that will receive the sample position.

*pk*

EPeak structure that will contain the sample peak properties.

### Remarks

The method provides the sample point corresponding to the supplied index.
The returned value is true when the sample is valid, and false otherwise.

# ERectangleGauge::GetSampleX

Allows to retrieve information on the samples found along the X edge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetSampleX(
 EPoint& pt,
 OEV_UINT32 index
 )

void GetSampleX(
 ESamplePoint& sp,
 OEV_UINT32 index
 )

BOOL GetSampleX(
 EPeak& pk,
 OEV_UINT32 index
 )
```

### Parameters

*pt*

EPoint structure that will receive the sample position.

*index*

The sample index

*sp*

ESamplePoint structure that will receive the sample position.

*pk*

EPeak structure that will contain the sample peak properties.

### Remarks

The method provides the sample point corresponding to the supplied index.
The returned value is true when the sample is valid, and false otherwise.

# ERectangleGauge::GetSampleY

Allows to retrieve information on the samples found along the Y edge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetSampleY(
 EPoint& pt,
 OEV_UINT32 index
 )
void GetSampleY(
 ESamplePoint& sp,
 OEV_UINT32 index
 )
BOOL GetSampleY(
 EPeak& pk,
 OEV_UINT32 index
 )
```

### Parameters

*pt*

EPoint structure that will receive the sample position.

*index*

The sample index

*sp*

ESamplePoint structure that will receive the sample position.

*pk*

EPeak structure that will contain the sample peak properties.

### Remarks

The method provides the sample point corresponding to the supplied index.
The returned value is true when the sample is valid, and false otherwise.

# ERectangleGauge::GetSampleY

Allows to retrieve information on the samples found along the Y edge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetSampleY(
 EPoint& pt,
 OEV_UINT32 index
 )
void GetSampleY(
 ESamplePoint& sp,
 OEV_UINT32 index
 )
BOOL GetSampleY(
 EPeak& pk,
 OEV_UINT32 index
 )
```

## Parameters

*pt*

EPoint structure that will receive the sample position.

*index*

The sample index

*sp*

ESamplePoint structure that will receive the sample position.

*pk*

EPeak structure that will contain the sample peak properties.

## Remarks

The method provides the sample point corresponding to the supplied index.
The returned value is true when the sample is valid, and false otherwise.

# ERectangleGauge::GetSkipRange

Allows to retrieve the start and end values of the skip range corresponding to the given index, if it is valid (i.e. if it has previously been created by a call to the ERectangleGauge::AddSkipRange method.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetSkipRange(
 const OEV_UINT32 index,
 OEV_UINT32& start,
 OEV_UINT32& end
 )
```

### Parameters

*index*

Index of the skip range.

*start*

Beginning of the skip range.

*end*

End of the skip range.

### Remarks

Start is guaranteed to be smaller or equal to end.

# ERectangleGauge::HitTest

Checks whether the cursor is positioned over a handle (**TRUE**) or not (**FALSE**).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL HitTest(
 BOOL daughters
 )
```

### Parameters

*daughters*

**TRUE** if the daughters gauges handles have to be considered as well.

# ERectangleGauge::GetHVConstraint

# ERectangleGauge::SetHVConstraint

Status of the restriction on the orientation of the point location gauge or model fitting sample paths.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetHVConstraint()
void SetHVConstraint(BOOL bHVConstraint)
```

### Remarks

*Sample paths* are the point location gauges placed along the model to be fitted.

# ERectangleGauge::GetInnerFilteringEnabled

Getter method for the **GetInnerFilteringEnabled** property. This property is the flag indicating if the inner sampled point filtering is enabled (**TRUE**).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetInnerFilteringEnabled()
```

### Remarks

The inner sampled point filtering is activated as soon as the corresponding threshold is set, getting the ERectangleGauge**.InnerFilteringThreshold** property. To disable inner filtering, use the **DisableInnerFiltering** method.

# ERectangleGauge::GetInnerFilteringThreshold

# ERectangleGauge::SetInnerFilteringThreshold

Sampled point inner filtering threshold.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetInnerFilteringThreshold()
void SetInnerFilteringThreshold(float f32InnerFilteringThreshold)
```

### Remarks

If inner filtering is enabled, the sampled points that have been found inside the measured rectangle are filtered in regard of their distance to it. If this distance is greater than the threshold, the sampled point is set as invalid, and removed from the measure. This distance is in physical units.
The inner sampled point filtering is activated as soon as the corresponding threshold is set. To disable inner filtering, use the **DisableInnerFiltering** method.

# ERectangleGauge::GetKnownAngle

# ERectangleGauge::SetKnownAngle

Flag indicating whether the rotation angle of the rectangle to be fitted is known or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetKnownAngle()
void SetKnownAngle(BOOL bKnownAngle)
```

### Remarks

A rectangle model to be fitted may have a well-known orientation. It is possible to impose the value of this rotation angle, thus removing one degree of freedom. The rectangle fitting gauge orientation is set by means of the ERectangleGauge**.Angle** property.
The sign of the rotation angle depends whether the field of view is calibrated or not.
 * When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value.
 * When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# ERectangleGauge::Measure

Triggers the point location or the model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void Measure(
 EROIBW8* sourceImage
 )
```

### Parameters

*sourceImage*

Pointer to the source image.

### Remarks

When this method is called, and if not enough valid points are detected, then the method returns directly, and the measured gauge is set to the nominal parameters.

# ERectangleGauge::GetMeasuredRectangle

Information pertaining to the fitted rectangle.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
ERectangle GetMeasuredRectangle()
```

# ERectangleGauge::MeasureSample

Computes the sample points along the sample path whose index in the list is given by the **pathIndex** parameter.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void MeasureSample(
 EROIBW8* sourceImage,
 OEV_UINT32 pathIndex
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*pathIndex*

Sample path index.

### Remarks

This method stores its results into a temporary variable inside the ERectangleGauge object.

# ERectangleGauge::MeasureWithoutFitting

Triggers the point location without rectangle fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void MeasureWithoutFitting(
 EROIBW8* sourceImage
 )
```

### Parameters

*sourceImage*

Source image.

### Remarks

This method performs the actual measurement for each transition, but does not perform the rectangle fitting. This means that individual samples will be available for each edges through the ERectangleGauge::GetSamplex (Edge x), ERectangleGauge::GetSampley (Edge y), ERectangleGauge::GetSampleX (Edge X), ERectangleGauge::GetSampleY (Edge Y) methods, but the gauge position will not be changed.
Please note that the filtering will not be performed in this method, since it relies upon the fitting process. The filtering parameters will thus be unused.

# ERectangleGauge::GetMinAmplitude

# ERectangleGauge::SetMinAmplitude

Offset added to the **Threshold** when a peak is to be detected.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetMinAmplitude()
void SetMinAmplitude(OEV_UINT32 un32MinAmplitude)
```

### Remarks

When analyzing a derivative profile, a peak is made up of consecutive pixel values above **Threshold**. To detect weak [strong] transitions, lower [raise] the **Threshold** value.
To avoid interference of noise, an additional parameter is provided. The **MinAmplitude** parameter is an offset added to **Threshold** when a peak is to be detected.
When the pixel values of the derivative profile do not reach **Threshold + MinAmplitude**, the peak is not taken into account. Anyway, when a peak is taken into account, all the pixels with values above **Threshold** are considered (for more accuracy). Setting the **MinAmplitude** value to **0** merely cancels its effect.

# ERectangleGauge::GetMinArea

# ERectangleGauge::SetMinArea

Minimum area value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetMinArea()
void SetMinArea(OEV_UINT32 un32MinArea)
```

## Remarks

A transition is detected if its derivative peak reaches **Threshold** + **MinAmplitude** value, and then declared valid if the area between the peak curve and the horizontal at level **Threshold** reaches the **MinArea** value.

# ERectangleGauge::GetNumFilteringPasses

# ERectangleGauge::SetNumFilteringPasses

Number of filtering passes for a model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumFilteringPasses()
void SetNumFilteringPasses(OEV_UINT32 un32NumFilteringPasses)
```

### Remarks

Irrelevant in case of a point location operation.
During a filtering pass, the points that are too distant from the model are discarded.
During a model fitting operation, the "filtering" process can be invoked to remove outliers, i.e. points that were located significantly far away from the fitted model (so that their position is dubious).
By default (the number of filtering passes is 0), the outliers rejection process is disabled.

# ERectangleGauge::GetNumSamples

Number of sampled points during the model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumSamples() const
```

### Remarks

Irrelevant in case of a point location operation.
After a model fitting operation, a number of points have been fitted along the model. Among them, some are not reliable because of their **Area** value. Among the remaining ones, some are filtered out (**NumFilteringPasses**, **FilteringThreshold**).

# ERectangleGauge::GetNumSamplesX

Number of sampled points found on edge X during the measure operation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumSamplesX() const
```

# ERectangleGauge::GetNumSamplesX

Number of sampled points found on edge X during the measure operation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumSamplesX() const
```

# ERectangleGauge::GetNumSamplesY

Number of sampled points found on edge Y during the measure operation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumSamplesY() const
```

# ERectangleGauge::GetNumSamplesY

Number of sampled points found on edge Y during the measure operation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumSamplesY() const
```

# ERectangleGauge::GetNumSkipRanges

Number of skip ranges in the gauge after a call to ERectangleGauge::AddSkipRange.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumSkipRanges() const
```

# ERectangleGauge::GetNumValidSamples

Number of valid sample points remaining after a model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumValidSamples()
```

## Remarks

Irrelevant in case of a point location operation.
After a model fitting operation, a number of points have been fitted along the model. Among them, some are not reliable because of their **Area** value. Among the remaining ones, some are filtered out (**NumFilteringPasses**, **FilteringThreshold**).

# ERectangleGauge::operator=

Copies all the data from another ERectangleGauge object into the current ERectangleGauge object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERectangleGauge& operator=(
 const ERectangleGauge& other
 )
```

### Parameters

*other*

ERectangleGauge object to be copied

# ERectangleGauge::Plot

Draws the profile that is crossed by one of the sample paths of the gauge, as defined by EPlotItem.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Plot(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )
```

```
void Plot(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )
void Plot(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawItems*

Boolean combination of EPlotItem members, that indicates which items must be displayed.

*width*

Width of the plot.

*height*

Height of the plot.

*originX*

Origin point coordinates of the plot along the X axis.

*originY*

Origin point coordinates of the plot along the Y axis.

*color*

The color in which to draw the overlay.

## Remarks

The sample path that is taken into considered is the one inspected with the last call to ERectangleGauge::MeasureSample.

**Note.** For this method to succeed, it is necessary to previously call ERectangleGauge::MeasureSample.

# ERectangleGauge::PlotWithCurrentPen

Draws the profile that is crossed by one of the sample paths of the gauge, as defined by EPlotItem.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void PlotWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawItems*

Boolean combination of EPlotItem members, that indicates which items must be displayed.

*width*

Width of the plot.

*height*

Height of the plot.

*originX*

Origin point coordinates of the plot along the X axis.

*originY*

Origin point coordinates of the plot along the Y axis.

## Remarks

The sample path that is taken into considered is the one inspected with the last call to ERectangleGauge::MeasureSample.

**Note.** For this method to succeed, it is necessary to previously call ERectangleGauge::MeasureSample.

# ERectangleGauge::Process

Triggers the process pertaining to a shape or gauge and all the daughter gauges attach to it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Process(
 EROIBW8* sourceImage,
 BOOL daughters
 )
```

### Parameters

*sourceImage*

Pointer to the source image.

*daughters*

Flag indicating whether the daughters shapes inherit of the same behavior.

### Remarks

When complex gauging is required, several gauges can be grouped together. Applying **Process** to the mother gauge or shape triggers the measurement of the whole. Only the active gauges will participate in the process.

# ERectangleGauge::GetRectangularSamplingArea

# ERectangleGauge::SetRectangularSamplingArea

Flag indicating whether the sampling area remains a rectangle when rotated, instead of becoming a parallelogram.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetRectangularSamplingArea()
void SetRectangularSamplingArea(BOOL bRectangularSamplingArea)
```

### Remarks

By default, this flag is set to **TRUE**: the sampling area always remains a rectangle.
Setting this property is only useful when the thickness transition parameter is greater than 1. In fact, when thickness transition parameter is equal to 1, rectangle and parallelogram reduce to the same segment.

# ERectangleGauge::RemoveAllSkipRanges

Removes all the skip ranges previously created by a call to ERectangleGauge::AddSkipRange.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveAllSkipRanges(
 )
```

# ERectangleGauge::RemoveSkipRange

After a call to ERectangleGauge::AddSkipRange, removes the skip range with the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void RemoveSkipRange(
 const OEV_UINT32 index
 )
```

### Parameters

*index*

Index of the skip range to remove, as returned by ERectangleGauge::AddSkipRange.

# ERectangleGauge::GetSamplingStep

# ERectangleGauge::SetSamplingStep

Approximate distance between sampled points during a model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSamplingStep()
void SetSamplingStep(float f32SamplingStep)
```

### Remarks

Irrelevant in case of a point location operation.
To fit a model, a series of point location operations are performed along the model. The point location gauges spacing is given by the sampling step.
By default, the sampling step is set to **5**, which means 5 pixels when the field of view is not calibrated, and 5 physical units in case of a calibrated field of view.
Be aware that if the sampling step is too large, the number of sampled points along the model will not be sufficient enough to accurately locate it.

# ERectangleGauge::SetMinNumFitSamples

Sets the minimum number of samples required for fitting on each side of the shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetMinNumFitSamples(
 OEV_INT32 side0,
 OEV_INT32 side1,
 OEV_INT32 side2,
 OEV_INT32 side3
 )
```

## Parameters

*side0*

Minimum number of samples on the *top side* of the rectangle. The default value is **2**.

*side1*

Minimum number of samples on the *left side* of the rectangle. If this value is not specified, it is equal to **n32Side0**. The default value is **2**.

*side2*

Minimum number of samples on the *bottom side* of the rectangle. If this value is not specified, it is equal to **n32Side0**. The default value is **2**.

*side3*

Minimum number of samples on the *right side* of the rectangle. If this value is not specified, it is equal to **n32Side1**. The default value is **2**.

## Remarks

When the ERectangleGauge::Measure method is called, and if not enough valid points are detected, then the method returns directly, and the measured gauge is set to the nominal parameters.

# ERectangleGauge::GetSmoothing

# ERectangleGauge::SetSmoothing

Number of pixels used for the low-pass filtering operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetSmoothing()
void SetSmoothing(OEV_UINT32 un32Smoothing)
```

## Remarks

To reduce the effect of noise, the profile data can be low-pass filtered along the point location gauge direction.

# ERectangleGauge::GetThickness

# ERectangleGauge::SetThickness

Number of parallel segments used to extract the data profile.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetThickness()
void SetThickness(OEV_UINT32 un32Thickness)
```

### Remarks

To reduce the effect of noise and/or strengthen a transition, several parallel profiles can be accumulated.

# ERectangleGauge::GetThreshold

# ERectangleGauge::SetThreshold

Threshold level used to delimit significant peaks in the data profile.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetThreshold()
void SetThreshold(OEV_UINT32 un32Threshold)
```

### Remarks

When analyzing a derivative profile, a peak is made up of consecutive pixel values above **Threshold**. To detect weak [strong] transitions, lower [raise] the **Threshold** value.
To avoid interference of noise, an additional parameter is provided. The **MinAmplitude** parameter is an offset added to **Threshold** when a peak is to be detected.
When the pixel values of the derivative profile do not reach **Threshold + MinAmplitude**, the peak is not taken into account. Anyway, when a peak is taken into account, all the pixels with values above **Threshold** are considered (for more accuracy). Setting the **MinAmplitude** value to **0** merely cancels its effect.

# ERectangleGauge::GetTolerance

# ERectangleGauge::SetTolerance

Searching area half thickness of the rectangle fitting gauge.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetTolerance()

void SetTolerance(float f32Tolerance)
```

### Remarks

A rectangle fitting gauge is fully defined knowing its nominal position (its center coordinates), its nominal size, its rotation angle, and its outline tolerance.
By default, the searching area thickness of the rectangle fitting gauge is **20** (2x10), which means 20 pixels when the field of view is not calibrated, and 20 physical units in case of a calibrated field of view.

# ERectangleGauge::GetTransitionChoice

# ERectangleGauge::SetTransitionChoice

Transition choice.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::ETransitionChoice GetTransitionChoice()

void SetTransitionChoice(Euresys::Open_eVision_2_10::ETransitionChoice
eTransitionChoice)
```

### Remarks

Several peaks may be detected along a point location gauge. This parameter helps to select the desired transition.
In case of ETransitionChoice_NthFromBegin or ETransitionChoice_NthFromEnd transition choice, set ERectangleGauge::TransitionIndex to specify the desired transition.
By default, the selected transition corresponds to the one with the largest amplitude (ETransitionChoice_LargestAmplitude).

# ERectangleGauge::GetTransitionIndex

# ERectangleGauge::SetTransitionIndex

Index (from **0** on) of the transition to be retained when the transition choice parameter is set to NthFromBegin or NthFromEnd.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetTransitionIndex()
void SetTransitionIndex(OEV_UINT32 un32TransitionIndex)
```

## Remarks

Several peaks may be detected along a point location gauge. This parameter helps to select the desired transition. By default, the first transition is retained (the index value is **0**).

# ERectangleGauge::GetTransitionType

# ERectangleGauge::SetTransitionType

Transition type.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::ETransitionType GetTransitionType()
void SetTransitionType(Euresys::Open_eVision_2_10::ETransitionType eTransitionType)
```

### Remarks

The type of a transition tells whether it crosses increasing or decreasing gray-level values. This helps discriminate between nearby edges of an object.
By default, the searched transition type is indifferently a black to white or a white to black transition (ETransitionType_BwOrWb).

# ERectangleGauge::GetType

Shape type.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::EShapeType GetType()
```

# ERectangleGauge::GetValid

Flag indicating if at least one valid transition has been found.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetValid()
```

### Remarks

A **FALSE** value means that no measurement has been performed.
A **TRUE** value means that a transition was found along the sample path inspected with the last call to ERectangleGauge::MeasureSample, and thus a point has been measured.

# 4.162. ERectangleRegion Class

Manages a complete context for an ERegion shaped like a rectangle.

**Base Class:** ERegion

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Drag | Moves the specified handle to a new position and updates all placement parameters of the region. |
| ERectangleRegion | Constructs an ERectangleRegion context. |
| GetAngle | Angle of the region |
| GetCenter | Center of the region |
| GetHeight | Height of the region |
| GetWidth | Width of the region |
| HitTest | Detects if the cursor is placed over one of the dragging handles. |

| | |
|---|---|
| Load | Loads the ERectangleRegion. The given ESerializer must have been created for reading. |
| operator!= | Checks if this ERectangleRegion instance is not strictly equal to another |
| operator= | Assignment operator. |
| operator== | Checks if this ERectangleRegion instance is strictly equal to another |
| Rotate | Creates a new ERectangleRegion by rotating the ERectangleRegion around its center. |
| Save | Saves the ERectangleRegion. The given ESerializer must have been created for writing. |
| Scale | Creates a new ERectangleRegion by scaling the ERectangleRegion. The center of the ERectangleRegion remains at the same place. |
| SetAngle | Angle of the region |
| SetCenter | Center of the region |
| SetHeight | Height of the region |

| | |
|---|---|
| SetWidth | Width of the region |
| Translate | Creates a new ERectangleRegion by translating the ERectangleRegion. |

# ERectangleRegion::GetAngle

# ERectangleRegion::SetAngle

Angle of the region

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetAngle() const
void SetAngle(float angle)
```

# ERectangleRegion::GetCenter

# ERectangleRegion::SetCenter

Center of the region

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetCenter() const
void SetCenter(const EPoint& center)
```

# ERectangleRegion::Drag

Moves the specified handle to a new position and updates all placement parameters of the region.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 int x,
 int y,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*x*

x-coordinate of the mouse cursor.

*y*

y-coordinate of the mouse cursor.

*zoomX*

Horizontal zoom factor. By default, true scale is used.

*zoomY*

Vertical zoom factor. By default, true scale is used.

*panX*

Horizontal pan offset. By default, no pan is added.

*panY*

Vertical pan offset. By default, no pan is added.

### Remarks

If zooming and/or panning were used when drawing the region, the same values must be used with ERectangleRegion::HitTest and ERectangleRegion::Drag.

# ERectangleRegion::ERectangleRegion

Constructs an ERectangleRegion context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ERectangleRegion(
 )
void ERectangleRegion(
 float originX,
 float originY,
 float width,
 float height
 )
void ERectangleRegion(
 const EPoint& origin,
 float width,
 float height
 )
void ERectangleRegion(
 float centerX,
 float centerY,
 float width,
 float height,
 float angle
 )
```

```
void ERectangleRegion(
 const EPoint& center,
 float width,
 float height,
 float angle
 )

void ERectangleRegion(
 const ERectangle& rectangle
 )

void ERectangleRegion(
 const ERectangleRegion& other
 )
```

## Parameters

*originX*

The abcissa of the ERectangleRegion's top left corner.

*originY*

The ordinate of the ERectangleRegion's top left corner.

*width*

The width of the ERectangleRegion.

*height*

The height of the ERectangleRegion.

*origin*

The top left corner of the ERectangleRegion.

*centerX*

The abcissa of the ERectangleRegion center.

*centerY*

The ordinate of the ERectangleRegion center.

*angle*

The angle of the ERectangleRegion.

*center*

The center of the ERectangleRegion.

*rectangle*

The result of an ERectangleGauge object.

*other*

ERectangleRegion context to copy.

### Remarks

A ERectangleRegion aligned with the image axes is defined from the top left corner.
Otherwise, an oriented ERectangleRegion is defined from its center point.

# ERectangleRegion::GetHeight

# ERectangleRegion::SetHeight

Height of the region

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetHeight() const
void SetHeight(float height)
```

# ERectangleRegion::HitTest

Detects if the cursor is placed over one of the dragging handles.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::EEditionMode HitTest(
 int x,
 int y,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

### Parameters

*x*

x-coordinate of the mouse cursor.

*y*

y-coordinate of the mouse cursor.

*zoomX*

Horizontal zoom factor. By default, true scale is used.

*zoomY*

Vertical zoom factor. By default, true scale is used.

*panX*

Horizontal pan offset. By default, no pan is added.

*panY*

Vertical pan offset. By default, no pan is added.

### Remarks

Returns a handle identifier, as defined by EEditionMode.
If zooming and/or panning were used when drawing the region, the same values must be used with ERectangleRegion::HitTest and ERectangleRegion::Drag.

# ERectangleRegion::Load

Loads the ERectangleRegion. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# ERectangleRegion::operator!=

Checks if this ERectangleRegion instance is not strictly equal to another

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool operator!=(
 const ERectangleRegion& other
 )
```

### Parameters

*other*

Reference to the other ERectangleRegion instance

# ERectangleRegion::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
ERectangleRegion& operator=(
 const ERectangleRegion& other
 )
```

### Parameters

*other*

Reference to the ERectangleRegion used for the assignment

# ERectangleRegion::operator==

Checks if this ERectangleRegion instance is strictly equal to another

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
bool operator==(
 const ERectangleRegion& other
 )
```

### Parameters

*other*

Reference to the other ERectangleRegion instance

# ERectangleRegion::Rotate

Creates a new ERectangleRegion by rotating the ERectangleRegion around its center.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
ERectangleRegion Rotate(
 float angle
 )
```

### Parameters

*angle*

rotation angle

# ERectangleRegion::Save

Saves the ERectangleRegion. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# ERectangleRegion::Scale

Creates a new ERectangleRegion by scaling the ERectangleRegion. The center of the ERectangleRegion remains at the same place.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERectangleRegion Scale(
 float scale
 )
ERectangleRegion Scale(
 float scaleX,
 float scaleY
 )
```

### Parameters

*scale*

Isotropic scale

*scaleX*

Horizontal scale

*scaleY*

Vertical scale

# ERectangleRegion::Translate

Creates a new ERectangleRegion by translating the ERectangleRegion.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERectangleRegion Translate(
 float dx,
 float dy
 )
```

### Parameters

*dx*

Horizontal translation in pixel value

*dy*

Vertical translation in pixel value

## ERectangleRegion::GetWidth

## ERectangleRegion::SetWidth

Width of the region

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetWidth() const
void SetWidth(float width)
```

# 4.163. ERectangleShape Class

Manages a rectangle shape.

**Base Class:** EShape
**Derived Class(es):** EBarCode ERectangleGauge
**Namespace:** Euresys::Open_eVision_2_10

### Methods

| | |
|---|---|
| Closest | Find the daughter shape that is the closest to this shape. To retrieve the closest shape, use EShape::ClosestShape. |
| CopyTo | Copies all the data of the current ERectangleShape object into another ERectangleShape object and returns it. |

| | |
|---|---|
| Drag | Moves a handle to a new position and updates the position parameters of the shape. |
| Draw | Draws a graphical representation of a shape, as defined by EDrawingMode. |
| DrawWithCurrentPen | Draws a graphical representation of a shape, as defined by EDrawingMode. |
| GetAngle | Orientation of the shape. |
| GetCenter | Center point of the shape. |
| GetCenterX | Abscissa of the origin point of the shape. |
| GetCenterY | Ordinate of the origin point of the shape. |
| GetCorners | Retrieves the coordinates of each corner of a ERectangleShape object. |
| GetEdges | Retrieves each edge of a ERectangleShape object. |
| GetMidEdges | Retrieves the center coordinates of each edge of a ERectangleShape object. |

| | |
|---|---|
| GetPoint | Returns the coordinates of a particular point, specified by its location in the ERectangleShape area. |
| GetScale | Horizontal sensor resolution, in pixels per unit. |
| GetSizeX | X size of the ERectangleShape |
| GetSizeY | Y size of the ERectangleShape |
| GetType | Shape type. |
| HitTest | Checks if there is a handle under the cursor. |
| operator= | Copies all the data from another ERectangleShape object into the current ERectangleShape object |
| SetAngle | Orientation of the shape. |
| SetCenter | Center point of the shape. |
| SetCenterXY | Sets the center coordinates of a ERectangleShape object. |
| SetFromOppositeCorners | Sets the geometric parameters of an ERectangleShape object using two opposed corners. |

| | |
|---|---|
| SetFromOriginMiddleEnd | DEPRECATED (you should use ERectangleShape::SetFromThreeCorners): Sets the geometric parameters of an ERectangleShape object using three corners. |
| SetFromThreeCorners | Sets the geometric parameters of an ERectangleShape object using three corners. |
| SetFromTwoPoints | DEPRECATED (you should use ERectangleShape::SetFromOppositeCorners): Sets the geometric parameters of an ERectangleShape object using two opposed corners. |
| SetRectangle | Sets the parameters of the rectangle according to a known ERectangle object. |
| SetScale | Horizontal sensor resolution, in pixels per unit. |
| SetSize | Sets the size of a ERectangleShape object. |

# ERectangleShape::GetAngle

# ERectangleShape::SetAngle

Orientation of the shape.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetAngle() const
void SetAngle(float f32Angle)
```

# ERectangleShape::GetCenter

# ERectangleShape::SetCenter

Center point of the shape.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetCenter() const
void SetCenter(const EPoint& center)
```

# ERectangleShape::GetCenterX

Abscissa of the origin point of the shape.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetCenterX() const
```

# ERectangleShape::GetCenterY

Ordinate of the origin point of the shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCenterY() const
```

# ERectangleShape::Closest

Find the daughter shape that is the closest to this shape. To retrieve the closest shape, use EShape::ClosestShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Closest(
 )
```

# ERectangleShape::CopyTo

Copies all the data of the current ERectangleShape object into another ERectangleShape object and returns it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERectangleShape* CopyTo(
 ERectangleShape* dest,
 BOOL bRecursive
 )
```

### Parameters

*dest*

Pointer to the ERectangleShape object in which the current ERectangleShape object data have to be copied.

*bRecursive*

**TRUE** if the children shapes have to be copied as well, **FALSE** otherwise.

### Remarks

In case of a **NULL** pointer, a new ERectangleShape object will be created and returned.

# ERectangleShape::Drag

Moves a handle to a new position and updates the position parameters of the shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 OEV_INT32 n32CursorX,
 OEV_INT32 n32CursorY
 )
```

### Parameters

*n32CursorX*

Current cursor coordinates.

*n32CursorY*

Current cursor coordinates.

# ERectangleShape::Draw

Draws a graphical representation of a shape, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
void Draw(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

*color*

-

# ERectangleShape::DrawWithCurrentPen

Draws a graphical representation of a shape, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

# ERectangleShape::GetCorners

Retrieves the coordinates of each corner of a ERectangleShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void GetCorners(
 EPoint& xy,
 EPoint& XXy,
 EPoint& xYY,
 EPoint& XXYY
 )
```

### Parameters

*xy*

Coordinates of the lower leftmost corner of the ERectangleShape object.

*XXy*

Coordinates of the lower rightmost corner of the ERectangleShape object.

*xYY*

Coordinates of the upper leftmost corner of the ERectangleShape object.

*XXYY*

Coordinates of the upper rightmost corner of the ERectangleShape object.

# ERectangleShape::GetEdges

Retrieves each edge of a ERectangleShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetEdges(
 ELine& x,
 ELine& XX,
 ELine& y,
 ELine& YY
 )
```

### Parameters

*x*

Leftmost edge of the ERectangleShape object.

*XX*

Rightmost edge of the ERectangleShape object.

*Y*

Lower edge of the ERectangleShape object.

*YY*

Upper edge of the ERectangleShape object.

# ERectangleShape::GetMidEdges

Retrieves the center coordinates of each edge of a ERectangleShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetMidEdges(
 EPoint& x,
 EPoint& XX,
 EPoint& y,
 EPoint& YY
 )
```

## Parameters

*x*

Center coordinates of the leftmost edge of the ERectangleShape object.

*XX*

Center coordinates of the rightmost edge of the ERectangleShape object.

*y*

Center coordinates of the lower edge of the ERectangleShape object.

*YY*

Center coordinates of the upper edge of the ERectangleShape object.

# ERectangleShape::GetPoint

Returns the coordinates of a particular point, specified by its location in the ERectangleShape area.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetPoint(
 float fractionX,
 float fractionY
 )
```

## Parameters

*fractionX*

Point location expressed as a fraction of the ERectangleShape vertical edges (range -1, +1).

*fractionY*

Point location expressed as a fraction of the ERectangleShape horizontal edges (range -1, +1).

# ERectangleShape::HitTest

Checks if there is a handle under the cursor.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL HitTest(
 BOOL bDaughters
 )
```

## Parameters

*bDaughters*

Indicates if the check must be done in the whole hierarchy or just this object.

# ERectangleShape::operator=

Copies all the data from another ERectangleShape object into the current ERectangleShape object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERectangleShape& operator=(
 const ERectangleShape& other
 )
```

## Parameters

*other*

ERectangleShape object to be copied

# ERectangleShape::SetRectangle

Sets the parameters of the rectangle according to a known ERectangle object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetRectangle(const ERectangle& rectangle)
```

# ERectangleShape::GetScale

# ERectangleShape::SetScale

Horizontal sensor resolution, in pixels per unit.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetScale() const

void SetScale(float f32Scale)
```

# ERectangleShape::SetCenterXY

Sets the center coordinates of a ERectangleShape object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetCenterXY(
 float centerX,
 float centerY
 )
```

## Parameters

*centerX*

Center coordinates of the ERectangleShape object.

*centerY*

Center coordinates of the ERectangleShape object.

# ERectangleShape::SetFromOppositeCorners

Sets the geometric parameters of an ERectangleShape object using two opposed corners.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromOppositeCorners(
 const EPoint& origin,
 const EPoint& end
 )
```

### Parameters

*origin*

Origin point coordinates of the rectangle.

*end*

End point coordinates of the rectangle.

### Remarks

In a direct coordinate system, the abscissa extends rightwards and the ordinate extends upwards. The coordinate system is said to be inverse if the abscissa extends rightwards and the ordinate extends downwards.

# ERectangleShape::SetFromOriginMiddleEnd

DEPRECATED (you should use ERectangleShape::SetFromThreeCorners): Sets the geometric parameters of an ERectangleShape object using three corners.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void SetFromOriginMiddleEnd(
 const EPoint& origin,
 const EPoint& middle,
 const EPoint& end
 )
```

### Parameters

*origin*

Origin point coordinates of the rectangle.

*middle*

Middle point coordinates of the rectangle.

*end*

End point coordinates of the rectangle.

### Remarks

In a direct coordinate system, the abscissa extends rightwards and the ordinate extends upwards. The coordinate system is said to be inverse if the abscissa extends rightwards and the ordinate extends downwards.

# ERectangleShape::SetFromThreeCorners

Sets the geometric parameters of an ERectangleShape object using three corners.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromThreeCorners(
 const EPoint& origin,
 const EPoint& middle,
 const EPoint& end
 )
```

### Parameters

*origin*

Origin point coordinates of the rectangle.

*middle*

Middle point coordinates of the rectangle.

*end*

End point coordinates of the rectangle.

### Remarks

In a direct coordinate system, the abscissa extends rightwards and the ordinate extends upwards. The coordinate system is said to be inverse if the abscissa extends rightwards and the ordinate extends downwards.

# ERectangleShape::SetFromTwoPoints

DEPRECATED (you should use ERectangleShape::SetFromOppositeCorners): Sets the geometric parameters of an ERectangleShape object using two opposed corners.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromTwoPoints(
 const EPoint& origin,
 const EPoint& end
 )
```

### Parameters

*origin*

Origin point coordinates of the rectangle.

*end*

End point coordinates of the rectangle.

### Remarks

In a direct coordinate system, the abscissa extends rightwards and the ordinate extends upwards. The coordinate system is said to be inverse if the abscissa extends rightwards and the ordinate extends downwards.

# ERectangleShape::SetSize

Sets the size of a ERectangleShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetSize(
 float sizeX,
 float sizeY
 )
```

### Parameters

*sizeX*

Nominal size X of the ERectangleShape object. Default values is **100**.

*sizeY*

Nominal size Y of the ERectangleShape object. Default values is **100**.

### Remarks

A ERectangleShape object is fully defined knowing its nominal position (given by the coordinates of its center), its nominal size, its rotation angle and its outline tolerance. By default, the width and height values are **100**, which means 100 pixels when the field of view is not calibrated and 100 "units" in case of a calibrated field of view.

# ERectangleShape::GetSizeX

X size of the ERectangleShape

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSizeX() const
```

# ERectangleShape::GetSizeY

Y size of the ERectangleShape

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSizeY() const
```

## ERectangleShape::GetType

Shape type.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EShapeType GetType()
```

# 4.164. ERectangularCropper Class

Manages a point cloud cropper in the shape of a rectangular parallelepiped.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

### Methods

| | |
|---|---|
| Crop | Crops a EPointCloud. |
| ERectangularCropper | Creates an ERectangularCropper object. |

| | |
|---|---|
| operator= | Assignment operator. |

# ERectangularCropper::Crop

Crops a EPointCloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Crop(
 EPointCloud& cloudIn,
 EPointCloud& cloudOut,
 bool invertCrop
 )
```

### Parameters

*cloudIn*

Cloud to be cropped.

*cloudOut*

Cropped cloud.

*invertCrop*

Indicates if the points kept must be the points inside (true) or outside (false) the rectangular parallelepiped.

# ERectangularCropper::ERectangularCropper

Creates an ERectangularCropper object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ERectangularCropper(
 E3DPoint& center,
 float xSize,
 float ySize,
 float zSize,
 float roll,
 float pitch,
 float yaw
 )
void ERectangularCropper(
 const ERectangularCropper& other
 )
```

### Parameters

*center*

Center of the rectangular parallelepiped.

*xSize*

Size along the X axis before rotation of the rectangular parallelepiped.

*ySize*

Size along the Y axis before rotation of the rectangular parallelepiped.

*zSize*

Size along the Z axis before rotation of the rectangular parallelepiped.

*roll*

Roll (rotation along the X axis) of the rectangular parallelepiped.

*pitch*

Pitch (rotation along the Y axis) of the rectangular parallelepiped.

*yaw*

Yaw (rotation along the Z axis) of the rectangular parallelepiped.

*other*

Reference ERectangularCropper used for the initialization.

# ERectangularCropper::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
ERectangularCropper& operator=(
 const ERectangularCropper& other
 )
```

### Parameters

*other*

An other ERectangularCropper.

# 4.165. EReferenceImageSegmenter Class

Segments an image using a pixel-by-pixel single threshold given as an image.

### Remarks

This segmenter is applicable to EROIBW8, EROIBW16 and EROIC24 images. It produces coded images with two layers. The threshold is defined for each pixel individually by means of a reference image of the same type as the source image.

For grayscales images, the White layer (usually, with index 1) contains unmasked pixels having a gray value in a range defined by the gray value of the respective pixel in the reference image and the white color.

For RGB color images, the White layer (usually, with index 1) contains unmasked pixels having a color inside the cube of the RGB color space defined by the color of the respective pixel in the reference image and the white color (255,255,255).

The Black layer (usually, with index 0) contains the remaining unmasked pixels.

**Base Class:** ETwoLayersImageSegmenter

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

## Methods

| | |
|---|---|
| GetReferenceImageBW16 | Reference image for the segmentation of EROIBW16 images. |

| | |
|---|---|
| GetReferenceImageBW8 | Reference image for the segmentation of EROIBW8 images. |
| GetReferenceImageC24 | Reference image for the segmentation of EROIC24 images. |
| SetBlackLayerEncoded | Black layer encoding status. |
| SetBlackLayerIndex | Index of the black layer in the destination coded image. |
| SetReferenceImageBW16 | Reference image for the segmentation of EROIBW16 images. |
| SetReferenceImageBW8 | Reference image for the segmentation of EROIBW8 images. |
| SetReferenceImageC24 | Reference image for the segmentation of EROIC24 images. |
| SetWhiteLayerEncoded | White layer encoding status. |
| SetWhiteLayerIndex | Index of the white layer in the destination coded image. |

# EReferenceImageSegmenter::SetBlackLayerEncoded

Black layer encoding status.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
void SetBlackLayerEncoded(bool encode)
```

# EReferenceImageSegmenter::SetBlackLayerIndex

Index of the black layer in the destination coded image.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
void SetBlackLayerIndex(OEV_UINT32 index)
```

## Remarks

Setting this property automatically switches on the encoding of the black layer.

# EReferenceImageSegmenter::GetReferenceImageBW16

# EReferenceImageSegmenter::SetReferenceImageBW16

Reference image for the segmentation of EROIBW16 images.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
const EROIBW16* GetReferenceImageBW16() const
```

```
void SetReferenceImageBW16(const EROIBW16* image)
```

## EReferenceImageSegmenter::GetReferenceImageBW8

## EReferenceImageSegmenter::SetReferenceImageBW8

Reference image for the segmentation of EROIBW8 images.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
const EROIBW8* GetReferenceImageBW8() const
void SetReferenceImageBW8(const EROIBW8* image)
```

## EReferenceImageSegmenter::GetReferenceImageC24

## EReferenceImageSegmenter::SetReferenceImageC24

Reference image for the segmentation of EROIC24 images.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
const EROIC24* GetReferenceImageC24() const
void SetReferenceImageC24(const EROIC24* image)
```

## EReferenceImageSegmenter::SetWhiteLayerEncoded

White layer encoding status.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
void SetWhiteLayerEncoded(bool encode)
```

## EReferenceImageSegmenter::SetWhiteLayerIndex

Index of the white layer in the destination coded image.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
void SetWhiteLayerIndex(OEV_UINT32 index)
```

### Remarks

Setting this property automatically switches on the encoding of the white layer.

# 4.166. ERegion Class

Manages a complete context for a ERegion (Abritrary Shaped ROI)

**Derived Class(es):** EEllipseRegion ECircleRegion EPolygonRegion ERectangleRegion

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| CropRuns | Creates a new ERegion by cropping the ERegion runs within a given area. |
| Drag | Moves the specified handle to a new position and updates all placement parameters of the region. |
| Draw | Draws the ERegion shape. |
| DrawContour | Draws the ERegion contour. |
| DrawContourWithCurrentPen | Draws the ERegion contour with the current pen. |
| DrawHandles | Draws the region handles. |
| DrawHandlesWithCurrentPen | Draws the region handles with the current pen. |
| DrawWithCurrentPen | Draws the ERegion area. |
| ERegion | Constructs an ERegion context. |
| GetBoundingBoxHeight | Bounding box height. |

| | |
|---|---|
| GetBoundingBoxOrgX | Bounding box origin abscissa. |
| GetBoundingBoxOrgY | Bounding box origin ordinate. |
| GetBoundingBoxWidth | Bounding box width. |
| GetEditionMode | Graphical edition mode |
| GetRuns | Runs of the region |
| HitTest | Detects if the cursor is placed over one of the dragging handles. |
| Intersection | Creates a new ERegion by intersecting two ERegion. |
| Load | Loads the ERegion. The given ESerializer must have been created for reading. |
| operator!= | Checks if this ERegion instance is not strictly equal to another (order of the runs included) |
| operator= | Assignment operator. |
| operator== | Checks if this ERegion instance is strictly equal to another (order of the runs included) |

| | |
|---|---|
| Prepare | Computes the values necessary for the ERegion to be used during processing. |
| Save | Saves the ERegion. The given ESerializer must have been created for writing. |
| SetEditionMode | Graphical edition mode |
| SetRuns | Runs of the region |
| Subtraction | Creates a new ERegion by substracting one ERegion from another. |
| ToImage | Exports the ERegion to a mask-type image. |
| Union | Creates a new ERegion by combining two ERegion. |

# ERegion::GetBoundingBoxHeight

Bounding box height.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
int GetBoundingBoxHeight() const
```

### Remarks

The height is the height of the upright rectangle encompassing the ERun.

# ERegion::GetBoundingBoxOrgX

Bounding box origin abscissa.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetBoundingBoxOrgX() const
```

### Remarks

The origin is the top left corner of the upright rectangle encompassing the ERun.

# ERegion::GetBoundingBoxOrgY

Bounding box origin ordinate.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetBoundingBoxOrgY() const
```

### Remarks

The origin is the top left corner of the upright rectangle encompassing the ERun.

# ERegion::GetBoundingBoxWidth

Bounding box width.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int GetBoundingBoxWidth() const
```

## Remarks

The width is the width of the upright rectangle encompassing the ERun.

# ERegion::CropRuns

Creates a new ERegion by cropping the ERegion runs within a given area.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERegion CropRuns(
 int orgX,
 int orgY,
 int width,
 int height
 )
```

## Parameters

*orgX*

X origin of the cropping area.

*orgY*

Y origin of the cropping area.

*width*

Width of the cropping area.

*height*

height of the cropping area.

# ERegion::Drag

Moves the specified handle to a new position and updates all placement parameters of the region.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 int x,
 int y,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*x*

x-coordinate of the mouse cursor.

*y*

y-coordinate of the mouse cursor.

*zoomX*

Horizontal zoom factor. By default, true scale is used.

*zoomY*

Vertical zoom factor. By default, true scale is used.

*panX*

Horizontal pan offset. By default, no pan is added.

*panY*

Vertical pan offset. By default, no pan is added.

### Remarks

If zooming and/or panning were used when drawing the region, the same values must be used with ERegion::HitTest and ERegion::Drag.

# ERegion::Draw

Draws the ERegion shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 HDC graphicContext,
 float opacity,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 float opacity,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*opacity*

Opacity of the drawn area (range: 0.0 to 1.0).

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*color*

The color in which to draw the ERegion.

# ERegion::DrawContour

Draws the ERegion contour.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawContour(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
void DrawContour(
 HDC graphicContext,
 const ERGBColor& color,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*color*

The color in which to draw the ERegion.

# ERegion::DrawContourWithCurrentPen

Draws the ERegion contour with the current pen.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawContourWithCurrentPen(
 HDC hdc,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*hdc*

Handle of the device context on which to draw.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

# ERegion::DrawHandles

Draws the region handles.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawHandles(
 HDC hdc,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
void DrawHandles(
 HDC hdc,
 const ERGBColor& color,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
void DrawHandles(
 EDrawAdapter* drawAdapter,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*hdc*

Handle of the device context on which to draw.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*color*

The color in which to draw the region.

*drawAdapter*

-

# ERegion::DrawHandlesWithCurrentPen

Draws the region handles with the current pen.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawHandlesWithCurrentPen(
 HDC hdc,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*hdc*

Handle of the device context on which to draw.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

# ERegion::DrawWithCurrentPen

Draws the ERegion area.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 float opacity,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*opacity*

Opacity of the drawn area (range: 0.0 to 1.0).

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

# ERegion::GetEditionMode

# ERegion::SetEditionMode

Graphical edition mode

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EEditionMode GetEditionMode() const
void SetEditionMode(Euresys::Open_eVision_2_10::EEditionMode mode)
```

# ERegion::ERegion

Constructs an ERegion context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ERegion(
 )
void ERegion(
 const ERegion& other
 )
void ERegion(
 const EROIBW8& roi,
 EBW8 threshold
 )
```

```
void ERegion(
 const EROIBW16& roi,
 EBW16 threshold
 )

void ERegion(
 const EMatchPosition& position,
 int modelWidth,
 int modelHeight
 )

void ERegion(
 const EFoundPattern& pattern
 )

void ERegion(
 const ECodedElement& codedElement
 )

void ERegion(
 const std::vector<Euresys::Open_eVision_2_10::ERun>& runs
 )
```

### Parameters

*other*

ERegion context to copy.

*roi*

Mask ROI.

*threshold*

Mask Threshold.

*position*

Result of an EasyMatch process.

*modelWidth*

Width of an EasyMatch model.

*modelHeight*

Height of an EasyMatch model.

*pattern*

Result of an EasyFind process.

*codedElement*

Result of an EasyObject process.

*runs*

List of ERun.

# ERegion::HitTest

Detects if the cursor is placed over one of the dragging handles.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EEditionMode HitTest(
 int x,
 int y,
 float zoomX,
 float zoomY,
 int panX,
 int panY
 )
```

## Parameters

*x*

x-coordinate of the mouse cursor.

*y*

y-coordinate of the mouse cursor.

*zoomX*

Horizontal zoom factor. By default, true scale is used.

*zoomY*

Vertical zoom factor. By default, true scale is used.

*panX*

Horizontal pan offset. By default, no pan is added.

*panY*

Vertical pan offset. By default, no pan is added.

## Remarks

Returns a handle identifier, as defined by EEditionMode.
If zooming and/or panning were used when drawing the region, the same values must be used with ERegion::HitTest and ERegion::Drag.

# ERegion::Intersection

Creates a new ERegion by intersecting two ERegion.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERegion Intersection(
 ERegion& region1,
 ERegion& region2
 )
```

### Parameters

*region1*

First region.

*region2*

Second region.

# ERegion::Load

Loads the ERegion. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Load(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# ERegion::operator!=

Checks if this ERegion instance is not strictly equal to another (order of the runs included)

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool operator!=(
 const ERegion& other
 )
```

## Parameters

*other*

Reference to the other ERegion instance

# ERegion::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERegion& operator=(
 const ERegion& other
 )
```

## Parameters

*other*

Reference to the ERegion used for the assignment.

# ERegion::operator==

Checks if this ERegion instance is strictly equal to another (order of the runs included)

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool operator==(
 const ERegion& other
 )
```

## Parameters

*other*

Reference to the other ERegion instance

# ERegion::Prepare

Computes the values necessary for the ERegion to be used during processing.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Prepare(
 const EROIBW8& roi
 )

void Prepare(
 const EROIBW16& roi
 )

void Prepare(
 const EROIC24& roi
 )
```

```
void Prepare(
 const EZMap8& zmap
 )

void Prepare(
 const EZMap16& zmap
 )

void Prepare(
 const EZMap32f& zmap
 )

void Prepare(
 const EDepthMap8& dm
 )

void Prepare(
 const EDepthMap16& dm
 )

void Prepare(
 const EDepthMap32f& dm
 )

void Prepare(
 int width,
 int height
 )

void Prepare(
 int orgX,
 int orgY,
 int width,
 int height
 )
```

## Parameters

*roi*

Destination or source EBaseROI.

*zmap*

Destination or source EZMap.

*dm*

Destination or source EDepthMap.

*width*

Width of the source or destination context.

*height*

Height of the source or destination context.

*orgX*

X-Axis origin of the source or destination context.

*orgY*

Y-Axis origin of the source or destination context.

### Remarks

This method should be called once after the ERegion has been parameterized and before the ERegion is used.
If necessary,it will be done automatically before any usage but it will increase the processing time.

# ERegion::GetRuns

# ERegion::SetRuns

Runs of the region

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
std::vector<Euresys::Open_eVision_2_10::ERun> GetRuns() const
void SetRuns(const std::vector<Euresys::Open_eVision_2_10::ERun>& runs)
```

# ERegion::Save

Saves the ERegion. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# ERegion::Subtraction

Creates a new ERegion by substracting one ERegion from another.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
ERegion Subtraction(
 ERegion& region1,
 ERegion& region2
 )
```

### Parameters

*region1*

Original region.

*region2*

Region to substract.

# ERegion::ToImage

Exports the ERegion to a mask-type image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ToImage(
 EImageC24& img,
 EC24 background,
 EC24 foreground
 )
void ToImage(
 EImageBW8& img,
 EBW8 background,
 EBW8 foreground
 )
void ToImage(
 EImageBW16& img,
 EBW16 background,
 EBW16 foreground
 )
```

### Parameters

*img*

Destination image.

*background*

Background color.

*foreground*

Foreground color.

# ERegion::Union

Creates a new ERegion by combining two ERegion.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERegion Union(
 ERegion& region1,
 ERegion& region2
 )
```

### Parameters

*region1*

First region.

*region2*

Second region.

# 4.167. EROIBW1 Class

The EROIBW1 class is used to represent rectangular regions of interest inside BW1 black and white images. See ROIs.

**Base Class:** EBaseROI
**Derived Class(es):** EImageBW1
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EROIBW1 | - |
| GetBitIndex | - |
| GetFirstSubROI | See GetFirstSubROI in the derived classes |
| GetNextROI | See GetNextROI in the derived classes |

| | |
|---|---|
| GetNextSiblingROI | This method returns the ROI that is the next sibling of this ROI. |
| GetParent | - |
| GetPixel | Allows reading a single pixel value in the ROI or image. |
| GetTopParent | - |
| operator= | - |
| Serialize | Serializes the EROIBW1. |
| SetPixel | Allows writing a single pixel value in the ROI or image. |

E

R

# OIBW1::EROIBW1

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EROIBW1(
 )
```

```
void EROIBW1(
 const EROIBW1& other
 )
```

### Parameters

*other*

-

# EROIBW1::GetFirstSubROI

See GetFirstSubROI in the derived classes

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIBW1* GetFirstSubROI()
```

# EROIBW1::GetBitIndex

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT64 GetBitIndex(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

-

*y*

-

# EROIBW1::GetNextROI

See GetNextROI in the derived classes

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIBW1* GetNextROI(
 EBaseROI* startROI
 )
```

### Parameters

*startROI*

-

# EROIBW1::GetPixel

Allows reading a single pixel value in the ROI or image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
EBW1 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

-

*y*

-

### Remarks

Although coordinates outside of the ROI can be supplied, this function will raise an error condition if the coordinates are outside of the top parent image. It should be noted that calling this function several thousand times can be slow. The recommended way to access pixel content in an image or ROI is to use the EROIBW1 and EROIBW1 functions. For BW8 images/ROIs, using the C++ API only, it is possible to access pixels data faster through an intermediate object: EBW8PixelAccessor.

# EROIBW1::GetNextSiblingROI

This method returns the ROI that is the next sibling of this ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIBW1* GetNextSiblingROI()
```

# EROIBW1::operator=

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIBW1& operator=(
 const EROIBW1& other
 )
```

### Parameters

*other*

-

# EROIBW1::GetParent

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIBW1* GetParent()
```

# EROIBW1::Serialize

Serializes the EROIBW1.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

Serializer used for the serialization.

# EROIBW1::SetPixel

Allows writing a single pixel value in the ROI or image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetPixel(
 EBW1 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

-

*x*

-

*y*

-

### Remarks

Although coordinates outside of the ROI can be supplied, this function will raise an error condition if the coordinates are outside of the top parent image. It should be noted that calling this function several thousand times can be slow. The recommended way to access pixel content in an image or ROI is to use the EROIBW1 and EROIBW1 functions. For BW8 images/ROIs, using the C++ API only, it is possible to access pixels data faster through an intermediate object: EBW8PixelAccessor.

## EROIBW1::GetTopParent

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageBW1* GetTopParent()
```

# 4.168. EROIBW16 Class

The EROIBW16 class is used to represent rectangular regions of interest inside BW16 gray-level images. See ROIs.

**Base Class:** EBaseROI
**Derived Class(es):** EImageBW16
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EROIBW16 | - |
| GetFirstSubROI | See GetFirstSubROI in the derived classes |

| | |
|---|---|
| GetNextROI | See GetNextROI in the derived classes |
| GetNextSiblingROI | This method returns the ROI that is the next sibling of this ROI. |
| GetParent | - |
| GetPixel | Allows reading a single pixel value in the ROI or image. |
| GetTopParent | - |
| operator= | - |
| Serialize | Serializes the EROIBW16. |
| SetPixel | Allows writing a single pixel value in the ROI or image. |

E

R

# OIBW16::EROIBW16

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EROIBW16(
 )
void EROIBW16(
 const EROIBW16& other
 )
```

### Parameters

*other*

-

# EROIBW16::GetFirstSubROI

See GetFirstSubROI in the derived classes

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EROIBW16* GetFirstSubROI()
```

# EROIBW16::GetNextROI

See GetNextROI in the derived classes

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EROIBW16* GetNextROI(
 EBaseROI* startROI
 )
```

### Parameters

*startROI*

-

# EROIBW16::GetPixel

Allows reading a single pixel value in the ROI or image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW16 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

-

*y*

-

### Remarks

Although coordinates outside of the ROI can be supplied, this function will raise an error condition if the coordinates are outside of the top parent image. It should be noted that calling this function several thousand times can be slow. The recommended way to access pixel content in an image or ROI is to use the EROIBW16 and EROIBW16 functions. For BW8 images/ROIs, using the C++ API only, it is possible to access pixels data faster through an intermediate object: EBW8PixelAccessor.

# EROIBW16::GetNextSiblingROI

This method returns the ROI that is the next sibling of this ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIBW16* GetNextSiblingROI()
```

# EROIBW16::operator=

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIBW16& operator=(
 const EROIBW16& other
 )
```

## Parameters

*other*

-

# EROIBW16::GetParent

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIBW16* GetParent()
```

# EROIBW16::Serialize

Serializes the EROIBW16.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

## Parameters

*serializer*

Serializer used for the serialization.

# EROIBW16::SetPixel

Allows writing a single pixel value in the ROI or image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void SetPixel(
 EBW16 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

-

*x*

-

*y*

-

### Remarks

Although coordinates outside of the ROI can be supplied, this function will raise an error condition if the coordinates are outside of the top parent image. It should be noted that calling this function several thousand times can be slow. The recommended way to access pixel content in an image or ROI is to use the EROIBW16 and EROIBW16 functions. For BW8 images/ROIs, using the C++ API only, it is possible to access pixels data faster through an intermediate object: EBW8PixelAccessor.

# EROIBW16::GetTopParent

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EImageBW16* GetTopParent()
```

# 4.169. EROIBW32 Class

The EROIBW32 class is used to represent rectangular regions of interest inside BW32 gray-level images. See ROIs.

**Base Class:** EBaseROI
**Derived Class(es):** EImageBW32
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EROIBW32 | - |
| GetFirstSubROI | See GetFirstSubROI in the derived classes |
| GetNextROI | See GetNextROI in the derived classes |
| GetNextSiblingROI | This method returns the ROI that is the next sibling of this ROI. |
| GetParent | - |
| GetPixel | Allows reading a single pixel value in the ROI or image. |
| GetTopParent | - |
| operator= | - |

| | |
|---|---|
| Serialize | Serializes the EROIBW32. |
| SetPixel | Allows writing a single pixel value in the ROI or image. |

E

R

# OIBW32::EROIBW32

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EROIBW32(
 )
void EROIBW32(
 const EROIBW32& other
 )
```

### Parameters

*other*

-

# EROIBW32::GetFirstSubROI

See GetFirstSubROI in the derived classes

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EROIBW32* GetFirstSubROI()
```

# EROIBW32::GetNextROI

See GetNextROI in the derived classes

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EROIBW32* GetNextROI(
 EBaseROI* startROI
 )
```

## Parameters

*startROI*

-

# EROIBW32::GetPixel

Allows reading a single pixel value in the ROI or image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EBW32 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

-

*y*

-

### Remarks

Although coordinates outside of the ROI can be supplied, this function will raise an error condition if the coordinates are outside of the top parent image. It should be noted that calling this function several thousand times can be slow. The recommended way to access pixel content in an image or ROI is to use the EROIBW32 and EROIBW32 functions. For BW8 images/ROIs, using the C++ API only, it is possible to access pixels data faster through an intermediate object: EBW8PixelAccessor.

# EROIBW32::GetNextSiblingROI

This method returns the ROI that is the next sibling of this ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIBW32* GetNextSiblingROI()
```

# EROIBW32::operator=

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIBW32& operator=(
 const EROIBW32& other
 )
```

### Parameters

*other*

-

# EROIBW32::GetParent

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIBW32* GetParent()
```

# EROIBW32::Serialize

Serializes the EROIBW32.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

Serializer used for the serialization.

# EROIBW32::SetPixel

Allows writing a single pixel value in the ROI or image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetPixel(
 EBW32 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

-

*x*

-

*y*

-

### Remarks

Although coordinates outside of the ROI can be supplied, this function will raise an error condition if the coordinates are outside of the top parent image. It should be noted that calling this function several thousand times can be slow. The recommended way to access pixel content in an image or ROI is to use the EROIBW32 and EROIBW32 functions. For BW8 images/ROIs, using the C++ API only, it is possible to access pixels data faster through an intermediate object: EBW8PixelAccessor.

## EROIBW32::GetTopParent

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageBW32* GetTopParent()
```

# 4.170. EROIBW8 Class

The EROIBW8 class is used to represent rectangular regions of interest inside BW8 gray-level images. See ROIs.

**Base Class:** EBaseROI
**Derived Class(es):** EImageBW8
**Namespace:** Euresys::Open_eVision_2_10

### Methods

| | |
|---|---|
| EROIBW8 | - |
| GetFirstSubROI | See GetFirstSubROI in the derived classes |

| | |
|---|---|
| GetNextROI | See GetNextROI in the derived classes |
| GetNextSiblingROI | This method returns the ROI that is the next sibling of this ROI. |
| GetParent | - |
| GetPixel | Allows reading a single pixel value in the ROI or image. |
| GetTopParent | - |
| operator= | - |
| Serialize | Serializes the EROIBW8. |
| SetPixel | Allows writing a single pixel value in the ROI or image. |

E

R

# OIBW8::EROIBW8

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EROIBW8(
 )
void EROIBW8(
 const EROIBW8& other
 )
```

### Parameters

*other*

-

# EROIBW8::GetFirstSubROI

See GetFirstSubROI in the derived classes

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EROIBW8* GetFirstSubROI()
```

# EROIBW8::GetNextROI

See GetNextROI in the derived classes

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EROIBW8* GetNextROI(
 EBaseROI* startROI
 )
```

### Parameters

*startROI*

-

# EROIBW8::GetPixel

Allows reading a single pixel value in the ROI or image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW8 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

-

*y*

-

### Remarks

Although coordinates outside of the ROI can be supplied, this function will raise an error condition if the coordinates are outside of the top parent image. It should be noted that calling this function several thousand times can be slow. The recommended way to access pixel content in an image or ROI is to use the EROIBW8 and EROIBW8 functions. For BW8 images/ROIs, using the C++ API only, it is possible to access pixels data faster through an intermediate object: EBW8PixelAccessor.

# EROIBW8::GetNextSiblingROI

This method returns the ROI that is the next sibling of this ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIBW8* GetNextSiblingROI()
```

# EROIBW8::operator=

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIBW8& operator=(
 const EROIBW8& other
 )
```

## Parameters

*other*

-

# EROIBW8::GetParent

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIBW8* GetParent()
```

# EROIBW8::Serialize

Serializes the EROIBW8.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

## Parameters

*serializer*

Serializer used for the serialization.

# EROIBW8::SetPixel

Allows writing a single pixel value in the ROI or image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void SetPixel(
 EBW8 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

-

*x*

-

*y*

-

### Remarks

Although coordinates outside of the ROI can be supplied, this function will raise an error condition if the coordinates are outside of the top parent image. It should be noted that calling this function several thousand times can be slow. The recommended way to access pixel content in an image or ROI is to use the EROIBW8 and EROIBW8 functions. For BW8 images/ROIs, using the C++ API only, it is possible to access pixels data faster through an intermediate object: EBW8PixelAccessor.

# EROIBW8::GetTopParent

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageBW8* GetTopParent()
```

# 4.171. EROIC15 Class

The EROIC15 class is used to represent rectangular regions of interest inside C15 color images. See ROIs.

**Base Class:** EBaseROI
**Derived Class(es):** EImageC15
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EROIC15 | - |
| GetFirstSubROI | See GetFirstSubROI in the derived classes |
| GetNextROI | See GetNextROI in the derived classes |
| GetNextSiblingROI | This method returns the ROI that is the next sibling of this ROI. |
| GetParent | - |
| GetPixel | Allows reading a single pixel value in the ROI or image. |
| GetTopParent | - |
| operator= | - |

| | |
|---|---|
| Serialize | Serializes the EROIC15. |
| SetPixel | Allows writing a single pixel value in the ROI or image. |

E

R

# OIC15::EROIC15

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EROIC15(
 )
void EROIC15(
 const EROIC15& other
 )
```

### Parameters

*other*

-

# EROIC15::GetFirstSubROI

See GetFirstSubROI in the derived classes

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC15* GetFirstSubROI()
```

# EROIC15::GetNextROI

See GetNextROI in the derived classes

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC15* GetNextROI(
 EBaseROI* startROI
 )
```

### Parameters

*startROI*

-

# EROIC15::GetPixel

Allows reading a single pixel value in the ROI or image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
EC15 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

-

*y*

-

### Remarks

Although coordinates outside of the ROI can be supplied, this function will raise an error condition if the coordinates are outside of the top parent image. It should be noted that calling this function several thousand times can be slow. The recommended way to access pixel content in an image or ROI is to use the EROIC15 and EROIC15 functions. For BW8 images/ROIs, using the C++ API only, it is possible to access pixels data faster through an intermediate object: EBW8PixelAccessor.

# EROIC15::GetNextSiblingROI

This method returns the ROI that is the next sibling of this ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC15* GetNextSiblingROI()
```

# EROIC15::operator=

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC15& operator=(
 const EROIC15& other
 )
```

### Parameters

*other*

-

# EROIC15::GetParent

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC15* GetParent()
```

# EROIC15::Serialize

Serializes the EROIC15.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

Serializer used for the serialization.

# EROIC15::SetPixel

Allows writing a single pixel value in the ROI or image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetPixel(
 EC15 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

-

*x*

-

*y*

-

### Remarks

Although coordinates outside of the ROI can be supplied, this function will raise an error condition if the coordinates are outside of the top parent image. It should be noted that calling this function several thousand times can be slow. The recommended way to access pixel content in an image or ROI is to use the EROIC15 and EROIC15 functions. For BW8 images/ROIs, using the C++ API only, it is possible to access pixels data faster through an intermediate object: EBW8PixelAccessor.

## EROIC15::GetTopParent

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageC15* GetTopParent()
```

# 4.172. EROIC16 Class

The EROIC16 class is used to represent rectangular regions of interest inside C16 color images. See ROIs.

**Base Class:** EBaseROI
**Derived Class(es):** EImageC16
**Namespace:** Euresys::Open_eVision_2_10

### Methods

| | |
|---|---|
| EROIC16 | - |
| GetFirstSubROI | See GetFirstSubROI in the derived classes |

| | |
|---|---|
| GetNextROI | See GetNextROI in the derived classes |
| GetNextSiblingROI | This method returns the ROI that is the next sibling of this ROI. |
| GetParent | - |
| GetPixel | Allows reading a single pixel value in the ROI or image. |
| GetTopParent | - |
| operator= | - |
| Serialize | Serializes the EROIC16. |
| SetPixel | Allows writing a single pixel value in the ROI or image. |

E

R

# OIC16::EROIC16

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EROIC16(
 )
void EROIC16(
 const EROIC16& other
 )
```

### Parameters

*other*

-

# EROIC16::GetFirstSubROI

See GetFirstSubROI in the derived classes

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EROIC16* GetFirstSubROI()
```

# EROIC16::GetNextROI

See GetNextROI in the derived classes

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EROIC16* GetNextROI(
 EBaseROI* startROI
 )
```

### Parameters

*startROI*

-

# EROIC16::GetPixel

Allows reading a single pixel value in the ROI or image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EC16 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

-

*y*

-

### Remarks

Although coordinates outside of the ROI can be supplied, this function will raise an error condition if the coordinates are outside of the top parent image. It should be noted that calling this function several thousand times can be slow. The recommended way to access pixel content in an image or ROI is to use the EROIC16 and EROIC16 functions. For BW8 images/ROIs, using the C++ API only, it is possible to access pixels data faster through an intermediate object: EBW8PixelAccessor.

# EROIC16::GetNextSiblingROI

This method returns the ROI that is the next sibling of this ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC16* GetNextSiblingROI()
```

# EROIC16::operator=

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC16& operator=(
 const EROIC16& other
 )
```

## Parameters

*other*

-

# EROIC16::GetParent

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC16* GetParent()
```

# EROIC16::Serialize

Serializes the EROIC16.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

## Parameters

*serializer*

Serializer used for the serialization.

# EROIC16::SetPixel

Allows writing a single pixel value in the ROI or image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void SetPixel(
 EC16 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

-

*x*

-

*y*

-

### Remarks

Although coordinates outside of the ROI can be supplied, this function will raise an error condition if the coordinates are outside of the top parent image. It should be noted that calling this function several thousand times can be slow. The recommended way to access pixel content in an image or ROI is to use the EROIC16 and EROIC16 functions. For BW8 images/ROIs, using the C++ API only, it is possible to access pixels data faster through an intermediate object: EBW8PixelAccessor.

# EROIC16::GetTopParent

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageC16* GetTopParent()
```

# 4.173. EROIC24 Class

The EROIC24 class is used to represent rectangular regions of interest inside C24 color images. See ROIs.

**Base Class:** EBaseROI
**Derived Class(es):** EImageC24
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EROIC24 | - |
| GetFirstSubROI | See GetFirstSubROI in the derived classes |
| GetNextROI | See GetNextROI in the derived classes |
| GetNextSiblingROI | This method returns the ROI that is the next sibling of this ROI. |
| GetParent | - |
| GetPixel | Allows reading a single pixel value in the ROI or image. |
| GetTopParent | - |
| operator= | - |

| | |
|---|---|
| Serialize | Serializes the EROIC24. |
| SetPixel | Allows writing a single pixel value in the ROI or image. |

E

R

# OIC24::EROIC24

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EROIC24(
 )
void EROIC24(
 const EROIC24& other
 )
```

### Parameters

*other*

-

# EROIC24::GetFirstSubROI

See GetFirstSubROI in the derived classes

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EROIC24* GetFirstSubROI()
```

# EROIC24::GetNextROI

See GetNextROI in the derived classes

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EROIC24* GetNextROI(
 EBaseROI* startROI
 )
```

## Parameters

*startROI*

-

# EROIC24::GetPixel

Allows reading a single pixel value in the ROI or image.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EC24 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

-

*y*

-

### Remarks

Although coordinates outside of the ROI can be supplied, this function will raise an error condition if the coordinates are outside of the top parent image. It should be noted that calling this function several thousand times can be slow. The recommended way to access pixel content in an image or ROI is to use the EROIC24 and EROIC24 functions. For BW8 images/ROIs, using the C++ API only, it is possible to access pixels data faster through an intermediate object: EBW8PixelAccessor.

# EROIC24::GetNextSiblingROI

This method returns the ROI that is the next sibling of this ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC24* GetNextSiblingROI()
```

# EROIC24::operator=

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC24& operator=(
 const EROIC24& other
 )
```

### Parameters

*other*

-

# EROIC24::GetParent

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC24* GetParent()
```

# EROIC24::Serialize

Serializes the EROIC24.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

Serializer used for the serialization.

# EROIC24::SetPixel

Allows writing a single pixel value in the ROI or image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetPixel(
 EC24 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

-

*x*

-

*y*

-

### Remarks

Although coordinates outside of the ROI can be supplied, this function will raise an error condition if the coordinates are outside of the top parent image. It should be noted that calling this function several thousand times can be slow. The recommended way to access pixel content in an image or ROI is to use the EROIC24 and EROIC24 functions. For BW8 images/ROIs, using the C++ API only, it is possible to access pixels data faster through an intermediate object: EBW8PixelAccessor.

## EROIC24::GetTopParent

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageC24* GetTopParent()
```

# 4.174. EROIC24A Class

The EROIC24A class is used to represent rectangular regions of interest inside C24A color images. See ROIs.

**Base Class:** EBaseROI
**Derived Class(es):** EImageC24A
**Namespace:** Euresys::Open_eVision_2_10

### Methods

| | |
|---|---|
| EROIC24A | - |
| GetFirstSubROI | See GetFirstSubROI in the derived classes |

| | |
|---|---|
| GetNextROI | See GetNextROI in the derived classes |
| GetNextSiblingROI | This method returns the ROI that is the next sibling of this ROI. |
| GetParent | - |
| GetPixel | Allows reading a single pixel value in the ROI or image. |
| GetTopParent | - |
| operator= | - |
| Serialize | Serializes the EROIC24A. |
| SetPixel | Allows writing a single pixel value in the ROI or image. |

E

R

# OIC24A::EROIC24A

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EROIC24A(
 )
void EROIC24A(
 const EROIC24A& other
 )
```

### Parameters

*other*

-

# EROIC24A::GetFirstSubROI

See GetFirstSubROI in the derived classes

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EROIC24A* GetFirstSubROI()
```

# EROIC24A::GetNextROI

See GetNextROI in the derived classes

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EROIC24A* GetNextROI(
 EBaseROI* startROI
 )
```

### Parameters

*startROI*

-

# EROIC24A::GetPixel

Allows reading a single pixel value in the ROI or image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EC24A GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

-

*y*

-

### Remarks

Although coordinates outside of the ROI can be supplied, this function will raise an error condition if the coordinates are outside of the top parent image. It should be noted that calling this function several thousand times can be slow. The recommended way to access pixel content in an image or ROI is to use the EROIC24A and EROIC24A functions. For BW8 images/ROIs, using the C++ API only, it is possible to access pixels data faster through an intermediate object: EBW8PixelAccessor.

# EROIC24A::GetNextSiblingROI

This method returns the ROI that is the next sibling of this ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC24A* GetNextSiblingROI()
```

# EROIC24A::operator=

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC24A& operator=(
 const EROIC24A& other
 )
```

## Parameters

*other*

-

# EROIC24A::GetParent

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC24A* GetParent()
```

# EROIC24A::Serialize

Serializes the EROIC24A.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

## Parameters

*serializer*

Serializer used for the serialization.

# EROIC24A::SetPixel

Allows writing a single pixel value in the ROI or image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void SetPixel(
 EC24A value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

-

*x*

-

*y*

-

### Remarks

Although coordinates outside of the ROI can be supplied, this function will raise an error condition if the coordinates are outside of the top parent image. It should be noted that calling this function several thousand times can be slow. The recommended way to access pixel content in an image or ROI is to use the EROIC24A and EROIC24A functions. For BW8 images/ROIs, using the C++ API only, it is possible to access pixels data faster through an intermediate object: EBW8PixelAccessor.

# EROIC24A::GetTopParent

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EImageC24A* GetTopParent()
```

# 4.175. EROIC48 Class

The EROIC48 class is used to represent rectangular regions of interest inside C48 color images. See ROIs.

**Base Class:** EBaseROI
**Derived Class(es):** EImageC48
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EROIC48 | - |
| GetFirstSubROI | See GetFirstSubROI in the derived classes |
| GetNextROI | See GetNextROI in the derived classes |
| GetNextSiblingROI | This method returns the ROI that is the next sibling of this ROI. |
| GetParent | - |
| GetPixel | Allows reading a single pixel value in the ROI or image. |
| GetTopParent | - |
| operator= | - |

| | |
|---|---|
| Serialize | Serializes the EROIC48. |
| SetPixel | Allows writing a single pixel value in the ROI or image. |

E

R

# OIC48::EROIC48

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EROIC48(
 )
void EROIC48(
 const EROIC48& other
 )
```

## Parameters

*other*

-

# EROIC48::GetFirstSubROI

See GetFirstSubROI in the derived classes

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC48* GetFirstSubROI()
```

# EROIC48::GetNextROI

See GetNextROI in the derived classes

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC48* GetNextROI(
 EBaseROI* startROI
 )
```

### Parameters

*startROI*

-

# EROIC48::GetPixel

Allows reading a single pixel value in the ROI or image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
EC48 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

-

*y*

-

### Remarks

Although coordinates outside of the ROI can be supplied, this function will raise an error condition if the coordinates are outside of the top parent image. It should be noted that calling this function several thousand times can be slow. The recommended way to access pixel content in an image or ROI is to use the EROIC48 and EROIC48 functions. For BW8 images/ROIs, using the C++ API only, it is possible to access pixels data faster through an intermediate object: EBW8PixelAccessor.

# EROIC48::GetNextSiblingROI

This method returns the ROI that is the next sibling of this ROI.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC48* GetNextSiblingROI()
```

# EROIC48::operator=

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC48& operator=(
 const EROIC48& other
 )
```

### Parameters

*other*

-

# EROIC48::GetParent

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EROIC48* GetParent()
```

# EROIC48::Serialize

Serializes the EROIC48.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

Serializer used for the serialization.

# EROIC48::SetPixel

Allows writing a single pixel value in the ROI or image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetPixel(
 EC48 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

-

*x*

-

*y*

-

### Remarks

Although coordinates outside of the ROI can be supplied, this function will raise an error condition if the coordinates are outside of the top parent image. It should be noted that calling this function several thousand times can be slow. The recommended way to access pixel content in an image or ROI is to use the EROIC48 and EROIC48 functions. For BW8 images/ROIs, using the C++ API only, it is possible to access pixels data faster through an intermediate object: EBW8PixelAccessor.

## EROIC48::GetTopParent

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EImageC48* GetTopParent()
```

# 4.176. ERotatedBoundingBox Class

This class represents a rotated bounding box.

### Remarks

The rotated bounding box is a rotated, rectangular surface. Its coordinates are floating-point, which makes this class appropriate to handle sub-pixel surfaces.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Draw | Draws the rotated bounding box. |
| DrawWithCurrentPen | Draws the rotated bounding box. |

| | |
|---|---|
| ERotatedBoundingBox | Constructor of the rotated bounding box. |
| GetAngle | Returns the angle of the bounding box (in the current angle units). |
| GetCenter | Returns the coordinate of the center of the bounding box. |
| GetCenterX | Returns the abscissa of the center of the bounding box. |
| GetCenterY | Returns the ordinate of the center of the bounding box. |
| GetHeight | Returns the height of the bounding box. |
| GetQuadrangle | Returns the coordinates of the four corners of the bounding box. |
| GetWidth | Returns the width of the bounding box. |
| LocalToGlobalBox | Transforms a (local) rotated bounding box, as another (global) rotated bounding box whose coordinates are defined relatively to the current rotated bounding box. |
| LocalToGlobalPoint | Transforms a (local) point, as another (global) point whose coordinates are defined relatively to the current rotated bounding box. |
| operator= | - |

| | |
|---|---|
| Translate | Applies a translation on the center of the bounding box. |

# ERotatedBoundingBox::GetAngle

Returns the angle of the bounding box (in the current angle units).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetAngle() const
```

# ERotatedBoundingBox::GetCenter

Returns the coordinate of the center of the bounding box.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetCenter() const
```

# ERotatedBoundingBox::GetCenterX

Returns the abscissa of the center of the bounding box.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetCenterX() const
```

# ERotatedBoundingBox::GetCenterY

Returns the ordinate of the center of the bounding box.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetCenterY() const
```

# ERotatedBoundingBox::Draw

Draws the rotated bounding box.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 bool drawDiagonals
 )
```

```
void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 bool drawDiagonals
 )

void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 bool drawDiagonals
 )
```

### Parameters

*graphicContext*

Graphic context on which to draw.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*drawDiagonals*

Specifies whether or not the diagonals of the bounding box are drawn.

*color*

The color in which to draw the overlay.

### Remarks

Drawing is done in the device context associated to the desired window.

# ERotatedBoundingBox::DrawWithCurrentPen

Draws the rotated bounding box.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 bool drawDiagonals
 )
```

## Parameters

*graphicContext*

Graphic context on which to draw.

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0** (default), the horizontal zooming factor is used instead, so as to provide isotropic zooming.

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

*drawDiagonals*

Specifies whether or not the diagonals of the bounding box are drawn.

## Remarks

Drawing is done in the device context associated to the desired window.

# ERotatedBoundingBox::ERotatedBoundingBox

Constructor of the rotated bounding box.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ERotatedBoundingBox(
 float centerX,
 float centerY,
 float width,
 float height,
 float angle
 )
void ERotatedBoundingBox(
 )
void ERotatedBoundingBox(
 const ERotatedBoundingBox& other
 )
```

## Parameters

*centerX*

The abscissa of the center of the bounding box.

*centerY*

The ordinate of the center of the bounding box.

*width*

The width of the bounding box.

*height*

The height of the bounding box.

*angle*

The angle of the bounding box (in the current angle units).

*other*

-

# ERotatedBoundingBox::GetHeight

Returns the height of the bounding box.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetHeight() const
```

# ERotatedBoundingBox::LocalToGlobalBox

Transforms a (local) rotated bounding box, as another (global) rotated bounding box whose coordinates are defined relatively to the current rotated bounding box.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERotatedBoundingBox LocalToGlobalBox(
 const ERotatedBoundingBox& localBox
 )
```

## Parameters

*localBox*

-

# ERotatedBoundingBox::LocalToGlobalPoint

Transforms a (local) point, as another (global) point whose coordinates are defined relatively to the current rotated bounding box.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint LocalToGlobalPoint(
 const EPoint& localPoint
 )
```

### Parameters

*localPoint*

-

# ERotatedBoundingBox::operator=

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ERotatedBoundingBox& operator=(
 const ERotatedBoundingBox& other
 )
```

### Parameters

*other*

-

# ERotatedBoundingBox::GetQuadrangle

Returns the coordinates of the four corners of the bounding box.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EQuadrangle GetQuadrangle()
```

# ERotatedBoundingBox::Translate

Applies a translation on the center of the bounding box.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Translate(
 float offsetX,
 float offsetY
 )
```

## Parameters

*offsetX*

The offset along the X-axis.

*offsetY*

The offset along the Y-axis.

# ERotatedBoundingBox::GetWidth

Returns the width of the bounding box.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetWidth() const
```

# 4.177. ESamplePoint Class

A point sampled by a gauge.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| ESamplePoint | Constructor |
| GetIsOutlier | Outlier status of the sample point |
| GetIsValid | Validity of the sample point |
| GetPosition | Position of the sample point |

operator=

# ESamplePoint::ESamplePoint

Copies all the data from another ESamplePoint object into the cur

Constructor

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ESamplePoint(
 )
void ESamplePoint(
 const ESamplePoint& other
 )
```

## Parameters

*other*

-

# ESamplePoint::GetIsOutlier

Outlier status of the sample point

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool GetIsOutlier()
```

# ESamplePoint::GetIsValid

Validity of the sample point

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool GetIsValid()
```

# ESamplePoint::operator=

Copies all the data from another ESamplePoint object into the current ESamplePoint object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ESamplePoint& operator=(
 const ESamplePoint& other
 )
```

## Parameters

*other*

ESamplePoint object to be copied.

# ESamplePoint::GetPosition

Position of the sample point

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetPosition()
```

# 4.178. EScaleCalibrationModel Class

EScaleCalibrationModel is used to convert depth map 2.5D point to 3D world position, only by applying a scale factor.
That kind of "calibration" does not correct the perspective or distorsion present in depth maps.
Is is a simple and fast way to get a 3D point cloud by applying a scale to each coordinate axis of the depth map pixels.

**Base Class:** ECalibrationModel

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| EScaleCalibrationModel | Constructs a EScaleCalibrationModel. Parameters are initialized to default unit values. |
| GetFactorX | Returns the width of a pixel in metric unit. |
| GetFactorY | Returns the distance between 2 profiles (depth map lines) in metric unit. |
| GetFactorZ | Returns the scale of the pixel value in metric unit. |

| | |
|---|---|
| GetType | Returns the type of calibration model, see ECalibrationType. |
| Load | Loads the EScaleCalibrationModel calibration model. The given ESerializer must have been created for reading. |
| operator= | Assignment operator. |
| operator== | Comparison operator. |
| Save | Saves the EScaleCalibrationModel calibration model. The given ESerializer must have been created for writing. |

# caleCalibrationModel::EScaleCalibrationModel

Constructs a EScaleCalibrationModel. Parameters are initialized to default unit values.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EScaleCalibrationModel(
 )

void EScaleCalibrationModel(
 float factorX,
 float factorY,
 float factorZ
 )
```

```
void EScaleCalibrationModel(
 const EScaleCalibrationModel& other
 )
```

### Parameters

*factorX*

Width of a pixel in metric unit (factor for X coordinate).

*factorY*

Distance between 2 profiles (depth map lines) in metric unit (factor for Y coordinate).

*factorZ*

Scale of the pixel value in metric unit (factor for Z coordinate).

*other*

Another EScaleCalibrationModel used for the initialization.

# EScaleCalibrationModel::GetFactorX

Returns the width of a pixel in metric unit.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetFactorX() const
```

# EScaleCalibrationModel::GetFactorY

Returns the distance between 2 profiles (depth map lines) in metric unit.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetFactorY() const
```

# EScaleCalibrationModel::GetFactorZ

Returns the scale of the pixel value in metric unit.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetFactorZ() const
```

# EScaleCalibrationModel::Load

Loads the EScaleCalibrationModel calibration model. The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Load(
 ESerializer* serializer
 )
```

## Parameters

*serializer*

The serializer.

# EScaleCalibrationModel::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EScaleCalibrationModel& operator=(
 const EScaleCalibrationModel& other
 )
```

## Parameters

*other*

Another EScaleCalibrationModel.

# EScaleCalibrationModel::operator==

Comparison operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool operator==(
 const EScaleCalibrationModel& other
 )
```

## Parameters

*other*

Another EScaleCalibrationModel.

# EScaleCalibrationModel::Save

Saves the EScaleCalibrationModel calibration model. The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The serializer.

# EScaleCalibrationModel::GetType

Returns the type of calibration model, see ECalibrationType.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
Euresys::Open_eVision_2_10::Easy3D::ECalibrationType GetType() const
```

# 4.179. ESearchParamsType Class

This class is instantiated once in each EMatrixCodeReader and represents the search parameters that are explored when reading a EMatrixCode.

### Remarks

This class contains 4 sets of search parameters that are scanned at read time. At EMatrixCodeReader construction time, these sets of values are initialized with all possible values. This means, for instance, that a data matrix code read in a freshly created reader is matched against all possible logical sizes. As a consequence, the default values of these sets are not repeated here. They are assumed to represent every possible search parameters. The ESearchParamsType object needs to be used only if you wish to "override" the learning process.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddContrast | Adds a new contrast mode to the list of EMatrixCodeContrastMode the candidate is matched against at read time. |
| AddFamily | Adds a new family to the list of EFamily the candidate is matched against at read time. |
| AddFlipping | Adds a new flipping to the list of EFlipping the candidate is matched against at read time. |
| AddLogicalSize | Adds a new logical size to the list of ELogicalSize the candidate is matched against at read time. |
| ClearContrast | Clears the list of contrast modes the candidate is matched against at read time. |

| | |
|---|---|
| ClearFamily | Clears the list of families the candidate is matched against at read time. |
| ClearFlipping | Clears the list of flippings the candidate is matched against at read time. |
| ClearLogicalSize | Clears the list of logical sizes the candidate is matched against at read time. |
| GetContrast | Gets an item in the list of EMatrixCodeContrastMode the candidate is matched against at read time. |
| GetContrastCount | The size of the list of EMatrixCodeContrastMode the candidate is matched against at read time. |
| GetFamily | Gets an item in the list of EFamily the candidate is matched against at read time. |
| GetFamilyCount | The size of the list of EFamily the candidate is matched against at read time. |
| GetFlipping | Gets an item in the list of EFlipping the candidate is matched against at read time. |
| GetFlippingCount | The size of the list of EFlipping the candidate is matched against at read time. |

| | |
|---|---|
| GetLogicalSize | Gets an item in the list of ELogicalSize the candidate is matched against at read time. |
| GetLogicalSizeCount | The size of the list of ELogicalSize the candidate is matched against at read time. |
| RemoveContrast | Removes the contrast mode from the list of EMatrixCodeContrastMode the candidate is matched against at read time. |
| RemoveFamily | Removes the family from the list of EFamily the candidate is matched against at read time. |
| RemoveFlipping | Removes the flipping from the list of EFlipping the candidate is matched against at read time. |
| RemoveLogicalSize | Removes the logical size from the list of ELogicalSize the candidate is matched against at read time. |

E

S

# earchParamsType::AddContrast

Adds a new contrast mode to the list of EMatrixCodeContrastMode the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void AddContrast(
 Euresys::Open_eVision_2_10::EMatrixCodeContrastMode searchContrast
 )
```

### Parameters

*searchContrast*

Contrast mode to add to the list.

# ESearchParamsType::AddFamily

Adds a new family to the list of EFamily the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddFamily(
 Euresys::Open_eVision_2_10::EFamily searchFamily
 )
```

### Parameters

*searchFamily*

Family to add to the list.

# ESearchParamsType::AddFlipping

Adds a new flipping to the list of EFlipping the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddFlipping(
 Euresys::Open_eVision_2_10::EFlipping searchFlipping
 )
```

### Parameters

*searchFlipping*

Flipping to add to the list.

# ESearchParamsType::AddLogicalSize

Adds a new logical size to the list of ELogicalSize the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddLogicalSize(
 Euresys::Open_eVision_2_10::ELogicalSize searchLogicalSize
 )
```

### Parameters

*searchLogicalSize*

Logical size to add to the list.

# ESearchParamsType::ClearContrast

Clears the list of contrast modes the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ClearContrast(
 )
```

# ESearchParamsType::ClearFamily

Clears the list of families the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ClearFamily(
 )
```

# ESearchParamsType::ClearFlipping

Clears the list of flippings the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ClearFlipping(
 )
```

# ESearchParamsType::ClearLogicalSize

Clears the list of logical sizes the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void ClearLogicalSize(
 )
```

# ESearchParamsType::GetContrastCount

The size of the list of EMatrixCodeContrastMode the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 GetContrastCount() const
```

# ESearchParamsType::GetFamilyCount

The size of the list of EFamily the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 GetFamilyCount() const
```

# ESearchParamsType::GetFlippingCount

The size of the list of EFlipping the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 GetFlippingCount() const
```

# ESearchParamsType::GetContrast

Gets an item in the list of EMatrixCodeContrastMode the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::EMatrixCodeContrastMode GetContrast(
 OEV_INT32 index
 )
```

## Parameters

*index*

Position in the list.

# ESearchParamsType::GetFamily

Gets an item in the list of EFamily the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EFamily GetFamily(
 OEV_INT32 index
 )
```

### Parameters

*index*

Position in the list.

# ESearchParamsType::GetFlipping

Gets an item in the list of EFlipping the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EFlipping GetFlipping(
 OEV_INT32 index
 )
```

### Parameters

*index*

Position in the list.

# ESearchParamsType::GetLogicalSize

Gets an item in the list of ELogicalSize the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::ELogicalSize GetLogicalSize(
 OEV_INT32 index
 )
```

### Parameters

*index*

Position in the list.

# ESearchParamsType::GetLogicalSizeCount

The size of the list of ELogicalSize the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetLogicalSizeCount() const
```

# ESearchParamsType::RemoveContrast

Removes the contrast mode from the list of EMatrixCodeContrastMode the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveContrast(
 Euresys::Open_eVision_2_10::EMatrixCodeContrastMode searchContrast
 )
```

### Parameters

*searchContrast*

Contrast mode to remove from the list.

# ESearchParamsType::RemoveFamily

Removes the family from the list of EFamily the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveFamily(
 Euresys::Open_eVision_2_10::EFamily searchFamily
 )
```

### Parameters

*searchFamily*

Family to remove from the list.

# ESearchParamsType::RemoveFlipping

Removes the flipping from the list of EFlipping the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveFlipping(
 Euresys::Open_eVision_2_10::EFlipping searchFlipping
 )
```

### Parameters

*searchFlipping*

Flipping to remove from the list.

# ESearchParamsType::RemoveLogicalSize

Removes the logical size from the list of ELogicalSize the candidate is matched against at read time.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveLogicalSize(
 Euresys::Open_eVision_2_10::ELogicalSize searchLogicalSize
 )
```

### Parameters

*searchLogicalSize*

Logical size to remove from the list.

# 4.180. ESerializer Class

Abstract interface for file-like objects.

### Remarks

The ESerializer object manages operations of reading from and writing to an archive (a file on the system hard disk, for instance). ESerializer objects cannot be instantiated directly. To create an ESerializer object, one of the following static factory methods has to be used:

**Note.** An ESerializer object can not be used in the same time for reading and writing. So, ESerializer::CreateFileWriter creates an ESerializer object that should be used with **Save** methods and ESerializer::CreateFileReader creates an ESerializer object that should be used with **Load** methods.

**Derived Class(es):** EFilePointerSerializer EFileSerializer EMemorySerializer

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Close | Closes the file associated with the ESerializer object. |
| CreateFileReader | Returns an ESerializer object suitable for reading from a file. |
| CreateFileWriter | Returns an ESerializer object suitable for opening a file and writing into it. |
| CreateMemoryReader | Returns an ESerializer object suitable for reading from a buffer. |
| CreateMemoryWriter | Returns an ESerializer object suitable for serializing into a buffer. |
| GetWriting | Returns **TRUE** if the ESerializer object has been created for writing and **FALSE** otherwise. |

E

S

# erializer::Close

Closes the file associated with the ESerializer object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Close(
 )
```

# ESerializer::CreateFileReader

Returns an ESerializer object suitable for reading from a file.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ESerializer* CreateFileReader(
 const std::string& filePath
 )
```

## Parameters

*filePath*

Full path and name specification of the file to be used to create the ESerializer object.

## Remarks

It is up to users to delete the ESerializer object when they have done using it in **Load** calls. If the call does not succeed, it returns **NULL**. Please check the Open eVision error code to get further informations.

# ESerializer::CreateFileWriter

Returns an ESerializer object suitable for opening a file and writing into it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ESerializer* CreateFileWriter(
 const std::string& filePath,
 Euresys::Open_eVision_2_10::ESerializerFileWriterMode mode
 )
```

### Parameters

*filePath*

Full path and name specification of the file to be used to create the ESerializer object.

*mode*

Creation mode of the storage file, as defined by ESerializerFileWriterMode (by default, **Create**).

### Remarks

The ESerializerFileWriterMode parameter is an enumerated type that allows to control what happens when the file already exists: * If **mode** is **Create**, the call will not succeed if the file already exists. * If **mode** is **Overwrite**, the existing file will be overwritten. * If **mode** is **Append**, the new data will be appended to the existing file content. It is up to users to delete the ESerializer object when they have done using it in **Save** calls. If the call does not succeed, it returns **NULL**. Please check the Open eVision error code to get further information.

# ESerializer::CreateMemoryReader

Returns an ESerializer object suitable for reading from a buffer.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ESerializer* CreateMemoryReader(
 OEV_UINT8* buffer,
 OEV_UINT32 size
 )
```

### Parameters

*buffer*

Address of the buffer to be used by the memory serializer.

*size*

Size of the buffer to be used by the memory serializer.

### Remarks

The buffer is copied inside the internal memory of the serializer. It is up to users to delete the ESerializer object when they have done using it in **Load** calls.

# ESerializer::CreateMemoryWriter

Returns an ESerializer object suitable for serializing into a buffer.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ESerializer* CreateMemoryWriter(
 )
ESerializer* CreateMemoryWriter(
 OEV_UINT32 initialBufferSize
 )
ESerializer* CreateMemoryWriter(
 OEV_UINT8* buffer,
 OEV_UINT32 size
 )
```

### Parameters

*initialBufferSize*

-

*buffer*

Address of the buffer to be used by the memory serializer.

*size*

Size of the buffer to be used by the memory serializer.

### Remarks

If a buffer is not provided, a buffer will be automatically created. Please note that, in this case, if you didn't provide a size, the buffer will be automatically resized as needed. It is up to users to delete the ESerializer object when they have done using it in **Save** calls. The returned serializer underlying type is EMemorySerializer, for more information, see the class documentation.

# ESerializer::GetWriting

Returns **TRUE** if the ESerializer object has been created for writing and **FALSE** otherwise.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetWriting() const
```

# 4.181. EShape Class

Abstract class to federate the classes that can be hierarchically attached together (from a geometrical point of view).

**Derived Class(es):** EWorldShape ERectangleShape ECircleShape EFrameShape ELineShape EPointShape EWedgeShape
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Attach | Attaches the gauge to a mother gauge or shape. |
| Closest | Find the daughter shape that is the closest to this shape. To retrieve the closest shape, use EShape::ClosestShape. |
| Detach | Detaches the gauge from its mother gauge or shape. |
| DetachDaughters | Detaches the daughter gauges or shapes. |

| | |
|---|---|
| DisableBehaviorFilter | Disables (i.e. removes) a condition from the list of conditions in the behavior filter. |
| DisableTypeFilter | Enables all shape types |
| Drag | Moves a handle to a new position and updates the position parameters of the shape. |
| Draw | Draws a graphical representation of a shape, as defined by EDrawingMode. |
| DrawWithCurrentPen | Draws a graphical representation of a shape, as defined by EDrawingMode. |
| EnableBehaviorFilter | Modifies the so-called **behavior filter** that is a conjunction of conditions specifying when a shape or a gauge should be displayed or taken into consideration when monitoring the mouse interactions. |
| EnableTypeFilter | Enables the filter of the specified shape type |
| GetActive | Flag indicating whether the shape is active or not. |
| GetActualShape | Flag indicating whether an inquiry returns a result pertaining to the nominal gauge (**FALSE**, default) or the fitted model (**TRUE**). |

| | |
|---|---|
| GetAllocated | Gets the allocated flag. |
| GetClosestShape | Closest shape among the daughters. |
| GetDaughter | Returns a pointer to the specified daughter gauge or shape. |
| GetDragable | Flag indicating whether the shape can be dragged or not. |
| GetDraggingMode | Gets the dragging mode which defines how the shape could be dragged. |
| GetHitHandle | Handle currently under the cursor. |
| GetHitShape | Pointer to the shape currently under the cursor. |
| GetLabeled | Flag indicating whether the shape label should be displayed or not. |
| GetMother | Pointer to the mother shape. |
| GetName | Name of the EShape object. |
| GetNumDaughters | Number of daughters attached to the shape. |

| | |
|---|---|
| GetPanX | Current horizontal panning factor for drawing operations. |
| GetPanY | Current vertical panning factor for drawing operations. |
| GetResizable | Flag indicating whether the shape can be resized or not. |
| GetRotatable | Flag indicating whether the shape can be rotated or not. |
| GetSelectable | Flag indicating whether the shape can be selected or not. |
| GetSelected | Flag indicating whether the shape is selected or not. |
| GetShapeNamed | Returns a pointer to a daughter gauge or shape specified by its name. |
| GetType | Shape type. |
| GetVisible | Flag indicating whether the shape is visible or not. |
| GetWorldShape | World ancestor. |
| GetZoomX | Current horizontal zooming factor for drawing operations. |
| GetZoomY | Current vertical zooming factor for drawing operations. |

| | |
|---|---|
| HitTest | Checks if there is a handle under the cursor. |
| InvalidateWorld | Invalidates the world shape for this shape and all its ancestors |
| Load | Loads a classifier.The given ESerializer must have been created for reading. |
| LocalToSensor | Transforms a point from local coordinates to sensor coordinates. |
| Save | Loads a classifier.The given ESerializer must have been created for writing. |
| SensorToLocal | Transforms a point from sensor coordinates to local coordinates. |
| SetActive | Flag indicating whether the shape is active or not. |
| SetActiveRecursive | Sets the flag indicating whether the shape and all its daughters are active or not. |
| SetActualShape | Flag indicating whether an inquiry returns a result pertaining to the nominal gauge (**FALSE**, default) or the fitted model (**TRUE**). |
| SetActualShapeRecursive | Sets the flag indicating whether an inquiry returns a result pertaining to the nominal gauge (**FALSE**, default) or the fitted model (**TRUE**). The flag is set in this shape and all its daughters. |

| | |
|---|---|
| SetAllocated | Sets the allocated flag. |
| SetCursor | Sets the cursor current coordinates. |
| SetDragable | Flag indicating whether the shape can be dragged or not. |
| SetDragableRecursive | Sets the flag indicating whether the shape and all its daughters can be dragged or not. |
| SetDraggingMode | Sets the dragging mode which defines how the shape could be dragged. |
| SetLabeled | Flag indicating whether the shape label should be displayed or not. |
| SetLabeledRecursive | Sets the flag indicating whether the shape label should be displayed or not. The flag is set in this shape and all its daughters. |
| SetName | Name of the EShape object. |
| SetPan | Sets the horizontal and vertical panning factors for drawing operations. |
| SetResizable | Flag indicating whether the shape can be resized or not. |

| | |
|---|---|
| SetResizableRecursive | Sets the flag indicating whether the shape and all its daughters can be resized or not. |
| SetRotatable | Flag indicating whether the shape can be rotated or not. |
| SetRotatableRecursive | Sets the flag indicating whether the shape and all its daughters can be rotated or not. |
| SetSelectable | Flag indicating whether the shape can be selected or not. |
| SetSelectableRecursive | Sets the flag indicating whether the shape and all its daughters can be selected or not. |
| SetSelected | Flag indicating whether the shape is selected or not. |
| SetSelectedRecursive | Sets the flag indicating whether the shape and all its daughters are selected or not. |
| SetVisible | Flag indicating whether the shape is visible or not. |
| SetVisibleRecursive | Sets the flag indicating whether the shape and its daughter shapes are visible or not. |
| SetZoom | Sets the horizontal and vertical zooming factors for drawing operations. |

# EShape::GetActive

# EShape::SetActive

Flag indicating whether the shape is active or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetActive()
void SetActive(BOOL active)
```

# EShape::SetActiveRecursive

Sets the flag indicating whether the shape and all its daughters are active or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetActiveRecursive(BOOL active)
```

## Remarks

When complex gauging is required, several gauges can be grouped together. Applying EShape::Process to the mother gauge or shape triggers the measurement of the whole. Only the active gauges will participate in the process.

# EShape::GetActualShape

# EShape::SetActualShape

Flag indicating whether an inquiry returns a result pertaining to the nominal gauge (**FALSE**, default) or the fitted model (**TRUE**).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetActualShape()
void SetActualShape(BOOL actualShape)
```

# EShape::SetActualShapeRecursive

Sets the flag indicating whether an inquiry returns a result pertaining to the nominal gauge (**FALSE**, default) or the fitted model (**TRUE**). The flag is set in this shape and all its daughters.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetActualShapeRecursive(BOOL actualShape)
```

# EShape::Attach

Attaches the gauge to a mother gauge or shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Attach(
 EShape* mother
 )
```

### Parameters

*mother*

Pointer to the mother gauge or shape.

### Remarks

When attached to a mother gauge, be aware that daughter gauges are not positioned according to the nominal mother gauge position, but to its corresponding fitted model.

# EShape::Closest

Find the daughter shape that is the closest to this shape. To retrieve the closest shape, use EShape::ClosestShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Closest(
 )
```

# EShape::GetClosestShape

Closest shape among the daughters.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EShape* GetClosestShape()
```

### Remarks

Use EShape::Closest to recompute the closest shape.

# EShape::Detach

Detaches the gauge from its mother gauge or shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Detach(
 )
```

# EShape::DetachDaughters

Detaches the daughter gauges or shapes.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DetachDaughters(
 )
```

# EShape::DisableBehaviorFilter

Disables (i.e. removes) a condition from the list of conditions in the behavior filter.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DisableBehaviorFilter(
 Euresys::Open_eVision_2_10::EShapeBehavior behavior
 )
```

### Parameters

*behavior*

The behavior of the shape to be removed from the behavior filter.

### Remarks

The condition to be disabled is identified by the behavior about which the condition is. Disabling a behavior leads to less restrictive conditions for the **Draw** and **HitTest** methods to be actually carried on.

# EShape::DisableTypeFilter

Enables all shape types

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DisableTypeFilter(
 )
```

# EShape::Drag

Moves a handle to a new position and updates the position parameters of the shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 OEV_INT32 n32CursorX,
 OEV_INT32 n32CursorY
 )
```

### Parameters

*n32CursorX*

Current cursor coordinates.

*n32CursorY*

Current cursor coordinates.

# EShape::GetDragable

# EShape::SetDragable

Flag indicating whether the shape can be dragged or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetDragable()

void SetDragable(BOOL dragable)
```

# EShape::SetDragableRecursive

Sets the flag indicating whether the shape and all its daughters can be dragged or not.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetDragableRecursive(BOOL dragable)
```

# EShape::Draw

Draws a graphical representation of a shape, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Draw(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )

void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

```
void Draw(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

*color*

-

# EShape::DrawWithCurrentPen

Draws a graphical representation of a shape, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

# EShape::EnableBehaviorFilter

Modifies the so-called **behavior filter** that is a conjunction of conditions specifying when a shape or a gauge should be displayed or taken into consideration when monitoring the mouse interactions.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EnableBehaviorFilter(
 Euresys::Open_eVision_2_10::EShapeBehavior behavior,
 BOOL value
 )
```

## Parameters

*behavior*

The behavior of the shape to be tested.

*value*

The value at which the behavior property should be set to pass the test. By default, equals **True**.

## Remarks

This method registers a new necessary condition for the **Draw** and **HitTest** families of methods to be actually carried on. Such a condition is about the behavior of the shape, as specified by the **behavior** argument. Initially, the behavior filter contains an empty list of conditions, which means that the **Draw** and **HitTest** methods will always be executed. Adding a new condition through EShape::EnableBehaviorFilter will introduce a new restriction on the effective execution of these methods. Use EShape::DisableBehaviorFilter to remove a condition from the behavior filter.

# EShape::EnableTypeFilter

Enables the filter of the specified shape type

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EnableTypeFilter(
 OEV_UINT32 un32Types
 )
```

### Parameters

*un32Types*

The type of the shape to filter.

# EShape::GetAllocated

Gets the allocated flag.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetAllocated(
 )
```

# EShape::GetDaughter

Returns a pointer to the specified daughter gauge or shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
EShape* GetDaughter(
 OEV_UINT32 index
 )
```

### Parameters

*index*

Daughter gauge or shape index.

# EShape::GetDraggingMode

Gets the dragging mode which defines how the shape could be dragged.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EDraggingMode GetDraggingMode(
 )
```

# EShape::GetShapeNamed

Returns a pointer to a daughter gauge or shape specified by its name.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EShape* GetShapeNamed(
 const std::string& name
 )
```

### Parameters

*name*

Name of the daughter gauge or shape.

# EShape::GetHitHandle

Handle currently under the cursor.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EDragHandle GetHitHandle()
```

### Remarks

When the cursor is over a particular handle, its shape could be changed for feedback.

# EShape::GetHitShape

Pointer to the shape currently under the cursor.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EShape* GetHitShape()
```

### Remarks

When the cursor is over a particular shape, its shape could be changed for feedback.

# EShape::HitTest

Checks if there is a handle under the cursor.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL HitTest(
 BOOL bDaughters
 )
```

### Parameters

*bDaughters*

Indicates if the check must be done in the whole hierarchy or just this object.

# EShape::InvalidateWorld

Invalidates the world shape for this shape and all its ancestors

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void InvalidateWorld(
 )
```

# EShape::GetLabeled

# EShape::SetLabeled

Flag indicating whether the shape label should be displayed or not.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetLabeled()
void SetLabeled(BOOL labeled)
```

# EShape::SetLabeledRecursive

Sets the flag indicating whether the shape label should be displayed or not. The flag is set in this shape and all its daughters.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetLabeledRecursive(BOOL labeled)
```

# EShape::Load

Loads a classifier.The given ESerializer must have been created for reading.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Load(
 ESerializer* serializer,
 BOOL daughters
 )
```

### Parameters

*serializer*

Pointer to the ESerializer created for reading.

*daughters*

Indicates if the load must be done on the whole hierarchy or just this object.

# EShape::LocalToSensor

Transforms a point from local coordinates to sensor coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint LocalToSensor(
 const EPoint& LPoint
 )
```

### Parameters

*LPoint*

The point in local coordinates.

# EShape::GetMother

Pointer to the mother shape.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EShape* GetMother()
```

# EShape::GetName

# EShape::SetName

Name of the EShape object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::string GetName()
void SetName(const std::string& name)
```

# EShape::GetNumDaughters

Number of daughters attached to the shape.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumDaughters()
```

# EShape::GetPanX

Current horizontal panning factor for drawing operations.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetPanX()
```

# EShape::GetPanY

Current vertical panning factor for drawing operations.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetPanY()
```

# EShape::GetResizable

# EShape::SetResizable

Flag indicating whether the shape can be resized or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetResizable()
void SetResizable(BOOL resizable)
```

# EShape::SetResizableRecursive

Sets the flag indicating whether the shape and all its daughters can be resized or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetResizableRecursive(BOOL resizable)
```

# EShape::GetRotatable

# EShape::SetRotatable

Flag indicating whether the shape can be rotated or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetRotatable()
```

```
void SetRotatable(BOOL rotatable)
```

# EShape::SetRotatableRecursive

Sets the flag indicating whether the shape and all its daughters can be rotated or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetRotatableRecursive(BOOL rotatable)
```

# EShape::Save

Loads a classifier.The given ESerializer must have been created for writing.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Save(
 ESerializer* serializer,
 BOOL daughters
 )
```

## Parameters

*serializer*

Pointer to the ESerializer created for writing.

*daughters*

Indicates if the save must be done on the whole hierarchy or just this object.

# EShape::GetSelectable

# EShape::SetSelectable

Flag indicating whether the shape can be selected or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetSelectable()
void SetSelectable(BOOL selectable)
```

# EShape::SetSelectableRecursive

Sets the flag indicating whether the shape and all its daughters can be selected or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetSelectableRecursive(BOOL selectable)
```

# EShape::GetSelected

# EShape::SetSelected

Flag indicating whether the shape is selected or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetSelected()
void SetSelected(BOOL selected)
```

# EShape::SetSelectedRecursive

Sets the flag indicating whether the shape and all its daughters are selected or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetSelectedRecursive(BOOL selected)
```

# EShape::SensorToLocal

Transforms a point from sensor coordinates to local coordinates.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint SensorToLocal(
 const EPoint& SPoint
 )
```

### Parameters

*SPoint*

The point in sensor coordinates.

# EShape::SetAllocated

Sets the allocated flag.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetAllocated(
 BOOL bAllocated,
 BOOL bDaughters
 )
```

### Parameters

*bAllocated*

Whether to set the allocated flag or not.

*bDaughters*

Indicates if the set must be done in the whole hierarchy or just this object.

# EShape::SetCursor

Sets the cursor current coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetCursor(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Cursor current coordinates.

*y*

Cursor current coordinates.

# EShape::SetDraggingMode

Sets the dragging mode which defines how the shape could be dragged.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetDraggingMode(
 Euresys::Open_eVision_2_10::EDraggingMode eDraggingMode,
 BOOL bDaughters
 )
```

### Parameters

*eDraggingMode*

The draggingMode.

*bDaughters*

Indicates if the set must be done in the whole hierarchy or just this object.

# EShape::SetPan

Sets the horizontal and vertical panning factors for drawing operations.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetPan(
 float f32PanX,
 float f32PanY
 )
```

### Parameters

*f32PanX*

-

*f32PanY*

-

### Remarks

All objects attached to an EWorldShape object inherit of the same panning factor.

# EShape::SetZoom

Sets the horizontal and vertical zooming factors for drawing operations.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetZoom(
 float f32ZoomX,
 float f32ZoomY
 )
```

### Parameters

*f32ZoomX*

-

*f32ZoomY*

-

### Remarks

All objects attached to an EWorldShape inherit of the same zooming factor.

# EShape::GetType

Shape type.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EShapeType GetType()
```

# EShape::GetVisible

# EShape::SetVisible

Flag indicating whether the shape is visible or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetVisible()
void SetVisible(BOOL visible)
```

# EShape::SetVisibleRecursive

Sets the flag indicating whether the shape and its daughter shapes are visible or not.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetVisibleRecursive(BOOL visible)
```

# EShape::GetWorldShape

World ancestor.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EWorldShape* GetWorldShape()
```

# EShape::GetZoomX

Current horizontal zooming factor for drawing operations.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetZoomX()
```

## EShape::GetZoomY

Current vertical zooming factor for drawing operations.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetZoomY()
```

# 4.182. ESimpleCropper Class

Manages a point cloud cropper based on X/Y/Z value ranges.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

### Methods

| | |
|---|---|
| Crop | Crops a point cloud. |
| ESimpleCropper | Creates an ESimpleCropper object. |
| operator= | Assignment operator |

| | |
|---|---|
| SetXRange | Allowed value range along the X axis. Pass NULL if no cropping should be done along this axis. |
| SetYRange | Allowed value range along the Y axis. Pass NULL if no cropping should be done along this axis. |
| SetZRange | Allowed value range along the Z axis. Pass NULL if no cropping should be done along this axis. |

E

S

# impleCropper::Crop

Crops a point cloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Crop(
 EPointCloud& cloudIn,
 EPointCloud& cloudOut
 )
```

## Parameters

*cloudIn*

Cloud to be cropped.

*cloudOut*

Cropped cloud.

# ESimpleCropper::ESimpleCropper

Creates an ESimpleCropper object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ESimpleCropper(
 )
void ESimpleCropper(
 const EFloatRange* rangeX,
 const EFloatRange* rangeY,
 const EFloatRange* rangeZ
 )
void ESimpleCropper(
 const ESimpleCropper& other
 )
```

### Parameters

*rangeX*

Allowed value range along the X axis. Pass NULL if no cropping should be done along this axis.

*rangeY*

Allowed value range along the Y axis. Pass NULL if no cropping should be done along this axis.

*rangeZ*

Allowed value range along the Z axis. Pass NULL if no cropping should be done along this axis.

*other*

An other ESimpleCropper

# ESimpleCropper::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
ESimpleCropper& operator=(
 const ESimpleCropper& other
 )
```

### Parameters

*other*

An other ESimpleCropper

# ESimpleCropper::SetXRange

Allowed value range along the X axis. Pass NULL if no cropping should be done along this axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetXRange(const EFloatRange* rangeX)
```

# ESimpleCropper::SetYRange

Allowed value range along the Y axis. Pass NULL if no cropping should be done along this axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetYRange(const EFloatRange* rangeY)
```

# ESimpleCropper::SetZRange

Allowed value range along the Z axis. Pass NULL if no cropping should be done along this axis.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetZRange(const EFloatRange* rangeZ)
```

# 4.183. ESphericalCropper Class

Manages a spherical point cloud cropper.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| Crop | Crops a point cloud. |
| ESphericalCropper | Creates an ESphericalCropper object. |
| operator= | Assignment operator. |

# ESphericalCropper::Crop

Crops a point cloud.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Crop(
 EPointCloud& cloudIn,
 EPointCloud& cloudOut,
 bool invertCrop
 )
```

### Parameters

*cloudIn*

Cloud to be cropped.

*cloudOut*

Cropped cloud.

*invertCrop*

Indicates if the points kept must be the points inside (true) or outside (false) the sphere.

# ESphericalCropper::ESphericalCropper

Creates an ESphericalCropper object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
void ESphericalCropper(
 E3DPoint& center,
 float radius
 )

void ESphericalCropper(
 const ESphericalCropper& other
 )
```

### Parameters

*center*

Center of the sphere.

*radius*

Radius of the sphere.

*other*

An other ESphericalCropper.

# ESphericalCropper::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
ESphericalCropper& operator=(
 const ESphericalCropper& other
 )
```

### Parameters

*other*

An other ESphericalCropper.

# 4.184. EStatistics Class

Calculates various statistics on the pixels values of an EDepthMap or EZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| ComputeAverageMap | Given an input EDepthMap or EZMap, calculates a depthmap where every pixel is the average of the input depthmap pixels within a window centered on that pixel. |
| ComputePixelStatistics | Calculates the minimum, maximum, the average and optionally the standard deviation of the pixels values of an EDepthMap or EZMap.<br>EStatistics::ComputePixelStatistics does not take the resolution of the depthmap into account: it calculates statistics on the pixels gray values. |
| ComputeStandardDeviationMap | Given an input EDepthMap or EZMap, calculates an image where every pixel is the standard deviation of the input depthmap or zmap pixels within a window centered on that pixel. |
| ComputeStatistics | Calculates the minimum, maximum, the average and optionally the standard deviation of an EDepthMap or EZMap values.<br>The standard deviation is optionally calculated. EStatistics::ComputeStatistics takes the resolution of the depthmap or the resolution of the Zmap into account: it calculates statistics on the metric values: |

# EStatistics::ComputeAverageMap

Given an input EDepthMap or EZMap, calculates a depthmap where every pixel is the average of the input depthmap pixels within a window centered on that pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void ComputeAverageMap(
 const EDepthMap16& sourceMap,
 EDepthMap16& destinationMap,
 OEV_INT16 halfKernelSize,
 float minValidRatio,
 bool fillUndefinedPixels
 )

void ComputeAverageMap(
 const EDepthMap8& sourceMap,
 EDepthMap8& destinationMap,
 OEV_INT16 halfKernelSize,
 float minValidRatio,
 bool fillUndefinedPixels
 )

void ComputeAverageMap(
 const EZMap16& sourceMap,
 EZMap16& destinationMap,
 OEV_INT16 halfKernelSize,
 float minValidRatio,
 bool fillUndefinedPixels
 )

void ComputeAverageMap(
 const EZMap8& sourceMap,
 EZMap8& destinationMap,
 OEV_INT16 halfKernelSize,
 float minValidRatio,
 bool fillUndefinedPixels
 )
```

### Parameters

*sourceMap*

The input depthmap/Zmap.

*destinationMap*

The destination depthmap/Zmap. It should have the same dimensions as the input depthmap.

*halfKernelSize*

The half-size of the window used for the calculation of the standard deviation. The filter window size (= kernel size) is halfKernelSize * 2 + 1, should be positive, smaller than (or equal to) the image size, and may not exceed 256.

*minValidRatio*

required ratio of valid pixels in the filter window to process the calculation. If not enough, the output pixel will be marked as invalid. The default value of this parameter is 0.25

*fillUndefinedPixels*

This boolean controls how undefined pixels are handled: either they filled by the calculated average value, or they are left undefined (default behavior).

# EStatistics::ComputePixelStatistics

Calculates the minimum, maximum, the average and optionally the standard deviation of the pixels values of an EDepthMap or EZMap.
EStatistics::ComputePixelStatistics does not take the resolution of the depthmap into account: it calculates statistics on the pixels gray values.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ComputePixelStatistics(
 const EDepthMap8& sourceMap,
 OEV_UINT32& validCount,
 EBW8& minimumValue,
 EBW8& maximumValue,
 float& average
 )
```

```
void ComputePixelStatistics(
 const EDepthMap16& sourceMap,
 OEV_UINT32& validCount,
 EBW16& minimumValue,
 EBW16& maximumValue,
 float& average
 )

void ComputePixelStatistics(
 const EDepthMap32f& sourceMap,
 OEV_UINT32& validCount,
 EBW32f& minimumValue,
 EBW32f& maximumValue,
 float& average
 )

void ComputePixelStatistics(
 const EDepthMap16& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 EBW16& minimumValue,
 EBW16& maximumValue,
 float& average
 )

void ComputePixelStatistics(
 const EDepthMap8& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 EBW8& minimumValue,
 EBW8& maximumValue,
 float& average
 )

void ComputePixelStatistics(
 const EDepthMap32f& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 EBW32f& minimumValue,
 EBW32f& maximumValue,
 float& average
 )
```

```
void ComputePixelStatistics(
 const EDepthMap8& sourceMap,
 OEV_UINT32& validCount,
 EBW8& minimumValue,
 EBW8& maximumValue,
 float& average,
 float& stddev
 )

void ComputePixelStatistics(
 const EDepthMap8& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 EBW8& minimumValue,
 EBW8& maximumValue,
 float& average,
 float& stddev
 )

void ComputePixelStatistics(
 const EDepthMap16& sourceMap,
 OEV_UINT32& validCount,
 EBW16& minimumValue,
 EBW16& maximumValue,
 float& average,
 float& stddev
 )

void ComputePixelStatistics(
 const EDepthMap16& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 EBW16& minimumValue,
 EBW16& maximumValue,
 float& average,
 float& stddev
 )
```

```
void ComputePixelStatistics(
 const EDepthMap32f& sourceMap,
 OEV_UINT32& validCount,
 EBW32f& minimumValue,
 EBW32f& maximumValue,
 float& average,
 float& stddev
 )

void ComputePixelStatistics(
 const EDepthMap32f& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 EBW32f& minimumValue,
 EBW32f& maximumValue,
 float& average,
 float& stddev
 )

void ComputePixelStatistics(
 const EZMap8& sourceMap,
 OEV_UINT32& validCount,
 EBW8& minimumValue,
 EBW8& maximumValue,
 float& average
 )

void ComputePixelStatistics(
 const EZMap8& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 EBW8& minimumValue,
 EBW8& maximumValue,
 float& average
 )

void ComputePixelStatistics(
 const EZMap16& sourceMap,
 OEV_UINT32& validCount,
 EBW16& minimumValue,
 EBW16& maximumValue,
 float& average
 )
```

```
void ComputePixelStatistics(
 const EZMap16& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 EBW16& minimumValue,
 EBW16& maximumValue,
 float& average
 )

void ComputePixelStatistics(
 const EZMap32f& sourceMap,
 OEV_UINT32& validCount,
 EBW32f& minimumValue,
 EBW32f& maximumValue,
 float& average
 )

void ComputePixelStatistics(
 const EZMap32f& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 EBW32f& minimumValue,
 EBW32f& maximumValue,
 float& average
 )

void ComputePixelStatistics(
 const EZMap8& sourceMap,
 OEV_UINT32& validCount,
 EBW8& minimumValue,
 EBW8& maximumValue,
 float& average,
 float& stddev
 )

void ComputePixelStatistics(
 const EZMap8& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 EBW8& minimumValue,
 EBW8& maximumValue,
 float& average,
 float& stddev
 )
```

```
void ComputePixelStatistics(
 const EZMap16& sourceMap,
 OEV_UINT32& validCount,
 EBW16& minimumValue,
 EBW16& maximumValue,
 float& average,
 float& stddev
 )

void ComputePixelStatistics(
 const EZMap16& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 EBW16& minimumValue,
 EBW16& maximumValue,
 float& average,
 float& stddev
 )

void ComputePixelStatistics(
 const EZMap32f& sourceMap,
 OEV_UINT32& validCount,
 EBW32f& minimumValue,
 EBW32f& maximumValue,
 float& average,
 float& stddev
 )

void ComputePixelStatistics(
 const EZMap32f& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 EBW32f& minimumValue,
 EBW32f& maximumValue,
 float& average,
 float& stddev
 )
```

## Parameters

*sourceMap*

The input depthmap/Zmap.

*validCount*

Variable to store the number of valid pixels in sourceMap.

*minimumValue*

Variable to store the minimum value.

*maximumValue*

Variable to store the maximum value.

*average*

Variable to store the average value.

*region*

The ERegion where the statistics has to be calculated.

*stddev*

Variable to store the standard deviation.

# EStatistics::ComputeStandardDeviationMap

Given an input EDepthMap or EZMap, calculates an image where every pixel is the standard deviation of the input depthmap or zmap pixels within a window centered on that pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void ComputeStandardDeviationMap(
 const EDepthMap8& sourceMap,
 EImageBW8& destinationImage,
 OEV_INT16 halfKernelSize,
 float minValidRatio,
 bool fillUndefinedPixels
 )

void ComputeStandardDeviationMap(
 const EDepthMap16& sourceMap,
 EImageBW16& destinationImage,
 OEV_INT16 halfKernelSize,
 float minValidRatio,
 bool fillUndefinedPixels
 )
```

```
void ComputeStandardDeviationMap(
 const EDepthMap8& sourceMap,
 EImageBW16& destinationImage,
 OEV_INT16 halfKernelSize,
 float minValidRatio,
 bool fillUndefinedPixels
 )

void ComputeStandardDeviationMap(
 const EZMap16& sourceMap,
 EImageBW16& destinationImage,
 OEV_INT16 halfKernelSize,
 float minValidRatio,
 bool fillUndefinedPixels
 )

void ComputeStandardDeviationMap(
 const EZMap8& sourceMap,
 EImageBW8& destinationImage,
 OEV_INT16 halfKernelSize,
 float minValidRatio,
 bool fillUndefinedPixels
 )

void ComputeStandardDeviationMap(
 const EZMap8& sourceMap,
 EImageBW16& destinationImage,
 OEV_INT16 halfKernelSize,
 float minValidRatio,
 bool fillUndefinedPixels
 )
```

## Parameters

*sourceMap*

The input depthmap/Zmap.

*destinationImage*

The destination image. It should have the same dimensions as the input depthmap.

*halfKernelSize*

The half-size of the window used for the calculation of the standard deviation.
The filter window size (= kernel size) is halfKernelSize * 2 + 1, should be positive, smaller than (or equal to) the image size, and may not exceed 256.

*minValidRatio*

required ratio of valid pixels in the filter window to process the calculation.
If not enough, the output pixel value will be 0. The default value of this parameter is 0.25 .

*fillUndefinedPixels*

This boolean controls how undefined pixels are handled: either they filled by the calculated standard deviation, or they are left undefined (default behavior).

### Remarks

When the input depthmap is on 8 bits and the destination image is on 16 bits, the resulting standard deviation scale is 256 times larger than in the source depthmap.

# EStatistics::ComputeStatistics

Calculates the minimum, maximum, the average and optionally the standard deviation of an EDepthMap or EZMap values.
The standard deviation is optionally calculated. EStatistics::ComputeStatistics takes the resolution of the depthmap or the resolution of the Zmap into account: it calculates statistics on the metric values:

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void ComputeStatistics(
 const EDepthMap8& sourceMap,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average
 )

void ComputeStatistics(
 const EDepthMap8& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average
 )
```

```
void ComputeStatistics(
 const EDepthMap16& sourceMap,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average
 )

void ComputeStatistics(
 const EDepthMap16& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average
 )

void ComputeStatistics(
 const EDepthMap32f& sourceMap,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average
 )

void ComputeStatistics(
 const EDepthMap32f& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average
 )

void ComputeStatistics(
 const EZMap8& sourceMap,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average
 )
```

```
void ComputeStatistics(
 const EZMap8& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average
 )

void ComputeStatistics(
 const EZMap16& sourceMap,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average
 )

void ComputeStatistics(
 const EZMap16& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average
 )

void ComputeStatistics(
 const EZMap32f& sourceMap,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average
 )

void ComputeStatistics(
 const EZMap32f& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average
 )
```

```
void ComputeStatistics(
 const EDepthMap8& sourceMap,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average,
 float& stddev
 )

void ComputeStatistics(
 const EDepthMap8& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average,
 float& stddev
 )

void ComputeStatistics(
 const EDepthMap16& sourceMap,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average,
 float& stddev
 )

void ComputeStatistics(
 const EDepthMap16& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average,
 float& stddev
 )
```

```
void ComputeStatistics(
 const EDepthMap32f& sourceMap,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average,
 float& stddev
 )

void ComputeStatistics(
 const EDepthMap32f& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average,
 float& stddev
 )

void ComputeStatistics(
 const EZMap8& sourceMap,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average,
 float& stddev
 )

void ComputeStatistics(
 const EZMap8& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average,
 float& stddev
 )
```

```
void ComputeStatistics(
 const EZMap16& sourceMap,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average,
 float& stddev
 )

void ComputeStatistics(
 const EZMap16& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average,
 float& stddev
 )

void ComputeStatistics(
 const EZMap32f& sourceMap,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average,
 float& stddev
 )

void ComputeStatistics(
 const EZMap32f& sourceMap,
 ERegion& region,
 OEV_UINT32& validCount,
 float& minimumValue,
 float& maximumValue,
 float& average,
 float& stddev
 )
```

## Parameters

*sourceMap*

The input depthmap/Zmap.

*validCount*

Variable to store the number of valid pixels in sourceMap.

*minimumValue*

Variable to store the minimum value.

*maximumValue*

Variable to store the maximum value.

*average*

Variable to store the average value.

*region*

The ERegion where the statistics has to be calculated.

*stddev*

Variable to store the standard deviation value.

# 4.185. EStringPair Class

Represent a Key/Value pair of strings.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EStringPair | Constructs an EStringPair context. |
| GetKey | Returns the key. |
| GetValue | Returns the value. |
| operator= | Assignment operator |

# EStringPair::EStringPair

Constructs an EStringPair context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EStringPair(
 std::string key,
 std::string value
 )
void EStringPair(
 const EStringPair& other
 )
void EStringPair(
 )
```

### Parameters

*key*

The key.

*value*

The value associated to the key.

*other*

-

# EStringPair::GetKey

Returns the key.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::string GetKey()
```

# EStringPair::operator=

Assignment operator

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EStringPair& operator=(
 const EStringPair& other
 )
```

## Parameters

*other*

-

# EStringPair::GetValue

Returns the value.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
std::string GetValue()
```

# 4.186. EThreeLayersImageSegmenter Class

The base class from which all the segmenters that produce three layers derive.

### Remarks

The Layer Encoding can be enabled/disabled for each layer individually. The index of the layers in the coded image can also be assigned individually.

**Base Class:** EImageSegmenter

**Derived Class(es):** EGrayscaleDoubleThresholdSegmenter

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

## Methods

| | |
|---|---|
| GetBlackLayerEncoded | Black layer encoding status. |
| GetBlackLayerIndex | Index of the black layer in the destination coded image. |
| GetNeutralLayerEncoded | Neutral layer encoding status. |
| GetNeutralLayerIndex | Index of the neutral layer in the destination coded image. |
| GetWhiteLayerEncoded | White layer encoding status. |
| GetWhiteLayerIndex | Index of the white layer in the destination coded image. |
| SetBlackLayerEncoded | Black layer encoding status. |

| | |
|---|---|
| SetBlackLayerIndex | Index of the black layer in the destination coded image. |
| SetNeutralLayerEncoded | Neutral layer encoding status. |
| SetNeutralLayerIndex | Index of the neutral layer in the destination coded image. |
| SetWhiteLayerEncoded | White layer encoding status. |
| SetWhiteLayerIndex | Index of the white layer in the destination coded image. |

# EThreeLayersImageSegmenter::GetBlackLayerEncoded

# EThreeLayersImageSegmenter::SetBlackLayerEncoded

Black layer encoding status.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
bool GetBlackLayerEncoded() const
void SetBlackLayerEncoded(bool encode)
```

# EThreeLayersImageSegmenter::GetBlackLayerIndex

# EThreeLayersImageSegmenter::SetBlackLayerIndex

Index of the black layer in the destination coded image.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
OEV_UINT32 GetBlackLayerIndex() const
void SetBlackLayerIndex(OEV_UINT32 index)
```

## Remarks

Setting this property automatically switches on the encoding of the black layer.

# EThreeLayersImageSegmenter::GetNeutralLayerEncoded

# EThreeLayersImageSegmenter::SetNeutralLayerEncoded

Neutral layer encoding status.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
bool GetNeutralLayerEncoded() const
void SetNeutralLayerEncoded(bool encode)
```

# EThreeLayersImageSegmenter::GetNeutralLayerIndex

# EThreeLayersImageSegmenter::SetNeutralLayerIndex

Index of the neutral layer in the destination coded image.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
OEV_UINT32 GetNeutralLayerIndex() const
void SetNeutralLayerIndex(OEV_UINT32 index)
```

## Remarks

Setting this property automatically switches on the encoding of the neutral layer.

# EThreeLayersImageSegmenter::GetWhiteLayerEncoded

# EThreeLayersImageSegmenter::SetWhiteLayerEncoded

White layer encoding status.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
bool GetWhiteLayerEncoded() const
void SetWhiteLayerEncoded(bool encode)
```

# EThreeLayersImageSegmenter::GetWhiteLayerIndex

# EThreeLayersImageSegmenter::SetWhiteLayerIndex

Index of the white layer in the destination coded image.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
OEV_UINT32 GetWhiteLayerIndex() const
void SetWhiteLayerIndex(OEV_UINT32 index)
```

### Remarks

Setting this property automatically switches on the encoding of the white layer.

# 4.187. ETwoLayersImageSegmenter Class

The base class from which all the segmenters that produce two layers derive.

### Remarks

The Layer Encoding can be enabled/disabled for each layer individually. The index of the layers in the coded image can also be assigned individually.

**Base Class:** EImageSegmenter

**Derived Class(es):** EBinaryImageSegmenter EColorRangeThresholdSegmenter EColorSingleThresholdSegmenter EGrayscaleSingleThresholdSegmenter EImageRangeSegmenter EReferenceImageSegmenter

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

## Methods

| | |
|---|---|
| GetBlackLayerEncoded | Black layer encoding status. |
| GetBlackLayerIndex | Index of the black layer in the destination coded image. |
| GetWhiteLayerEncoded | White layer encoding status. |
| GetWhiteLayerIndex | Index of the white layer in the destination coded image. |
| SetBlackLayerEncoded | Black layer encoding status. |
| SetBlackLayerIndex | Index of the black layer in the destination coded image. |
| SetWhiteLayerEncoded | White layer encoding status. |
| SetWhiteLayerIndex | Index of the white layer in the destination coded image. |

# ETwoLayersImageSegmenter::GetBlackLayerEncoded

# ETwoLayersImageSegmenter::SetBlackLayerEncoded

Black layer encoding status.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
bool GetBlackLayerEncoded() const
void SetBlackLayerEncoded(bool encode)
```

# ETwoLayersImageSegmenter::GetBlackLayerIndex

# ETwoLayersImageSegmenter::SetBlackLayerIndex

Index of the black layer in the destination coded image.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
OEV_UINT32 GetBlackLayerIndex() const
void SetBlackLayerIndex(OEV_UINT32 index)
```

## Remarks

Setting this property automatically switches on the encoding of the black layer.

# ETwoLayersImageSegmenter::GetWhiteLayerEncoded

# ETwoLayersImageSegmenter::SetWhiteLayerEncoded

White layer encoding status.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
bool GetWhiteLayerEncoded() const
void SetWhiteLayerEncoded(bool encode)
```

## ETwoLayersImageSegmenter::GetWhiteLayerIndex

## ETwoLayersImageSegmenter::SetWhiteLayerIndex

Index of the white layer in the destination coded image.

**Namespace:** Euresys::Open_eVision_2_10::Segmenters

```
[C++]
OEV_UINT32 GetWhiteLayerIndex() const
void SetWhiteLayerIndex(OEV_UINT32 index)
```

### Remarks

Setting this property automatically switches on the encoding of the white layer.

# 4.188. EUnwarpingLut Class

This class is used only as a lookup table in the EWorldShape::Unwarp and EWorldShape::SetupUnwarp methods. It has no other use of its own.

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| EUnwarpingLut | Constructs a EUnwarpingLut object that is used to speed-up the unwarping process. |

U

# nwarpingLut::EUnwarpingLut

Constructs a EUnwarpingLut object that is used to speed-up the unwarping process.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EUnwarpingLut(
 const EUnwarpingLut& other
 )
void EUnwarpingLut(
 )
```

### Parameters

*other*

-

# 4.189. EVector Class

Base class for all typed vectors.

### Remarks

This class contains all methods that are not type specific. Mainly methods to handle elements count and serialization

**Derived Class(es):** EBW32Vector EPathVector EBW16PathVector EBW16Vector EBW8PathVector EBW8Vector EBWHistogramVector EC24PathVector EC24Vector EColorVector EPeakVector

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Empty | Resets the number of elements to **0**. |
| GetNumElements | Number of elements in the vector. |
| RemoveElement | Removes the element at the specified position |
| Serialize | - |
| SetNumElements | Number of elements in the vector. |

E

V

# ector::Empty

Resets the number of elements to **0**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Empty(
 )
```

# EVector::GetNumElements

# EVector::SetNumElements

Number of elements in the vector.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumElements() const
void SetNumElements(OEV_UINT32 un32NumElements)
```

# EVector::RemoveElement

Removes the element at the specified position

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveElement(
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index, between **0** and EVector::NumElements (excluded) of the element to be accessed.

# EVector::Serialize

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Serialize(
 ESerializer* serializer,
 OEV_UINT32 un32Version
 )
```

### Parameters

*serializer*

-

*un32Version*

-

# 4.190. EWedge Class

Represents a model of a wedge (disk, ring, sector or curvilinear quadrilateral) in EasyGauge.

**Base Class:** EFrame

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| CopyTo | Copies all the data of the current EWedge object into another EWedge object and returns it. |

| | |
|---|---|
| EWedge | Constructs a EWedge object. |
| GetAmplitude | Angular amplitude of the EWedge. |
| GetApexAngle | Angular position at the apex of the EWedge. |
| GetBreadth | Breadth of the EWedge. |
| GetCorners | Retrieves the coordinates of each corner of the EWedge. |
| GetDirect | Flag indicating whether positive angles correspond to a clockwise or an anticlockwise rotation. |
| GetEdges | Retrieves each edge of the EWedge. |
| GetEndAngle | Ending angular position of the EWedge. |
| GetFullBreadth | Flag indicating if the EWedge has a full breadth (**breadth = radius**). |
| GetFullCircle | Flag indicating if the EWedge is a full circle (**amplitude = 2 PI**). |
| GetInnerApex | Inner apex point coordinates of the EWedge. |

| | |
|---|---|
| GetInnerArcLength | Inner circle arc length of the EWedge. |
| GetInnerDiameter | Inner diameter of the EWedge. |
| GetInnerEnd | Inner end point coordinates of the EWedge. |
| GetInnerOrg | Inner origin point coordinates of the EWedge. |
| GetInnerPoint | Returns the coordinates of a particular point specified by its location along the inner circle arc. |
| GetInnerRadius | Inner radius of the EWedge |
| GetMidEdges | Retrieves the center coordinates of each edge of the wedge fitting gauge. |
| GetOrgAngle | Angular position from where the EWedge extents. |
| GetOuterApex | Outer apex point coordinates of the EWedge. |
| GetOuterArcLength | Outer circle arc length of the EWedge. |
| GetOuterDiameter | Outer diameter of the EWedge. |

| | |
|---|---|
| GetOuterEnd | Outer end point coordinates of the EWedge. |
| GetOuterOrg | Outer origin point coordinates of the EWedge. |
| GetOuterPoint | Returns the coordinates of a particular point specified by its location along the outter circle arc. |
| GetOuterRadius | Outer radius of the EWedge. |
| GetPoint | Returns the coordinates of a particular point specified by its location along the wedge perimeter. |
| operator= | Copies all the data from another EWedge object into the current EWedge object |
| SetAmplitude | Angular amplitude of the EWedge. |
| SetDiameters | Sets the nominal inner/outer diameter and breadth of the EWedge. |
| SetFromCenterAndOrigin | Sets the geometric parameters (nominal position and diameter, breadth, angular position and amplitude) of an EWedge object. |
| SetFromOriginMiddleEnd | Sets the geometric parameters (nominal position and diameter, breadth, angular position and amplitude) of an EWedge object. |

| | |
|---|---|
| SetFromTwoPoints | DEPRECATED (you should use EWedge::SetFromCenterAndOrigin): Sets the geometric parameters (nominal position and diameter, breadth, angular position and amplitude) of an EWedge object. |
| SetRadii | Sets the nominal radius and breadth of the EWedge. |

# EWedge::GetAmplitude

# EWedge::SetAmplitude

Angular amplitude of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetAmplitude() const
void SetAmplitude(float ampl)
```

## Remarks

A EWedge is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.
The sign of the rotation angle depends whether the field of view is calibrated or not.
* When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value.
* When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# EWedge::GetApexAngle

Angular position at the apex of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetApexAngle() const
```

## Remarks

A EWedge is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.
The sign of the rotation angle depends whether the field of view is calibrated or not.
* When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value.
* When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# EWedge::GetBreadth

Breadth of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetBreadth() const
```

## Remarks

A EWedge is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.

# EWedge::CopyTo

Copies all the data of the current EWedge object into another EWedge object and returns it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EWedge* CopyTo(
 EWedge* destinationImage
 )
```

### Parameters

*destinationImage*

Pointer to the EWedge object in which the current EWedge object data have to be copied.

### Remarks

In case of a **NULL** pointer, a new EWedge object will be created and returned.

# EWedge::GetDirect

Flag indicating whether positive angles correspond to a clockwise or an anticlockwise rotation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetDirect() const
```

### Remarks

**TRUE** (default) means that angles increase anticlockwisely in a direct coordinate system, and clockwisely in an inverse coordinate system.
* When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards.
* When the field of view is not calibrated, the coordinate system is said to be inverse the abscissa extends rightwards and the ordinate extends downwards.

# EWedge::GetEndAngle

Ending angular position of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetEndAngle() const
```

### Remarks

A EWedge is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.
The sign of the rotation angle depends whether the field of view is calibrated or not.
* When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value.
* When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# EWedge::EWedge

Constructs a EWedge object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EWedge(
 )
void EWedge(
 const EPoint& center,
 float diameter,
 float breadth,
 float originAngle,
 BOOL direct
 )
void EWedge(
 const EPoint& center,
 const EPoint& origin,
 float breadth,
 BOOL direct
 )
void EWedge(
 const EPoint& center,
 float diameter,
 float breadth,
 float originAngle,
 float amplitude
 )
void EWedge(
 const EPoint& origin,
 const EPoint& middle,
 const EPoint& end,
 float breadth,
 BOOL fullCircle
 )
void EWedge(
 const EWedge& other
 )
```

## Parameters

*center*

Center coordinates of the wedge at its nominal position. The default value is **(0,0)**.

*diameter*

Nominal diameter of the wedge. The default value is **100**.

*breadth*

Nominal breadth of the wedge. It must be negative or zero. The default value is **-50**.

*originAngle*

Origin point coordinates of the wedge.

*direct*

**TRUE** (default) means that angles increase anticlockwise in a direct coordinate system and clockwise in an inverse coordinate system.

*origin*

Origin point coordinates of the wedge.

*amplitude*

Nominal angular amplitude of the wedge. The default value is **360**.

*middle*

Middle point coordinates of the wedge.

*end*

End point coordinates of the wedge.

*fullCircle*

**TRUE** (default) in case of a full turn wedge. If **fullCircle** is **FALSE**, **origin** and **end** give the wedge's amplitude.

*other*

Another EWedge object to be copied in the new EWedge object.

# EWedge::GetFullBreadth

Flag indicating if the EWedge has a full breadth (**breadth = radius**).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetFullBreadth() const
```

# EWedge::GetFullCircle

Flag indicating if the EWedge is a full circle (**amplitude = 2 PI**).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetFullCircle() const
```

# EWedge::GetCorners

Retrieves the coordinates of each corner of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetCorners(
 EPoint& ar,
 EPoint& AAr,
 EPoint& aRR,
 EPoint& AARR
 )
```

### Parameters

*ar*

Coordinates of the inner org corner of the EWedge.

*AAr*

Coordinates of the inner end corner of the EWedge.

*aRR*

Coordinates of the outer org corner of the EWedge.

*AARR*

Coordinates of the outer end corner of the EWedge.

# EWedge::GetEdges

Retrieves each edge of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetEdges(
 ELine& a,
 ELine& AA,
 ECircle& r,
 ECircle& RR
 )
```

### Parameters

*a*

Org edge of the EWedge.

*AA*

End edge of the EWedge.

*r*

Inner edge of the EWedge.

*RR*

Outer edge of the EWedge.

# EWedge::GetInnerPoint

Returns the coordinates of a particular point specified by its location along the inner circle arc.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetInnerPoint(
 float fraction
 )
```

### Parameters

*fraction*

Point location expressed as a fraction of the circle arc (range **[-1, +1]**).

# EWedge::GetMidEdges

Retrieves the center coordinates of each edge of the wedge fitting gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetMidEdges(
 EPoint& a,
 EPoint& AA,
 EPoint& r,
 EPoint& RR
 )
```

### Parameters

*a*

Center coordinates of the org edge of the EWedge.

*AA*

Center coordinates of the end edge of the EWedge.

*r*

Center coordinates of the inner edge of the EWedge.

*RR*

Center coordinates of the outer edge of the EWedge.

# EWedge::GetOuterPoint

Returns the coordinates of a particular point specified by its location along the outter circle arc.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetOuterPoint(
 float fraction
 )
```

### Parameters

*fraction*

Point location expressed as a fraction of the circle arc (range **[-1, +1]**).

# EWedge::GetPoint

Returns the coordinates of a particular point specified by its location along the wedge perimeter.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetPoint(
 float breadthFraction,
 float angleFraction
 )
```

### Parameters

*breadthFraction*

Point location expressed as a fraction of the wedge breadth (range -1, +1).

*angleFraction*

Point location expressed as a fraction of the wedge amplitude (range -1, +1).

# EWedge::GetInnerApex

Inner apex point coordinates of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetInnerApex() const
```

# EWedge::GetInnerArcLength

Inner circle arc length of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetInnerArcLength() const
```

# EWedge::GetInnerDiameter

Inner diameter of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetInnerDiameter() const
```

# EWedge::GetInnerEnd

Inner end point coordinates of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetInnerEnd() const
```

# EWedge::GetInnerOrg

Inner origin point coordinates of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetInnerOrg() const
```

# EWedge::GetInnerRadius

Inner radius of the EWedge

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetInnerRadius() const
```

# EWedge::operator=

Copies all the data from another EWedge object into the current EWedge object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EWedge& operator=(
 const EWedge& other
 )
```

### Parameters

*other*

EWedge object to be copied

# EWedge::GetOrgAngle

Angular position from where the EWedge extents.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetOrgAngle() const
```

### Remarks

A EWedge is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.
The sign of the rotation angle depends whether the field of view is calibrated or not.
* When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value.
* When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# EWedge::GetOuterApex

Outer apex point coordinates of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetOuterApex() const
```

### Remarks

A EWedge is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.

# EWedge::GetOuterArcLength

Outer circle arc length of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetOuterArcLength() const
```

### Remarks

A EWedge is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.

# EWedge::GetOuterDiameter

Outer diameter of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
float GetOuterDiameter() const
```

### Remarks

A EWedge is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.

# EWedge::GetOuterEnd

Outer end point coordinates of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetOuterEnd() const
```

### Remarks

A EWedge is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.

# EWedge::GetOuterOrg

Outer origin point coordinates of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetOuterOrg() const
```

### Remarks

A EWedge is fully defined knowing its nominal position (its center coordinates), its nominal outer radius (diameter), its breadth (must be negative), the angular position from where it extents, its angular amplitude, and its outline tolerance.

# EWedge::GetOuterRadius

Outer radius of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetOuterRadius() const
```

### Remarks

A EWedge is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.

# EWedge::SetDiameters

Sets the nominal inner/outer diameter and breadth of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetDiameters(
 float diameter,
 float breadth
 )
```

### Parameters

*diameter*

Outer diameter of the EWedge. The default value is **100**.

*breadth*

Breadth of the EWedge. It must be negative or zero. Its default value is **-50**.

### Remarks

A EWedge is fully defined knowing its nominal position (given by the coordinates of its center), its nominal outer radius (diameter), its breadth, the angular position from where it extents, its angular amplitude and its outline tolerance.
By default, the EWedge diameter value is **100**, which means 100 pixels when the field of view is not calibrated and 100 "units" in case of a calibrated field of view.

# EWedge::SetFromCenterAndOrigin

Sets the geometric parameters (nominal position and diameter, breadth, angular position and amplitude) of an EWedge object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromCenterAndOrigin(
 const EPoint& center,
 const EPoint& origin,
 float breadth,
 BOOL direct
 )
```

### Parameters

*center*

Center coordinates of the wedge at its nominal position. The default value is **(0,0)**.

*origin*

Origin point coordinates of the wedge.

*breadth*

Nominal breadth of the wedge. It must be negative or zero. The default value is **-50**.

*direct*

**TRUE** (default) means that angles increase anticlockwise in a direct coordinate system and clockwise in an inverse coordinate system.

### Remarks

In a direct coordinate system, the abscissa extends rightwards and the ordinate extends upwards. The coordinate system is said to be inverse if the abscissa extends rightwards and the ordinate extends downwards.

# EWedge::SetFromOriginMiddleEnd

Sets the geometric parameters (nominal position and diameter, breadth, angular position and amplitude) of an EWedge object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromOriginMiddleEnd(
 const EPoint& origin,
 const EPoint& middle,
 const EPoint& end,
 float breadth,
 BOOL fullCircle
 )
```

### Parameters

*origin*

Origin point coordinates of the wedge.

*middle*

Middle point coordinates of the wedge.

*end*

End point coordinates of the wedge.

*breadth*

Nominal breadth of the wedge. It must be negative or zero. The default value is **-50**.

*fullCircle*

**TRUE** (default) in case of a full turn wedge. If **fullCircle** is **FALSE**, **origin** and **end** give the wedge's amplitude.

### Remarks

In a direct coordinate system, the abscissa extends rightwards and the ordinate extends upwards. The coordinate system is said to be inverse if the abscissa extends rightwards and the ordinate extends downwards.

# EWedge::SetFromTwoPoints

DEPRECATED (you should use EWedge::SetFromCenterAndOrigin): Sets the geometric parameters (nominal position and diameter, breadth, angular position and amplitude) of an EWedge object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromTwoPoints(
 const EPoint& center,
 const EPoint& origin,
 float breadth,
 BOOL direct
 )
```

## Parameters

*center*

Center coordinates of the wedge at its nominal position. The default value is **(0,0)**.

*origin*

Origin point coordinates of the wedge.

*breadth*

Nominal breadth of the wedge. It must be negative or zero. The default value is **-50**.

*direct*

**TRUE** (default) means that angles increase anticlockwise in a direct coordinate system and clockwise in an inverse coordinate system.

## Remarks

In a direct coordinate system, the abscissa extends rightwards and the ordinate extends upwards. The coordinate system is said to be inverse if the abscissa extends rightwards and the ordinate extends downwards.

# EWedge::SetRadii

Sets the nominal radius and breadth of the EWedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetRadii(
 float radius,
 float breadth
 )
```

### Parameters

*radius*

Outer radius of the EWedge. The default value is **50**.

*breadth*

Breadth of the EWedge, which must be negative or zero. Its default value is **-50**.

### Remarks

A EWedge is fully defined knowing its nominal position (given by the coordinates of its center), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude and its outline tolerance. By default, the EWedge radius value is **50**, which means 50 pixels when the field of view is not calibrated and 50 "units" in case of a calibrated field of view.

# 4.191. EWedgeGauge Class

Manages a wedge fitting gauge.

**Base Class:** EWedgeShape

**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddSkipRange | Adds an item to the set of skip ranges and returns the index of the newly added range. |

| | |
|---|---|
| CopyTo | Copies all the data of the current EWedgeGauge object into another EWedgeGauge object, and returns it. |
| Drag | Moves a handle to a new position and updates the position parameters of the gauge. |
| Draw | Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode. |
| DrawWithCurrentPen | Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode. |
| EWedgeGauge | Constructs a wedge measurement context. |
| GetActiveEdges | Active edges as defined in EDragHandle. |
| GetAverageDistance | Average distance between the sampled points and the fitted model. |
| GetFilteringThreshold | Relative filtering threshold, that is the fraction of the average distance between the sampled points and the fitted model above which a point is filtered out. |
| GetHVConstraint | Status of the restriction on the orientation of the point location gauge or model fitting sample paths. |

| | |
|---|---|
| GetMeasuredPoint | Returns the coordinates of a sample point, measured along one of the sample paths of the gauge. |
| GetMeasuredWedge | Information pertaining to the fitted wedge. |
| GetMinAmplitude | Offset added to the **Threshold** when a peak is to be detected. |
| GetMinArea | Minimum area value. |
| GetMinNumFitSamples | Returns the minimum number of samples required for fitting on each side of the shape. |
| GetNumFilteringPasses | Number of filtering passes for a model fitting operation. |
| GetNumSamples | Number of sampled points during the model fitting operation. |
| GetNumSamplesa | Number of sampled points found on edge a during the measure operation. |
| GetNumSamplesA | Number of sampled points found on edge A during the measure operation. |
| GetNumSamplesr | Number of sampled points found on edge r during the measure operation. |

| | |
|---|---|
| GetNumSamplesR | Number of sampled points found on edge R during the measure operation. |
| GetNumSkipRanges | Number of skip ranges in the gauge after a call to EWedgeGauge::AddSkipRange. |
| GetNumValidSamples | Number of valid sample points remaining after a model fitting operation. |
| GetRectangularSamplingArea | Flag indicating whether the sampling area remains a rectangle when rotated, instead of becoming a parallelogram. |
| GetSamplea | Allows to retrieve information on the samples found along the a edge. |
| GetSampleA | Allows to retrieve information on the samples found along the A edge. |
| GetSampler | Allows to retrieve information on the samples found along the r edge. |
| GetSampleR | Allows to retrieve information on the samples found along the R edge. |
| GetSamplingStep | Approximate distance between sampled points during a model fitting operation. |

| | |
|---|---|
| GetSkipRange | Allows to retrieve the start and end values of the skip range corresponding to the given index, if it is valid (i.e. if it has previously been created by a call to the EWedgeGauge::AddSkipRange method. |
| GetSmoothing | Number of pixels used for the low-pass filtering operation. |
| GetThickness | Number of parallel segments used to extract the data profile. |
| GetThreshold | Threshold level used to delimit significant peaks in the data profile. |
| GetTolerance | Searching area half thickness of the wedge fitting gauge. |
| GetTransitionChoice | Transition choice. |
| GetTransitionIndex | Index (from **0** on) of the transition to be retained when the transition choice parameter is set to ETransitionChoice_NthFromBegin or ETransitionChoice_NthFromEnd. |
| GetTransitionType | Transition type. |
| GetType | Shape type. |
| GetValid | Flag indicating if at least one valid transition has been found. |

| | |
|---|---|
| HitTest | Checks whether the cursor is positioned over a handle (**TRUE**) or not (**FALSE**). |
| Measure | Triggers the point location or the model fitting operation. |
| MeasureSample | Computes the sample points along the sample path whose index in the list is given by the **pathIndex** parameter. |
| MeasureWithoutFitting | Triggers the point location without wedge fitting operation. |
| operator= | Copies all the data from another EWedgeGauge object into the current EWedgeGauge object |
| Plot | Draws the profile that is crossed by one of the sample paths of the gauge, as defined by EPlotItem. |
| PlotWithCurrentPen | Draws the profile that is crossed by one of the sample paths of the gauge, as defined by EPlotItem. |
| Process | Triggers the process pertaining to a shape or gauge and all the daughter gauges attach to it. |
| RemoveAllSkipRanges | Removes all the skip ranges previously created by a call to EWedgeGauge::AddSkipRange. |

| | |
|---|---|
| RemoveSkipRange | After a call to EWedgeGauge::AddSkipRange, removes the skip range with the given index. |
| SetActive | Sets the flag indicating whether the gauge is active or not. |
| SetActiveEdges | Active edges as defined in EDragHandle. |
| SetDiameters | Sets the nominal inner/outer diameter and breadth of the EWedgeGauge. |
| SetFilteringThreshold | Relative filtering threshold, that is the fraction of the average distance between the sampled points and the fitted model above which a point is filtered out. |
| SetFromOriginMiddleEnd | Sets the geometric parameters (nominal position and diameter, breadth, angular position and amplitude) of an EWedgeGauge object. |
| SetFromTwoPoints | DEPRECATED (you should use EWedgeGauge): Sets the geometric parameters (nominal position and diameter, breadth, angular position and amplitude) of an EWedgeGauge object. |
| SetHVConstraint | Status of the restriction on the orientation of the point location gauge or model fitting sample paths. |
| SetMinAmplitude | Offset added to the **Threshold** when a peak is to be detected. |

| | |
|---|---|
| SetMinArea | Minimum area value. |
| SetMinNumFitSamples | Sets the minimum number of samples required for fitting on each side of the shape. |
| SetNumFilteringPasses | Number of filtering passes for a model fitting operation. |
| SetRadii | Sets the nominal radius and breadth of the EWedgeGauge. |
| SetRectangularSamplingArea | Flag indicating whether the sampling area remains a rectangle when rotated, instead of becoming a parallelogram. |
| SetSamplingStep | Approximate distance between sampled points during a model fitting operation. |
| SetSmoothing | Number of pixels used for the low-pass filtering operation. |
| SetThickness | Number of parallel segments used to extract the data profile. |
| SetThreshold | Threshold level used to delimit significant peaks in the data profile. |
| SetTolerance | Searching area half thickness of the wedge fitting gauge. |
| SetTransitionChoice | Transition choice. |

| | |
|---|---|
| SetTransitionIndex | Index (from **0** on) of the transition to be retained when the transition choice parameter is set to ETransitionChoice_NthFromBegin or ETransitionChoice_NthFromEnd. |
| SetTransitionType | Transition type. |
| SetWedge | |

# EWedgeGauge::SetActive

Sets the nominal position (center coordinates), diameter, breadth, angular origin and amplitude of the wedge fitting gauge according

Sets the flag indicating whether the gauge is active or not.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetActive(BOOL active)
```

## Remarks

When complex gauging is required, several gauges can be grouped together. Applying EWedgeGauge::Process to the mother gauge or shape triggers the measurement of the whole. Only the active gauges will participate in the process. By default, the gauge is active (**TRUE**).

# EWedgeGauge::GetActiveEdges

# EWedgeGauge::SetActiveEdges

Active edges as defined in EDragHandle.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetActiveEdges()
void SetActiveEdges(OEV_UINT32 un32ActiveEdges)
```

### Remarks

In the case of a wedge fitting gauge, each edge can have its own transition detection parameters. Updating the transition parameters only affect the current active edges. By default, all edges are active.

# EWedgeGauge::AddSkipRange

Adds an item to the set of skip ranges and returns the index of the newly added range.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 AddSkipRange(
 const OEV_UINT32 start,
 const OEV_UINT32 end
 )
```

### Parameters

*start*

Beginning of the skip range.

*end*

End of the skip range.

### Remarks

The samples indices between start and end (including the boundaries) will be discarded during the measurement process.
The EWedgeGauge::AddSkipRange method allows to define skip ranges in an EWedgeGauge. This means that, at measure time, samples belonging to these ranges will not be taken into account.
A sample may belong to more than one skip range; to be discarded, a sample has to pertain to at least one range.
Moreover, the skip ranges are allowed to overlap one another.
The range is allowed to be reversed (i.e. end is not required to be greater than start).
Also, start and end are not required to reference valid indices at the time of the call (i.e. the range may lie outside of the current return value for EWedgeGauge::NumSamples).

# EWedgeGauge::GetAverageDistance

Average distance between the sampled points and the fitted model.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetAverageDistance()
```

### Remarks

Irrelevant in case of a point location operation.

# EWedgeGauge::CopyTo

Copies all the data of the current EWedgeGauge object into another EWedgeGauge object, and returns it.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EWedgeGauge* CopyTo(
 EWedgeGauge* other,
 BOOL recursive
 )
```

### Parameters

*other*

Pointer to the EWedgeGauge object in which the current EWedgeGauge object data have to be copied.

*recursive*

**TRUE** if the children gauges have to be copied as well, **FALSE** otherwise.

### Remarks

In case of a **NULL** pointer, a new EWedgeGauge object will be created and returned.

# EWedgeGauge::Drag

Moves a handle to a new position and updates the position parameters of the gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Cursor current X coordinate.

*y*

Cursor current Y coordinate.

# EWedgeGauge::Draw

Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
void Draw(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

*color*

The color in which to draw the overlay.

# EWedgeGauge::DrawWithCurrentPen

Draws a graphical representation of a point location or model fitting gauge, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

# EWedgeGauge::EWedgeGauge

Constructs a wedge measurement context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EWedgeGauge(
 )
```

```
void EWedgeGauge(
 const EWedgeGauge& other
 )
```

### Parameters

*other*

Another EWedgeGauge object to be copied in the new EWedgeGauge object.

### Remarks

With the default constructor, all the parameters are initialized to their respective default values. With the copy constructor, the constructed wedge measurement context is based on a pre-existing EWedgeGauge object. The gauge children are also copied. Hierarchy copying through a copy constructor is always recursive. To disable this recursion, use instead the EWedgeGauge::CopyTo method.

# EWedgeGauge::GetFilteringThreshold

# EWedgeGauge::SetFilteringThreshold

Relative filtering threshold, that is the fraction of the average distance between the sampled points and the fitted model above which a point is filtered out.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetFilteringThreshold()
void SetFilteringThreshold(float f32FilteringThreshold)
```

### Remarks

Irrelevant in case of a point location operation. During a model fitting operation, the "filtering" process can be invoked to remove outliers, i.e. points that were located significantly far away from the fitted model (so that their position is dubious).

# EWedgeGauge::GetMeasuredPoint

Returns the coordinates of a sample point, measured along one of the sample paths of the gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetMeasuredPoint(
 OEV_UINT32 index
 )
```

## Parameters

*index*

This argument must be left unchanged from its default value, i.e. **~0** (**= 0xFFFFFFFF**).

## Remarks

These coordinates pertain to the world space; they are expressed in the reference frame to which the current EWedgeGauge object belongs. The gauging process uses a list of sample points to find the shape position and size that best fit a given image. These sample points are measured along the sample paths defined by the gauge geometry. EWedgeGauge::GetMeasuredPoint returns the coordinates of the sample point that meets the following two requirements: 1. It lies on the sample path inspected with the last call to EWedgeGauge::MeasureSample, and 1. Among all the sample points along the latter sample path, it is the one selected by the EWedgeGauge::TransitionChoice property.

**Note.** For this method to succeed, it is necessary to previously call EWedgeGauge::MeasureSample.

# EWedgeGauge::GetMinNumFitSamples

Returns the minimum number of samples required for fitting on each side of the shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetMinNumFitSamples(
 OEV_INT32& side0,
 OEV_INT32& side1,
 OEV_INT32& side2,
 OEV_INT32& side3
 )
```

### Parameters

*side0*

Minimum number of samples on the top side of the rectangle.

*side1*

Minimum number of samples on the left side of the rectangle.

*side2*

Minimum number of samples on the bottom side of the rectangle.

*side3*

Minimum number of samples on the right side of the rectangle.

### Remarks

Irrelevant in case of a point location operation.

# EWedgeGauge::GetSampleA

Allows to retrieve information on the samples found along the A edge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetSampleA(
 EPoint& pt,
 OEV_UINT32 index
 )
```

```
void GetSampleA(
 ESamplePoint& sp,
 OEV_UINT32 index
 )

BOOL GetSampleA(
 EPeak& pk,
 OEV_UINT32 index
 )
```

### Parameters

*pt*

EPoint structure that will receive the sample position.

*index*

The sample index

*sp*

ESamplePoint structure that will receive the sample position.

*pk*

EPeak structure that will contain the sample peak properties.

### Remarks

The method provides the sample point corresponding to the supplied index.
The returned value is true when the sample is valid, and false otherwise.

# EWedgeGauge::GetSampleA

Allows to retrieve information on the samples found along the A edge.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetSampleA(
 EPoint& pt,
 OEV_UINT32 index
 )
```

```
void GetSampleA(
 ESamplePoint& sp,
 OEV_UINT32 index
 )

BOOL GetSampleA(
 EPeak& pk,
 OEV_UINT32 index
 )
```

### Parameters

*pt*

EPoint structure that will receive the sample position.

*index*

The sample index

*sp*

ESamplePoint structure that will receive the sample position.

*pk*

EPeak structure that will contain the sample peak properties.

### Remarks

The method provides the sample point corresponding to the supplied index.
The returned value is true when the sample is valid, and false otherwise.

# EWedgeGauge::GetSampleR

Allows to retrieve information on the samples found along the R edge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetSampleR(
 EPoint& pt,
 OEV_UINT32 index
 )
```

```
void GetSampleR(
 ESamplePoint& sp,
 OEV_UINT32 index
 )

BOOL GetSampleR(
 EPeak& pk,
 OEV_UINT32 index
 )
```

### Parameters

*pt*

EPoint structure that will receive the sample position.

*index*

The sample index

*sp*

ESamplePoint structure that will receive the sample position.

*pk*

EPeak structure that will contain the sample peak properties.

### Remarks

The method provides the sample point corresponding to the supplied index.
The returned value is true when the sample is valid, and false otherwise.

# EWedgeGauge::GetSampleR

Allows to retrieve information on the samples found along the R edge.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetSampleR(
 EPoint& pt,
 OEV_UINT32 index
 )
```

```
void GetSampleR(
 ESamplePoint& sp,
 OEV_UINT32 index
 )

BOOL GetSampleR(
 EPeak& pk,
 OEV_UINT32 index
 )
```

### Parameters

*pt*

EPoint structure that will receive the sample position.

*index*

The sample index

*sp*

ESamplePoint structure that will receive the sample position.

*pk*

EPeak structure that will contain the sample peak properties.

### Remarks

The method provides the sample point corresponding to the supplied index.
The returned value is true when the sample is valid, and false otherwise.

# EWedgeGauge::GetSkipRange

Allows to retrieve the start and end values of the skip range corresponding to the given index, if it is valid (i.e. if it has previously been created by a call to the EWedgeGauge::AddSkipRange method.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void GetSkipRange(
 const OEV_UINT32 index,
 OEV_UINT32& start,
 OEV_UINT32& end
 )
```

### Parameters

*index*

Index of the skip range.

*start*

Beginning of the skip range.

*end*

End of the skip range.

### Remarks

Start is guaranteed to be smaller or equal to end.

# EWedgeGauge::HitTest

Checks whether the cursor is positioned over a handle (**TRUE**) or not (**FALSE**).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL HitTest(
 BOOL daughters
 )
```

### Parameters

*daughters*

**TRUE** if the daughters gauges handles have to be considered as well.

# EWedgeGauge::GetHVConstraint

# EWedgeGauge::SetHVConstraint

Status of the restriction on the orientation of the point location gauge or model fitting sample paths.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetHVConstraint()
void SetHVConstraint(BOOL bHVConstraint)
```

## Remarks

*Sample paths* are the point location gauges placed along the model to be fitted.

# EWedgeGauge::Measure

Triggers the point location or the model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Measure(
 EROIBW8* sourceImage
 )
```

## Parameters

*sourceImage*

Pointer to the source image.

### Remarks

When this method is called, and if not enough valid points are detected, then the method returns directly, and the measured gauge is set to the nominal parameters.

# EWedgeGauge::GetMeasuredWedge

Information pertaining to the fitted wedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EWedge GetMeasuredWedge()
```

# EWedgeGauge::MeasureSample

Computes the sample points along the sample path whose index in the list is given by the **pathIndex** parameter.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void MeasureSample(
 EROIBW8* sourceImage,
 OEV_UINT32 pathIndex
 )
```

### Parameters

*sourceImage*

Pointer to the source image/ROI.

*pathIndex*

Sample path index.

### Remarks

This method stores its results into a temporary variable inside the EWedgeGauge object.

# EWedgeGauge::MeasureWithoutFitting

Triggers the point location without wedge fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void MeasureWithoutFitting(
 EROIBW8* sourceImage
 )
```

### Parameters

*sourceImage*

Source image.

### Remarks

This method performs the actual measurement for each transition, but does not perform the wedge fitting. This means that individual samples will be available for each edges through the EWedgeGauge::GetSamplea (Edge a), EWedgeGauge::GetSampler (Edge r), EWedgeGauge::GetSampleA (Edge A), EWedgeGauge::GetSampleR (Edge R) methods, but the gauge position will not be changed.
Please note that the filtering will not be performed in this method, since it relies upon the fitting process. The filtering parameters will thus be unused.

# EWedgeGauge::GetMinAmplitude

# EWedgeGauge::SetMinAmplitude

Offset added to the **Threshold** when a peak is to be detected.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetMinAmplitude()
void SetMinAmplitude(OEV_UINT32 un32MinAmplitude)
```

### Remarks

When analyzing a derivative profile, a peak is made up of consecutive pixel values above **Threshold**. To detect weak [strong] transitions, lower [raise] the **Threshold** value.
To avoid interference of noise, an additional parameter is provided. The **MinAmplitude** parameter is an offset added to **Threshold** when a peak is to be detected.
When the pixel values of the derivative profile do not reach **Threshold + MinAmplitude**, the peak is not taken into account. Anyway, when a peak is taken into account, all the pixels with values above **Threshold** are considered (for more accuracy). Setting the **MinAmplitude** value to **0** merely cancels its effect.

# EWedgeGauge::GetMinArea

# EWedgeGauge::SetMinArea

Minimum area value.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetMinArea()
void SetMinArea(OEV_UINT32 un32MinArea)
```

### Remarks

A transition is detected if its derivative peak reaches **Threshold** + **MinAmplitude** value, and then declared valid if the area between the peak curve and the horizontal at level **Threshold** reaches the **MinArea** value.

# EWedgeGauge::GetNumFilteringPasses

# EWedgeGauge::SetNumFilteringPasses

Number of filtering passes for a model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumFilteringPasses()
void SetNumFilteringPasses(OEV_UINT32 un32NumFilteringPasses)
```

### Remarks

Irrelevant in case of a point location operation. During a filtering pass, the points that are too distant from the model are discarded. During a model fitting operation, the "filtering" process can be invoked to remove outliers, i.e. points that were located significantly far away from the fitted model (so that their position is dubious). By default (the number of filtering passes is 0), the outliers rejection process is disabled.

# EWedgeGauge::GetNumSamples

Number of sampled points during the model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumSamples() const
```

### Remarks

Irrelevant in case of a point location operation.
After a model fitting operation, a number of points have been fitted along the model. Among them, some are not reliable because of their **Area** value. Among the remaining ones, some are filtered out (**NumFilteringPasses**, **FilteringThreshold**).

# EWedgeGauge::GetNumSamplesA

Number of sampled points found on edge A during the measure operation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumSamplesA() const
```

# EWedgeGauge::GetNumSamplesA

Number of sampled points found on edge A during the measure operation.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumSamplesA() const
```

# EWedgeGauge::GetNumSamplesR

Number of sampled points found on edge R during the measure operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumSamplesR() const
```

# EWedgeGauge::GetNumSamplesR

Number of sampled points found on edge R during the measure operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumSamplesR() const
```

# EWedgeGauge::GetNumSkipRanges

Number of skip ranges in the gauge after a call to EWedgeGauge::AddSkipRange.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumSkipRanges() const
```

# EWedgeGauge::GetNumValidSamples

Number of valid sample points remaining after a model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetNumValidSamples()
```

### Remarks

Irrelevant in case of a point location operation. After a model fitting operation, a number of points have been fitted along the model. Among them, some are not reliable because of their **Area** value. Among the remaining ones, some are filtered out (**NumFilteringPasses**, **FilteringThreshold**).

# EWedgeGauge::operator=

Copies all the data from another EWedgeGauge object into the current EWedgeGauge object

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EWedgeGauge& operator=(
 const EWedgeGauge& other
 )
```

### Parameters

*other*

EWedgeGauge object to be copied

# EWedgeGauge::Plot

Draws the profile that is crossed by one of the sample paths of the gauge, as defined by EPlotItem.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Plot(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )

void Plot(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )

void Plot(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawItems*

Boolean combination of EPlotItem members, that indicates which items must be displayed.

*width*

Width of the plot.

*height*

Height of the plot.

*originX*

Origin point coordinates of the plot along the X axis.

*originY*

Origin point coordinates of the plot along the Y axis.

*color*

The color in which to draw the overlay.

### Remarks

The sample path that is taken into considered is the one inspected with the last call to EWedgeGauge::MeasureSample.

**Note.** For this method to succeed, it is necessary to previously call EWedgeGauge::MeasureSample.

# EWedgeGauge::PlotWithCurrentPen

Draws the profile that is crossed by one of the sample paths of the gauge, as defined by EPlotItem.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void PlotWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EPlotItem drawItems,
 float width,
 float height,
 float originX,
 float originY
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawItems*

Boolean combination of EPlotItem members, that indicates which items must be displayed.

*width*

Width of the plot.

*height*

Height of the plot.

*originX*

Origin point coordinates of the plot along the X axis.

*originY*

Origin point coordinates of the plot along the Y axis.

### Remarks

The sample path that is taken into considered is the one inspected with the last call to EWedgeGauge::MeasureSample.

**Note.** For this method to succeed, it is necessary to previously call EWedgeGauge::MeasureSample.

# EWedgeGauge::Process

Triggers the process pertaining to a shape or gauge and all the daughter gauges attach to it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Process(
 EROIBW8* sourceImage,
 BOOL daughters
 )
```

### Parameters

*sourceImage*

Pointer to the source image.

*daughters*

Flag indicating whether the daughters shapes inherit of the same behavior.

### Remarks

When complex gauging is required, several gauges can be grouped together. Applying **Process** to the mother gauge or shape triggers the measurement of the whole. Only the active gauges will participate in the process.

# EWedgeGauge::GetRectangularSamplingArea

# EWedgeGauge::SetRectangularSamplingArea

Flag indicating whether the sampling area remains a rectangle when rotated, instead of becoming a parallelogram.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetRectangularSamplingArea()
void SetRectangularSamplingArea(BOOL bRectangularSamplingArea)
```

### Remarks

By default, this flag is set to **TRUE**: the sampling area always remains a rectangle. Setting this property is only useful when the thickness transition parameter is greater than 1. In fact, when thickness transition parameter is equal to 1, rectangle and parallelogram reduce to the same segment.

# EWedgeGauge::RemoveAllSkipRanges

Removes all the skip ranges previously created by a call to EWedgeGauge::AddSkipRange.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveAllSkipRanges(
 )
```

# EWedgeGauge::RemoveSkipRange

After a call to EWedgeGauge::AddSkipRange, removes the skip range with the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void RemoveSkipRange(
 const OEV_UINT32 index
 )
```

### Parameters

*index*

Index of the skip range to remove, as returned by EWedgeGauge::AddSkipRange.

# EWedgeGauge::GetSamplingStep

# EWedgeGauge::SetSamplingStep

Approximate distance between sampled points during a model fitting operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetSamplingStep()
void SetSamplingStep(float f32SamplingStep)
```

### Remarks

Irrelevant in case of a point location operation.
To fit a model, a series of point location operations are performed along the model. The point location gauges spacing is given by the sampling step.
By default, the sampling step is set to **5**, which means 5 pixels when the field of view is not calibrated, and 5 physical units in case of a calibrated field of view.
Be aware that if the sampling step is too large, the number of sampled points along the model will not be sufficient enough to accurately locate it.

# EWedgeGauge::SetDiameters

Sets the nominal inner/outer diameter and breadth of the EWedgeGauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetDiameters(
 float diameter,
 float breadth
 )
```

### Parameters

*diameter*

Outer diameter of the EWedgeGauge. The default value is **100**.

*breadth*

Breadth of the EWedgeGauge. It must be negative or zero. Its default value is **-50**.

### Remarks

A EWedgeGauge is fully defined knowing its nominal position (given by the coordinates of its center), its nominal outer radius (diameter), its breadth, the angular position from where it extents, its angular amplitude and its outline tolerance.

By default, the EWedgeGauge diameter value is **100**, which means 100 pixels when the field of view is not calibrated and 100 "units" in case of a calibrated field of view.

# EWedgeGauge::SetFromOriginMiddleEnd

Sets the geometric parameters (nominal position and diameter, breadth, angular position and amplitude) of an EWedgeGauge object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetFromOriginMiddleEnd(
 const EPoint& origin,
 const EPoint& middle,
 const EPoint& end,
 float breadth,
 BOOL full
 )
```

### Parameters

*origin*

Origin point coordinates of the wedge.

*middle*

Middle point coordinates of the wedge.

*end*

End point coordinates of the wedge.

*breadth*

Nominal breadth of the wedge. It must be negative or zero. The default value is **-50**.

*full*

-

### Remarks

In a direct coordinate system, the abscissa extends rightwards and the ordinate extends upwards. The coordinate system is said to be inverse if the abscissa extends rightwards and the ordinate extends downwards.

# EWedgeGauge::SetFromTwoPoints

DEPRECATED (you should use EWedgeGauge): Sets the geometric parameters (nominal position and diameter, breadth, angular position and amplitude) of an EWedgeGauge object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetFromTwoPoints(
 const EPoint& center,
 const EPoint& origin,
 float breadth,
 BOOL direct
 )
```

### Parameters

*center*

Center coordinates of the wedge at its nominal position. The default value is **(0,0)**.

*origin*

Origin point coordinates of the wedge.

*breadth*

Nominal breadth of the wedge. It must be negative or zero. The default value is **-50**.

*direct*

**TRUE** (default) means that angles increase anticlockwise in a direct coordinate system and clockwise in an inverse coordinate system.

### Remarks

In a direct coordinate system, the abscissa extends rightwards and the ordinate extends upwards. The coordinate system is said to be inverse if the abscissa extends rightwards and the ordinate extends downwards.

# EWedgeGauge::SetMinNumFitSamples

Sets the minimum number of samples required for fitting on each side of the shape.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetMinNumFitSamples(
 OEV_INT32 side0,
 OEV_INT32 side1,
 OEV_INT32 side2,
 OEV_INT32 side3
 )
```

### Parameters

*side0*

Minimum number of samples on the *outer circle* of the wedge. The default value is **3**.

*side1*

Minimum number of samples on the *original border* of the wedge. If this value is not specified, it is equal to **n32Side0**. The default value is **2**.

*side2*

Minimum number of samples on the *inner circle* of the wedge. If this value is not specified, it is equal to **n32Side0**. The default value is **3**.

*side3*

Minimum number of samples on the *end border* of the wedge. If this value is not specified, it is equal to **n32Side1**. The default value is **2**.

### Remarks

Irrelevant in case of a point location operation. When the EWedgeGauge::Measure method is called, and if not enough valid points are detected, then the method returns directly, and the measured gauge is set to the nominal parameters.

# EWedgeGauge::SetRadii

Sets the nominal radius and breadth of the EWedgeGauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetRadii(
 float outerRadius,
 float breadth
 )
```

### Parameters

*outerRadius*

-

*breadth*

Breadth of the EWedgeGauge, which must be negative or zero. Its default value is **-50**.

### Remarks

A EWedgeGauge is fully defined knowing its nominal position (given by the coordinates of its center), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude and its outline tolerance.
By default, the EWedgeGauge radius value is **50**, which means 50 pixels when the field of view is not calibrated and 50 "units" in case of a calibrated field of view.

# EWedgeGauge::GetSmoothing

# EWedgeGauge::SetSmoothing

Number of pixels used for the low-pass filtering operation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetSmoothing()
void SetSmoothing(OEV_UINT32 un32Smoothing)
```

### Remarks

To reduce the effect of noise, the profile data can be low-pass filtered along the point location gauge direction.

# EWedgeGauge::GetThickness

# EWedgeGauge::SetThickness

Number of parallel segments used to extract the data profile.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetThickness()
void SetThickness(OEV_UINT32 un32Thickness)
```

### Remarks

To reduce the effect of noise and/or strengthen a transition, several parallel profiles can be accumulated.

# EWedgeGauge::GetThreshold

# EWedgeGauge::SetThreshold

Threshold level used to delimit significant peaks in the data profile.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetThreshold()
void SetThreshold(OEV_UINT32 un32Threshold)
```

### Remarks

When analyzing a derivative profile, a peak is made up of consecutive pixel values above **Threshold**. To detect weak [strong] transitions, lower [raise] the **Threshold** value.
To avoid interference of noise, an additional parameter is provided. The **MinAmplitude** parameter is an offset added to **Threshold** when a peak is to be detected.
When the pixel values of the derivative profile do not reach **Threshold + MinAmplitude**, the peak is not taken into account. Anyway, when a peak is taken into account, all the pixels with values above **Threshold** are considered (for more accuracy). Setting the **MinAmplitude** value to **0** merely cancels its effect.

# EWedgeGauge::GetTolerance

# EWedgeGauge::SetTolerance

Searching area half thickness of the wedge fitting gauge.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetTolerance()
void SetTolerance(float f32Tolerance)
```

## Remarks

A wedge fitting gauge is fully defined knowing its nominal position (its center coordinates), its outer nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.
By default, the searching area thickness of the wedge fitting gauge is **20** (2x10), which means 20 pixels when the field of view is not calibrated, and 20 physical units in case of a calibrated field of view.

# EWedgeGauge::GetTransitionChoice

# EWedgeGauge::SetTransitionChoice

Transition choice.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::ETransitionChoice GetTransitionChoice()
```

```
void SetTransitionChoice(Euresys::Open_eVision_2_10::ETransitionChoice transitionChoice)
```

### Remarks

Several peaks may be detected along a point location gauge. This parameter helps to select the desired transition.
In case of ETransitionChoice_NthFromBegin or ETransitionChoice_NthFromEnd transition choice, set EWedgeGauge::TransitionIndex to specify the desired transition.
By default, the selected transition corresponds to the one with the largest amplitude (ETransitionChoice_LargestAmplitude).

# EWedgeGauge::GetTransitionIndex

# EWedgeGauge::SetTransitionIndex

Index (from **0** on) of the transition to be retained when the transition choice parameter is set to NthFromBegin or NthFromEnd.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetTransitionIndex()
void SetTransitionIndex(OEV_UINT32 un32TransitionIndex)
```

### Remarks

Several peaks may be detected along a point location gauge. This parameter helps to select the desired transition.
By default, the first transition is retained (the index value is **0**).

# EWedgeGauge::GetTransitionType

# EWedgeGauge::SetTransitionType

Transition type.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::ETransitionType GetTransitionType()
void SetTransitionType(Euresys::Open_eVision_2_10::ETransitionType transitionType)
```

## Remarks

The type of a transition tells whether it crosses increasing or decreasing gray-level values. This helps discriminate between nearby edges of an object.
By default, the searched transition type is indifferently a black to white or a white to black transition (ETransitionType_BwOrWb).

# EWedgeGauge::GetType

Shape type.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::EShapeType GetType()
```

# EWedgeGauge::GetValid

Flag indicating if at least one valid transition has been found.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetValid()
```

### Remarks

A **FALSE** value means that no measurement has been performed. A **TRUE** value means that a transition was found along the sample path inspected with the last call to EWedgeGauge::MeasureSample, and thus a point has been measured.

# EWedgeGauge::SetWedge

Sets the nominal position (center coordinates), diameter, breadth, angular origin and amplitude of the wedge fitting gauge according to a known wedge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetWedge(const EWedge& wedge)
```

# 4.192. EWedgeShape Class

Manages a wedge shape.

**Base Class:** EShape
**Derived Class(es):** EWedgeGauge
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| Closest | Find the daughter shape that is the closest to this shape. To retrieve the closest shape, use EShape::ClosestShape. |
| CopyTo | Copies all the data of the current EWedgeShape object into another EWedgeShape object and returns it. |
| Drag | Moves a handle to a new position and updates the position parameters of the shape. |
| Draw | Draws a graphical representation of a shape, as defined by EDrawingMode. |
| DrawWithCurrentPen | Draws a graphical representation of a shape, as defined by EDrawingMode. |
| EWedgeShape | Constructs a EWedgeShape object. |
| GetAmplitude | Angular amplitude of the EWedgeShape. |
| GetAngle | Orientation of the shape. |
| GetApexAngle | Angular position at the apex of the EWedgeShape. |

| | |
|---|---|
| GetBreadth | Breadth of the EWedgeShape. |
| GetCenter | Center point of the shape. |
| GetCenterX | Abscissa of the origin point of the shape. |
| GetCenterY | Ordinate of the origin point of the shape. |
| GetCorners | Retrieves the coordinates of each corner of the EWedgeShape. |
| GetDirect | Flag indicating whether positive angles correspond to a clockwise or an anticlockwise rotation. |
| GetEdges | Retrieves each edge of the EWedgeShape. |
| GetEndAngle | Ending angular position of the EWedgeShape. |
| GetFullBreadth | Flag indicating if the EWedgeShape has a full breadth (**breadth = radius**). |
| GetFullCircle | Flag indicating if the EWedgeShape is a full circle (**amplitude = 2 PI**). |
| GetInnerApex | Inner apex point coordinates of the EWedgeShape. |

| | |
|---|---|
| GetInnerArcLength | Inner circle arc length of the EWedgeShape. |
| GetInnerDiameter | Inner diameter of the EWedgeShape. |
| GetInnerEnd | Inner end point coordinates of the EWedgeShape. |
| GetInnerOrg | Inner origin point coordinates of the EWedgeShape. |
| GetInnerPoint | Returns the coordinates of a particular point specified by its location along the inner circle arc. |
| GetInnerRadius | Inner radius of the EWedgeShape |
| GetMidEdges | Retrieves the center coordinates of each edge of the wedge fitting gauge. |
| GetOrgAngle | Angular position from where the EWedgeShape extents. |
| GetOuterApex | Outer apex point coordinates of the EWedgeShape. |
| GetOuterArcLength | Outer circle arc length of the EWedgeShape. |
| GetOuterDiameter | Outer diameter of the EWedgeShape. |

| | |
|---|---|
| GetOuterEnd | Outer end point coordinates of the EWedgeShape. |
| GetOuterOrg | Outer origin point coordinates of the EWedgeShape. |
| GetOuterPoint | Returns the coordinates of a particular point specified by its location along the outter circle arc. |
| GetOuterRadius | Outer radius of the EWedgeShape. |
| GetPoint | Returns the coordinates of a particular point specified by its location along the wedge perimeter. |
| GetScale | Horizontal sensor resolution, in pixels per unit. |
| GetType | Shape type. |
| HitTest | Checks if there is a handle under the cursor. |
| operator= | Copies all the data from another EWedgeShape object into the current EWedgeShape object |
| SetAmplitude | Angular amplitude of the EWedgeShape. |

| | |
|---|---|
| SetAngle | Orientation of the shape. |
| SetCenter | Center point of the shape. |
| SetCenterXY | Sets the center coordinates of a EWedgeShape object. |
| SetDiameters | Sets the nominal inner/outer diameter and breadth of the EWedgeShape. |
| SetFromCenterAndOrigin | Sets the geometric parameters (nominal position and diameter, breadth, angular position and amplitude) of an EWedgeShape object. |
| SetFromOriginMiddleEnd | Sets the geometric parameters (nominal position and diameter, breadth, angular position and amplitude) of an EWedgeShape object. |
| SetFromTwoPoints | DEPRECATED (you should use EWedgeShape::SetFromCenterAndOrigin): Sets the geometric parameters (nominal position and diameter, breadth, angular position and amplitude) of an EWedgeShape object. |
| SetRadii | Sets the nominal radius and breadth of the EWedgeShape. |
| SetScale | Horizontal sensor resolution, in pixels per unit. |

| | |
|---|---|
| SetWedge | Sets the nominal position, length and rotation angle of the wedge according to a known EWedge object. |

# EWedgeShape::GetAmplitude

# EWedgeShape::SetAmplitude

Angular amplitude of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetAmplitude() const
void SetAmplitude(float amplitude)
```

## Remarks

A EWedgeShape is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.
The sign of the rotation angle depends whether the field of view is calibrated or not.
* When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value.
* When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# EWedgeShape::GetAngle

# EWedgeShape::SetAngle

Orientation of the shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetAngle() const
void SetAngle(float f32Angle)
```

# EWedgeShape::GetApexAngle

Angular position at the apex of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetApexAngle() const
```

## Remarks

A EWedgeShape is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.
The sign of the rotation angle depends whether the field of view is calibrated or not.
* When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value.
* When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# EWedgeShape::GetBreadth

Breadth of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetBreadth() const
```

### Remarks

A EWedgeShape is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.

# EWedgeShape::GetCenter

# EWedgeShape::SetCenter

Center point of the shape.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetCenter() const
void SetCenter(const EPoint& center)
```

# EWedgeShape::GetCenterX

Abscissa of the origin point of the shape.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetCenterX() const
```

# EWedgeShape::GetCenterY

Ordinate of the origin point of the shape.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetCenterY() const
```

# EWedgeShape::Closest

Find the daughter shape that is the closest to this shape. To retrieve the closest shape, use EShape::ClosestShape.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Closest(
 )
```

# EWedgeShape::CopyTo

Copies all the data of the current EWedgeShape object into another EWedgeShape object and returns it.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EWedgeShape* CopyTo(
 EWedgeShape* dest,
 BOOL bRecursive
 )
```

### Parameters

*dest*

Pointer to the EWedgeShape object in which the current EWedgeShape object data have to be copied.

*bRecursive*

**TRUE** if the children shapes have to be copied as well, **FALSE** otherwise.

### Remarks

In case of a **NULL** pointer, a new EWedgeShape object will be created and returned.

# EWedgeShape::GetDirect

Flag indicating whether positive angles correspond to a clockwise or an anticlockwise rotation.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL GetDirect()
```

### Remarks

**TRUE** (default) means that angles increase anticlockwisely in a direct coordinate system, and clockwisely in an inverse coordinate system.
* When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards.
* When the field of view is not calibrated, the coordinate system is said to be inverse the abscissa extends rightwards and the ordinate extends downwards.

# EWedgeShape::Drag

Moves a handle to a new position and updates the position parameters of the shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 OEV_INT32 cursorX,
 OEV_INT32 cursorY
 )
```

### Parameters

*cursorX*

-

*cursorY*

-

# EWedgeShape::Draw

Draws a graphical representation of a shape, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

```
void Draw(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

*color*

-

# EWedgeShape::DrawWithCurrentPen

Draws a graphical representation of a shape, as defined by EDrawingMode.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingMode,
 BOOL daughters
 )
```

### Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingMode*

Indicates how the point location or model fitting gauge must be displayed, as defined by EDrawingMode.

*daughters*

**TRUE** if the daughters gauges are to be displayed also.

# EWedgeShape::GetEndAngle

Ending angular position of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetEndAngle() const
```

### Remarks

A EWedgeShape is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.
The sign of the rotation angle depends whether the field of view is calibrated or not.
* When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value.
* When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# EWedgeShape::EWedgeShape

Constructs a EWedgeShape object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EWedgeShape(
 )

void EWedgeShape(
 const EWedgeShape& other
 )
```

### Parameters

*other*

Another EWedgeShape object to be copied in the new EWedgeShape object.

# EWedgeShape::GetFullBreadth

Flag indicating if the EWedgeShape has a full breadth (**breadth = radius**).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetFullBreadth() const
```

# EWedgeShape::GetFullCircle

Flag indicating if the EWedgeShape is a full circle (**amplitude = 2 PI**).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL GetFullCircle() const
```

# EWedgeShape::GetCorners

Retrieves the coordinates of each corner of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetCorners(
 EPoint& ar,
 EPoint& AAr,
 EPoint& aRR,
 EPoint& AARR
 )
```

### Parameters

*ar*

Coordinates of the inner org corner of the EWedgeShape.

*AAr*

Coordinates of the inner end corner of the EWedgeShape.

*aRR*

Coordinates of the outer org corner of the EWedgeShape.

*AARR*

Coordinates of the outer end corner of the EWedgeShape.

# EWedgeShape::GetEdges

Retrieves each edge of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetEdges(
 ELine& a,
 ELine& AA,
 ECircle& r,
 ECircle& RR
 )
```

### Parameters

*a*

Org edge of the EWedgeShape.

*AA*

End edge of the EWedgeShape.

*r*

Inner edge of the EWedgeShape.

*RR*

Outer edge of the EWedgeShape.

# EWedgeShape::GetInnerPoint

Returns the coordinates of a particular point specified by its location along the inner circle arc.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetInnerPoint(
 float fraction
 )
```

### Parameters

*fraction*

Point location expressed as a fraction of the circle arc (range **[-1, +1]**).

# EWedgeShape::GetMidEdges

Retrieves the center coordinates of each edge of the wedge fitting gauge.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void GetMidEdges(
 EPoint& a,
 EPoint& AA,
 EPoint& r,
 EPoint& RR
 )
```

### Parameters

*a*

Center coordinates of the org edge of the EWedgeShape.

*AA*

Center coordinates of the end edge of the EWedgeShape.

*r*

Center coordinates of the inner edge of the EWedgeShape.

*RR*

Center coordinates of the outer edge of the EWedgeShape.

# EWedgeShape::GetOuterPoint

Returns the coordinates of a particular point specified by its location along the outter circle arc.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetOuterPoint(
 float fraction
 )
```

### Parameters

*fraction*

Point location expressed as a fraction of the circle arc (range **[-1, +1]**).

# EWedgeShape::GetPoint

Returns the coordinates of a particular point specified by its location along the wedge perimeter.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetPoint(
 float breadthFraction,
 float angleFraction
 )
```

### Parameters

*breadthFraction*

Point location expressed as a fraction of the wedge breadth (range -1, +1).

*angleFraction*

Point location expressed as a fraction of the wedge amplitude (range -1, +1).

# EWedgeShape::HitTest

Checks if there is a handle under the cursor.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL HitTest(
 BOOL daughters
 )
```

### Parameters

*daughters*

-

# EWedgeShape::GetInnerApex

Inner apex point coordinates of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetInnerApex() const
```

# EWedgeShape::GetInnerArcLength

Inner circle arc length of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetInnerArcLength() const
```

# EWedgeShape::GetInnerDiameter

Inner diameter of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetInnerDiameter() const
```

# EWedgeShape::GetInnerEnd

Inner end point coordinates of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetInnerEnd() const
```

# EWedgeShape::GetInnerOrg

Inner origin point coordinates of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint GetInnerOrg() const
```

# EWedgeShape::GetInnerRadius

Inner radius of the EWedgeShape

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetInnerRadius() const
```

# EWedgeShape::operator=

Copies all the data from another EWedgeShape object into the current EWedgeShape object

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EWedgeShape& operator=(
 const EWedgeShape& other
 )
```

## Parameters

*other*

EWedgeShape object to be copied

# EWedgeShape::GetOrgAngle

Angular position from where the EWedgeShape extents.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetOrgAngle() const
```

## Remarks

A EWedgeShape is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.
The sign of the rotation angle depends whether the field of view is calibrated or not.
* When the field of view is calibrated, the coordinate system is said to be direct, the abscissa extends rightwards and the ordinate extends upwards. In this case, an anticlockwise rotation leads to a positive angle value.
* When the field of view is not calibrated, the coordinate system is said to be inverse, the abscissa extends rightwards and the ordinate extends downwards. In this case, a clockwise rotation leads to a positive angle value.

# EWedgeShape::GetOuterApex

Outer apex point coordinates of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetOuterApex() const
```

## Remarks

A EWedgeShape is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.

# EWedgeShape::GetOuterArcLength

Outer circle arc length of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetOuterArcLength() const
```

## Remarks

A EWedgeShape is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.

# EWedgeShape::GetOuterDiameter

Outer diameter of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetOuterDiameter() const
```

## Remarks

A EWedgeShape is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.

# EWedgeShape::GetOuterEnd

Outer end point coordinates of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetOuterEnd() const
```

### Remarks

A EWedgeShape is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.

# EWedgeShape::GetOuterOrg

Outer origin point coordinates of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetOuterOrg() const
```

### Remarks

A EWedgeShape is fully defined knowing its nominal position (its center coordinates), its nominal outer radius (diameter), its breadth (must be negative), the angular position from where it extents, its angular amplitude, and its outline tolerance.

# EWedgeShape::GetOuterRadius

Outer radius of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetOuterRadius() const
```

### Remarks

A EWedgeShape is fully defined knowing its nominal position (its center coordinates), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude, and its outline tolerance.

# EWedgeShape::GetScale

# EWedgeShape::SetScale

Horizontal sensor resolution, in pixels per unit.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetScale() const
void SetScale(float f32Scale)
```

# EWedgeShape::SetCenterXY

Sets the center coordinates of a EWedgeShape object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetCenterXY(
 float centerX,
 float centerY
 )
```

### Parameters

*centerX*

Center coordinates of the EWedgeShape object.

*centerY*

Center coordinates of the EWedgeShape object.

# EWedgeShape::SetDiameters

Sets the nominal inner/outer diameter and breadth of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetDiameters(
 float diameter,
 float breadth
 )
```

### Parameters

*diameter*

Outer diameter of the EWedgeShape. The default value is **100**.

*breadth*

Breadth of the EWedgeShape. It must be negative or zero. Its default value is **-50**.

### Remarks

A EWedgeShape is fully defined knowing its nominal position (given by the coordinates of its center), its nominal outer radius (diameter), its breadth, the angular position from where it extents, its angular amplitude and its outline tolerance.
By default, the EWedgeShape diameter value is **100**, which means 100 pixels when the field of view is not calibrated and 100 "units" in case of a calibrated field of view.

# EWedgeShape::SetFromCenterAndOrigin

Sets the geometric parameters (nominal position and diameter, breadth, angular position and amplitude) of an EWedgeShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromCenterAndOrigin(
 const EPoint& center,
 const EPoint& origin,
 float breadth,
 BOOL direct
 )
```

### Parameters

*center*

Center coordinates of the wedge at its nominal position. The default value is **(0,0)**.

*origin*

Origin point coordinates of the wedge.

*breadth*

Nominal breadth of the wedge. It must be negative or zero. The default value is **-50**.

*direct*

**TRUE** (default) means that angles increase anticlockwise in a direct coordinate system and clockwise in an inverse coordinate system.

### Remarks

In a direct coordinate system, the abscissa extends rightwards and the ordinate extends upwards. The coordinate system is said to be inverse if the abscissa extends rightwards and the ordinate extends downwards.

# EWedgeShape::SetFromOriginMiddleEnd

Sets the geometric parameters (nominal position and diameter, breadth, angular position and amplitude) of an EWedgeShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromOriginMiddleEnd(
 const EPoint& origin,
 const EPoint& middle,
 const EPoint& end,
 float breadth,
 BOOL fullCircle
 )
```

## Parameters

*origin*

Origin point coordinates of the wedge.

*middle*

Middle point coordinates of the wedge.

*end*

End point coordinates of the wedge.

*breadth*

Nominal breadth of the wedge. It must be negative or zero. The default value is **-50**.

*fullCircle*

**TRUE** (default) in case of a full turn wedge. If **fullCircle** is **FALSE**, **origin** and **end** give the wedge's amplitude.

## Remarks

In a direct coordinate system, the abscissa extends rightwards and the ordinate extends upwards. The coordinate system is said to be inverse if the abscissa extends rightwards and the ordinate extends downwards.

# EWedgeShape::SetFromTwoPoints

DEPRECATED (you should use EWedgeShape::SetFromCenterAndOrigin): Sets the geometric parameters (nominal position and diameter, breadth, angular position and amplitude) of an EWedgeShape object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFromTwoPoints(
 const EPoint& center,
 const EPoint& origin,
 float breadth,
 BOOL direct
 )
```

### Parameters

*center*

Center coordinates of the wedge at its nominal position. The default value is **(0,0)**.

*origin*

Origin point coordinates of the wedge.

*breadth*

Nominal breadth of the wedge. It must be negative or zero. The default value is **-50**.

*direct*

**TRUE** (default) means that angles increase anticlockwise in a direct coordinate system and clockwise in an inverse coordinate system.

### Remarks

In a direct coordinate system, the abscissa extends rightwards and the ordinate extends upwards. The coordinate system is said to be inverse if the abscissa extends rightwards and the ordinate extends downwards.

# EWedgeShape::SetRadii

Sets the nominal radius and breadth of the EWedgeShape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetRadii(
 float outerRadius,
 float breadth
 )
```

### Parameters

*outerRadius*

Outer radius of the EWedgeShape. The default value is **50**.

*breadth*

Breadth of the EWedgeShape, which must be negative or zero. Its default value is **-50**.

### Remarks

A EWedgeShape is fully defined knowing its nominal position (given by the coordinates of its center), its nominal radius (diameter), its breadth, the angular position from where it extents, its angular amplitude and its outline tolerance.
By default, the EWedgeShape radius value is **50**, which means 50 pixels when the field of view is not calibrated and 50 "units" in case of a calibrated field of view.

# EWedgeShape::GetType

Shape type.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EShapeType GetType()
```

# EWedgeShape::SetWedge

Sets the nominal position, length and rotation angle of the wedge according to a known EWedge object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetWedge(const EWedge& wedge)
```

# 4.193. EWorldShape Class

Manages a complete context for calibrating a field of view.

**Base Class:** EShape
**Namespace:** Euresys::Open_eVision_2_10

## Methods

| | |
|---|---|
| AddLandmark | Adds a new pair of points coordinates (in Sensor and World spaces) to the set of landmarks used for calibration. |
| AddPoint | Adds a new point coordinates (in Sensor space) to the set of grid points used for calibration. |
| AutoCalibrate | Returns the best calibration modes for the current calibration grid and calibrates the field of view accordingly. |
| AutoCalibrateDotGrid | Performs an automatic calibration based on a dot grid image. |

| | |
|---|---|
| AutoCalibrateLandmarks | Returns the best calibration modes for the current landmark set and calibrates the field of view accordingly. |
| Calibrate | Performs a calibration according to the specified combination of calibration modes. |
| CalibrationSucceeded | Getter method for the **CalibrationSucceeded** property. This property is the flag indicating if the calibration has succeeded (**TRUE**), that is whether the mean variation of grid points (distance between computed grid points and ideal grid points in world space) and the maximum variation of grid points are between given tolerances. |
| Closest | Find the daughter shape that is the closest to this shape. To retrieve the closest shape, use EShape::ClosestShape. |
| DisableTypeFilter | Enables all shape types |
| Drag | Moves a handle to a new position and updates the position parameters of the shape. |
| DragLandmark | Moves the landmark to a new position. |
| Draw | Draws the world coordinate axis. |
| DrawCrossGrid | Draws a regular grid of crosses in world coordinates. |

| | |
|---|---|
| DrawCrossGridWithCurrentPen | Draws a regular grid of crosses in world coordinates. |
| DrawGrid | Draws the reconstructed grid to be used for grid calibration. |
| DrawGridWithCurrentPen | Draws the reconstructed grid to be used for grid calibration. |
| DrawLandmarks | Draws the landmarks to be used for landmark calibration. |
| DrawWithCurrentPen | Draws the world coordinate axis. |
| EmptyLandmarks | Resets the landmark specification sequence. |
| EnableTypeFilter | Enables the filter of the specified shape type |
| EWorldShape | Constructs a EWorldShape object. |
| GetAngle | Tilt X angle, that is the amplitude of the rotation applied around the X-axis of the sensor to bring the optical (Z) axis perpendicular to the field of view. |
| GetCalibrationModes | Current calibration mode, made from a combination of values. |
| GetCenter | Position of the origin point of the reference frame as projected onto the image, i.e. the Sensor position of the point at World coordinates **(0,0)**. |

| | |
|---|---|
| GetCenterX | Horizontal position of the origin point of the reference frame as projected onto the image, i.e. the Sensor abscissa of the point at World coordinates **(0,0)**. |
| GetCenterY | Vertical position of the origin point of the reference frame as projected onto the image, i.e. the Sensor ordinate of the point at World coordinates **(0,0)**. |
| GetDistortionStrength | Optical distortion strength, i.e. the ratio of the image diagonal length with and without optical distortion introduced by the lens. |
| GetDistortionStrength2 | Optical distortion strength of the second order. |
| GetFieldHeight | Field-of-view height, in physical units. |
| GetFieldWidth | Field-of-view width, in physical units. |
| GetGridPointsMaxVariation | Maximum variation of grid points (maximum distance between computed grid points and ideal grid points in world space), normalized using the pitch values, after a call to EWorldShape::CalibrationSucceeded |
| GetGridPointsMaxVariationThreshold | Grid points maximum variation threshold, that is the value above which the maximum variation of grid points (maximum distance between computed grid points and ideal grid points in world space) is considered too high, and thus marks the calibration as a failure when using EWorldShape::CalibrationSucceeded. |

| | |
|---|---|
| GetGridPointsMeanVariation | Mean variation of grid points (mean distance between computed grid points and ideal grid points in world space), normalized using the pitch values, after a call to EWorldShape::CalibrationSucceeded. |
| GetGridPointsMeanVariationThreshold | Grid points mean variation threshold, that is the value above which the mean variation of grid points (mean distance between computed grid points and ideal grid points in world space) is considered too high, and thus marks the calibration as a failure when using EWorldShape::CalibrationSucceeded. |
| GetHitLandmark | Returns the landmark selected by EWorldShape::HitLandmarks or ~0 if no landmark is selected. |
| GetLandmarkElement | Returns the landmark element corresponding to the given index. |
| GetNumLandmarkElements | Returns the number of landmark elements. |
| GetPanX | Current horizontal panning factor for drawing operations. |
| GetPanY | Current vertical panning factor for drawing operations. |
| GetPerspectiveStrength | Perspective effect coefficient, that is the inverse of the observation distance. |
| GetRatio | **XResolution/YResolution** ratio. |

| | |
|---|---|
| GetScale | The scale of the frame. |
| GetSensorHeight | Logical image height, that is the number of pixels vertically. |
| GetSensorWidth | Logical image width, that is the number of pixels horizontally. |
| GetTiltXAngle | Tilt X angle, that is the amplitude of the rotation applied around the X-axis of the sensor to bring the optical (Z) axis perpendicular to the field of view. |
| GetTiltYAngle | Tilt Y angle, that is the amplitude of the rotation applied around the Y-axis of the sensor to bring the optical (Z) axis perpendicular to the field of view. |
| GetType | Shape type. |
| GetXResolution | Horizontal sensor resolution, in pixels per unit. |
| GetYResolution | Vertical sensor resolution, in pixels per unit. |
| GetZoomX | Current horizontal zooming factor for drawing operations. |
| GetZoomY | Current vertical zooming factor for drawing operations. |
| HitLandmarks | Checks if the cursor is placed over a landmark point. |

| | |
|---|---|
| HitTest | Checks if there is a handle under the cursor. |
| operator= | Copies all the data from another EWorldShape object into the current EWorldShape object |
| RebuildGrid | Reconstructs the grid of points from the given dot centers, to compute the World coordinates of the points. |
| RemoveLandmark | Removes a landmark. |
| SensorToWorld | Performs coordinate transform for arbitrary points from Sensor space to World space. |
| SetAngle | Tilt X angle, that is the amplitude of the rotation applied around the X-axis of the sensor to bring the optical (Z) axis perpendicular to the field of view. |
| SetCalibrationModes | Current calibration mode, made from a combination of values. |
| SetCenter | Position of the origin point of the reference frame as projected onto the image, i.e. the Sensor position of the point at World coordinates **(0,0)**. |
| SetCenterXY | Position of the origin point of the reference frame as projected onto the image, i.e. the Sensor position of the point at World coordinates **(0,0)**. |

| | |
|---|---|
| SetDistortion | Sets the optical distortion parameters |
| SetFieldSize | Sets the field of view size in physical units. |
| SetGridPointsMaxVariationThreshold | Grid points maximum variation threshold, that is the value above which the maximum variation of grid points (maximum distance between computed grid points and ideal grid points in world space) is considered too high, and thus marks the calibration as a failure when using EWorldShape::CalibrationSucceeded. |
| SetGridPointsMeanVariationThreshold | Grid points mean variation threshold, that is the value above which the mean variation of grid points (mean distance between computed grid points and ideal grid points in world space) is considered too high, and thus marks the calibration as a failure when using EWorldShape::CalibrationSucceeded. |
| SetPan | Sets the horizontal and vertical panning factors for drawing operations. |
| SetPerspective | Sets the perspective effect coefficient, i.e. the inverse of the observation distance. |
| SetRatio | **XResolution/YResolution** ratio. |
| SetResolution | Sets the sensor resolution in pixels per unit in both directions. |

| | |
|---|---|
| SetScale | The scale of the frame. |
| SetSensor | Initializes the calibration object using all given parameters. |
| SetSensorSize | Sets the logical image size, i.e. the number of pixels horizontally and vertically. |
| SetSize | Sets the frame size. |
| SetupUnwarp | Prepares a lookup table for fast image unwarping. |
| SetZoom | Sets the horizontal and vertical zooming factors for drawing operations. |
| Unwarp | Unwarps a distorted image using the current calibration model. |
| WorldToSensor | Performs coordinate transform for arbitrary points from World space to Sensor space. |

# WorldShape::AddLandmark

Adds a new pair of points coordinates (in Sensor and World spaces) to the set of landmarks used for calibration.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddLandmark(
 const EPoint& sensorPoint,
 const EPoint& worldPoint
 )
```

### Parameters

*sensorPoint*

Sensor point coordinates.

*worldPoint*

Corresponding World point coordinates.

### Remarks

Landmark calibration is the process of computing the calibration parameters by means of a set of known points for which the coordinates are available in both World and Sensor spaces. Usually, such points are chosen as salient features on the part or target in view. They must be such that appropriate image processing techniques allow measuring their positions from the image (directly or indirectly by geometric constructions), while at the same time their coordinates in a reference frame are known.

# EWorldShape::AddPoint

Adds a new point coordinates (in Sensor space) to the set of grid points used for calibration.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void AddPoint(
 const EPoint& sensorPoint
 )
```

### Parameters

*sensorPoint*

Sensor point coordinates.

## Remarks

Grid calibration is the process of computing the calibration parameters by means of a set of points known to lie on a rectangular grid. If the grid pitch is known and one of the points is chosen as the origin point, the points can be used as landmarks. By contrast with the landmark calibration functions, the World coordinates of the grid points need not be specified, nor do they have to be given in any specific order. The calibration algorithm is capable of sorting out the points to reconstruct the grid topology. Typically, this function is used in conjunction with blob analysis to extract the dot centers from a grid of dots. Anyway, any other scheme can be used. The grid of points need not be complete, i.e. some of the nodes may be missing, and the points need not completely fill a rectangular area. Landmark calibration is simply achieved by providing a series of point coordinates (in Sensor space only) and then calling the grid reconstruction function followed by the calibration function.

# EWorldShape::GetAngle

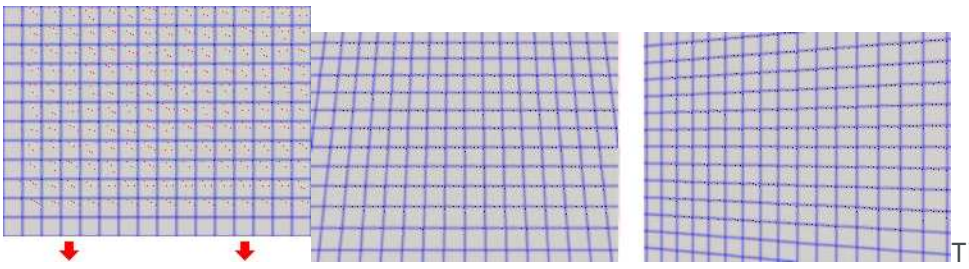
# EWorldShape::SetAngle

Tilt X angle, that is the amplitude of the rotation applied around the X-axis of the sensor to bring the optical (Z) axis perpendicular to the field of view.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetAngle() const
void SetAngle(float f32Angle)
```

## Remarks



Tilt X Angle and Tilt Y Angle

# EWorldShape::AutoCalibrate

Returns the best calibration modes for the current calibration grid and calibrates the field of view accordingly.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 AutoCalibrate(
 BOOL testEmpiricalModes
 )
```

## Parameters

*testEmpiricalModes*

Boolean indicating whether empirical calibration modes (ECalibrationMode_Quadratic and ECalibrationMode_Bilinear) should be considered when determining the best calibration modes.

# EWorldShape::AutoCalibrateDotGrid

Performs an automatic calibration based on a dot grid image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 AutoCalibrateDotGrid(
 EROIBW8* sourceImage,
 float columnPitch,
 float rowPitch,
 BOOL testEmpiricalModes
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*columnPitch*

Actual pitches of the grid, i.e. distances between vertical and horizontal rows of the grid.

*rowPitch*

Actual pitches of the grid, i.e. distances between vertical and horizontal rows of the grid.

*testEmpiricalModes*

Boolean indicating whether empirical calibration modes (ECalibrationMode_Quadratic and ECalibrationMode_Bilinear) should be considered when determining the best calibration modes. Default value is **FALSE**.

### Remarks

Returns the best calibration mode for the current dot grid. The EWorldShape::AutoCalibrateDotGrid method will first do an automatic blob analysis in order to extract all dots (all blobs whose area is smaller than 5 pixels will be considered as noise and rejected). The dot gravity centers are used as the grid reference points. Then, the EWorldShape::AutoCalibrateDotGrid method will select and compute the best calibration mode by reducing the fitting error.

# EWorldShape::AutoCalibrateLandmarks

Returns the best calibration modes for the current landmark set and calibrates the field of view accordingly.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 AutoCalibrateLandmarks(
 BOOL testEmpiricalModes
 )
```

### Parameters

*testEmpiricalModes*

Boolean indicating whether empirical calibration modes (ECalibrationMode_Quadratic and ECalibrationMode_Bilinear) should be considered when determining the best calibration modes. Default value is **FALSE**.

### Remarks

Landmark calibration is the process of computing the calibration parameters by means of a set of known points for which the coordinates are available in both World and Sensor spaces. Usually, such points are chosen as salient features on the part or target in view. They must be such that appropriate image processing techniques allow measuring their positions from the image (directly or indirectly by geometric constructions), while at the same time their coordinates in a reference frame are known. The EWorldShape::AutoCalibrateLandmarks method is meant to be used with landmark calibration only. To calibrate automatically your field of view using a dot grid, use the EWorldShape::AutoCalibrate method instead.

# EWorldShape::Calibrate

Performs a calibration according to the specified combination of calibration modes.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Calibrate(
 OEV_UINT32 calibrationModes
 )
```

### Parameters

*calibrationModes*

Calibration modes, as defined by a combination of values from ECalibrationMode.

### Remarks

Landmark calibration is the process of computing the calibration parameters by means of a set of known points for which the coordinates are available in both World and Sensor spaces. Usually, such points are chosen as salient features on the part or target in view. They must be such that appropriate image processing techniques allow measuring their positions from the image (directly or indirectly by geometric constructions), while at the same time their coordinates in a reference frame are known. In some cases, not all requested calibration modes are honored. After calibration, EWorldShape::CalibrationModes returns the actual combination of modes in effect.

# EWorldShape::GetCalibrationModes

# EWorldShape::SetCalibrationModes

Current calibration mode, made from a combination of values.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetCalibrationModes()
void SetCalibrationModes(OEV_UINT32 calibrationModes)
```

### Remarks

The supported calibration modes can be set to ECalibrationMode_Raw, meaning that no calibration at all is performed (the World coordinates are pixel indices), or to the logical sum of other values from ECalibrationMode.

# EWorldShape::CalibrationSucceeded

Getter method for the **CalibrationSucceeded** property. This property is the flag indicating if the calibration has succeeded (**TRUE**), that is whether the mean variation of grid points (distance between computed grid points and ideal grid points in world space) and the maximum variation of grid points are between given tolerances.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL CalibrationSucceeded(
 )
```

### Remarks

The mean and maximum grid point variations are normalized using the pitch values. By default, tolerances are set to **0.05** (5 %) for the mean, and **0.1** (10 %) for the maximum grid point variation. You can get and set these tolerances using the EWorldShape::GridPointsMeanVariationThreshold and EWorldShape::GridPointsMaxVariationThreshold properties.

# EWorldShape::GetCenter

# EWorldShape::SetCenter

Position of the origin point of the reference frame as projected onto the image, i.e. the Sensor position of the point at World coordinates **(0,0)**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint GetCenter() const
void SetCenter(const EPoint& center)
```

# EWorldShape::GetCenterX

Horizontal position of the origin point of the reference frame as projected onto the image, i.e. the Sensor abscissa of the point at World coordinates **(0,0)**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetCenterX() const
```

# EWorldShape::GetCenterY

Vertical position of the origin point of the reference frame as projected onto the image, i.e. the Sensor ordinate of the point at World coordinates **(0,0)**.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetCenterY() const
```

# EWorldShape::Closest

Find the daughter shape that is the closest to this shape. To retrieve the closest shape, use EShape::ClosestShape.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Closest(
 )
```

# EWorldShape::DisableTypeFilter

Enables all shape types

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DisableTypeFilter(
 )
```

# EWorldShape::GetDistortionStrength

Optical distortion strength, i.e. the ratio of the image diagonal length with and without optical distortion introduced by the lens.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetDistortionStrength()
```

# EWorldShape::GetDistortionStrength2

Optical distortion strength of the second order.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetDistortionStrength2()
```

# EWorldShape::Drag

Moves a handle to a new position and updates the position parameters of the shape.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Drag(
 OEV_INT32 n32CursorX,
 OEV_INT32 n32CursorY
 )
```

### Parameters

*n32CursorX*

Current cursor coordinates.

*n32CursorY*

Current cursor coordinates.

# EWorldShape::DragLandmark

Moves the landmark to a new position.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DragLandmark(
 OEV_INT32 n32CursorX,
 OEV_INT32 n32CursorY
 )
```

### Parameters

*n32CursorX*

Current cursor coordinates.

*n32CursorY*

Current cursor coordinates.

# EWorldShape::Draw

Draws the world coordinate axis.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void Draw(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingModes,
 BOOL daughters
 )

void Draw(
 HDC graphicContext,
 const ERGBColor& color,
 Euresys::Open_eVision_2_10::EDrawingMode drawingModes,
 BOOL daughters
 )

void Draw(
 EDrawAdapter* graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingModes,
 BOOL daughters
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingModes*

Indicates how the world coordinate axis must be displayed, as defined by EDrawingMode.

*daughters*

Indicates whether the daughter shapes are to be displayed as well.

*color*

The color in which to draw the overlay.

# EWorldShape::DrawCrossGrid

Draws a regular grid of crosses in world coordinates.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void DrawCrossGrid(
 HDC graphicContext,
 float minimumX,
 float maximumX,
 float minimumY,
 float maximumY,
 OEV_UINT32 numberOfIntervalsX,
 OEV_UINT32 numberOfIntervalsY
 )

void DrawCrossGrid(
 HDC graphicContext,
 const ERGBColor& color,
 float minimumX,
 float maximumX,
 float minimumY,
 float maximumY,
 OEV_UINT32 numberOfIntervalsX,
 OEV_UINT32 numberOfIntervalsY
 )

void DrawCrossGrid(
 EDrawAdapter* graphicContext,
 float minimumX,
 float maximumX,
 float minimumY,
 float maximumY,
 OEV_UINT32 numberOfIntervalsX,
 OEV_UINT32 numberOfIntervalsY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*minimumX*

Abscissa of the leftmost crosses, in world coordinates.

*maximumX*

Abscissa of the rightmost crosses, in world coordinates.

*minimumY*

Ordinate of the leftmost crosses, in world coordinates.

*maximumY*

Ordinate of the rightmost crosses, in world coordinates.

*numberOfIntervalsX*

Number of intervals between crosses along the horizontal direction.

*numberOfIntervalsY*

Number of intervals between crosses along the vertical direction.

*color*

The color in which to draw the overlay.

# EWorldShape::DrawCrossGridWithCurrentPen

Draws a regular grid of crosses in world coordinates.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawCrossGridWithCurrentPen(
 HDC graphicContext,
 float minimumX,
 float maximumX,
 float minimumY,
 float maximumY,
 OEV_UINT32 numberOfIntervalsX,
 OEV_UINT32 numberOfIntervalsY
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*minimumX*

Abscissa of the leftmost crosses, in world coordinates.

*maximumX*

Abscissa of the rightmost crosses, in world coordinates.

*minimumY*

Ordinate of the leftmost crosses, in world coordinates.

*maximumY*

Ordinate of the rightmost crosses, in world coordinates.

*numberOfIntervalsX*

Number of intervals between crosses along the horizontal direction.

*numberOfIntervalsY*

Number of intervals between crosses along the vertical direction.

# EWorldShape::DrawGrid

Draws the reconstructed grid to be used for grid calibration.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawGrid(
 HDC graphicContext
 )
void DrawGrid(
 HDC graphicContext,
 const ERGBColor& color
 )
void DrawGrid(
 EDrawAdapter* graphicContext
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*color*

The color in which to draw the overlay.

# EWorldShape::DrawGridWithCurrentPen

Draws the reconstructed grid to be used for grid calibration.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawGridWithCurrentPen(
 HDC graphicContext
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

# EWorldShape::DrawLandmarks

Draws the landmarks to be used for landmark calibration.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawLandmarks(
 HDC graphicContext
 )

void DrawLandmarks(
 EDrawAdapter* graphicContext
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

# EWorldShape::DrawWithCurrentPen

Draws the world coordinate axis.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void DrawWithCurrentPen(
 HDC graphicContext,
 Euresys::Open_eVision_2_10::EDrawingMode drawingModes,
 BOOL daughters
 )
```

## Parameters

*graphicContext*

Handle of the device context on which to draw.

*drawingModes*

Indicates how the world coordinate axis must be displayed, as defined by EDrawingMode.

*daughters*

Indicates whether the daughter shapes are to be displayed as well.

# EWorldShape::EmptyLandmarks

Resets the landmark specification sequence.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void EmptyLandmarks(
 )
```

### Remarks

Landmark calibration is the process of computing the calibration parameters by means of a set of known points for which the coordinates are available in both World and Sensor spaces. Usually, such points are chosen as salient features on the part or target in view. They must be such that appropriate image processing techniques allow measuring their positions from the image (directly or indirectly by geometric constructions), while at the same time their coordinates in a reference frame are known.

# EWorldShape::EnableTypeFilter

Enables the filter of the specified shape type

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EnableTypeFilter(
 OEV_UINT32 un32Types
 )
```

### Parameters

*un32Types*

The type of the shape to filter from EShapeType.

# EWorldShape::EWorldShape

Constructs a EWorldShape object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EWorldShape(
 const EWorldShape& other
 )

void EWorldShape(
 )
```

### Parameters

*other*

Another EWorldShape object to be copied in the new EWorldShape object.

# EWorldShape::GetFieldHeight

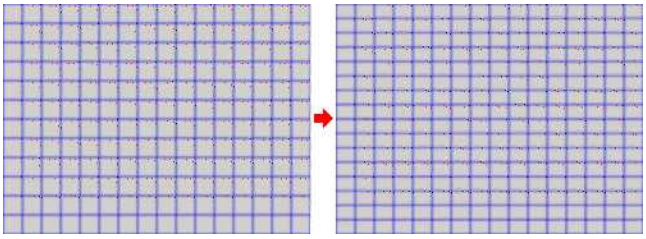
Field-of-view height, in physical units.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetFieldHeight()
```

### Remarks



Field size not matching the sensor size results in non-square pixels. Pixels having non-square aspect ratioBeware there is a restriction pertaining to the allowed image anisotropy. The resulting pixels aspect ratio (**XResolution/YResolution**) should be in the range **[-4/3, -3/4]** (or **[3/4, 4/3]**), otherwise the calibration process could fail.

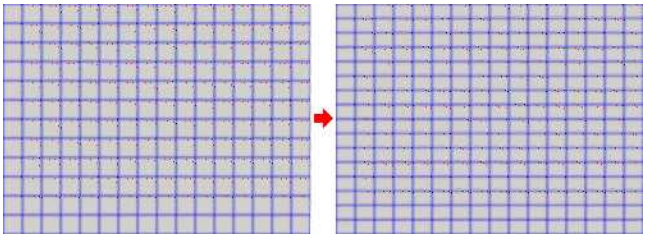
# EWorldShape::GetFieldWidth

Field-of-view width, in physical units.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetFieldWidth()
```

### Remarks



Field size not matching the sensor size results in non-square pixels. Pixels having non-square aspect ratioBeware there is a restriction pertaining to the allowed image anisotropy. The resulting pixels aspect ratio (**XResolution/YResolution**) should be in the range **[-4/3, -3/4]** (or **[3/4, 4/3]**), otherwise the calibration process could fail.

# EWorldShape::GetLandmarkElement

Returns the landmark element corresponding to the given index.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
ELandmark& GetLandmarkElement(
 OEV_UINT32 i
 )
const ELandmark& GetLandmarkElement(
 OEV_UINT32 i
 )
```

### Parameters

*i*

Landmark index.

# EWorldShape::GetGridPointsMaxVariation

Maximum variation of grid points (maximum distance between computed grid points and ideal grid points in world space), normalized using the pitch values, after a call to EWorldShape::CalibrationSucceeded

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetGridPointsMaxVariation()
```

### Remarks

The maximum grid point variation is normalized using the pitch values.

# EWorldShape::GetGridPointsMaxVariationThreshold

# EWorldShape::SetGridPointsMaxVariationThreshold

Grid points maximum variation threshold, that is the value above which the maximum variation of grid points (maximum distance between computed grid points and ideal grid points in world space) is considered too high, and thus marks the calibration as a failure when using EWorldShape::CalibrationSucceeded.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetGridPointsMaxVariationThreshold()
void SetGridPointsMaxVariationThreshold(float maxVariationThreshold)
```

## Remarks

The maximum grid point variation is normalized using the pitch values.

# EWorldShape::GetGridPointsMeanVariation

Mean variation of grid points (mean distance between computed grid points and ideal grid points in world space), normalized using the pitch values, after a call to EWorldShape::CalibrationSucceeded.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetGridPointsMeanVariation()
```

## Remarks

The mean grid point variation is normalized using the pitch values.

# EWorldShape::GetGridPointsMeanVariationThreshold

# EWorldShape::SetGridPointsMeanVariationThreshold

Grid points mean variation threshold, that is the value above which the mean variation of grid points (mean distance between computed grid points and ideal grid points in world space) is considered too high, and thus marks the calibration as a failure when using EWorldShape::CalibrationSucceeded.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetGridPointsMeanVariationThreshold()
void SetGridPointsMeanVariationThreshold(float meanVariationThreshold)
```

### Remarks

The mean grid point variation is normalized using the pitch values.

# EWorldShape::GetHitLandmark

Returns the landmark selected by EWorldShape::HitLandmarks or ~0 if no landmark is selected.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 GetHitLandmark()
```

# EWorldShape::HitLandmarks

Checks if the cursor is placed over a landmark point.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void HitLandmarks(
 )
```

## Remarks

Landmark calibration is the process of computing the calibration parameters by means of a set of known points for which the coordinates are available in both World and Sensor spaces. Usually, such points are chosen as salient features on the part or target in view. They must be such that appropriate image processing techniques allow measuring their positions from the image (directly or indirectly by geometric constructions), while at the same time their coordinates in a reference frame are known.

# EWorldShape::HitTest

Checks if there is a handle under the cursor.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL HitTest(
 BOOL bDaughters
 )
```

## Parameters

*bDaughters*

Indicates if the check must be done in the whole hierarchy or just this object.

# EWorldShape::GetNumLandmarkElements

Returns the number of landmark elements.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 GetNumLandmarkElements() const
```

# EWorldShape::operator=

Copies all the data from another EWorldShape object into the current EWorldShape object

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EWorldShape& operator=(
 const EWorldShape& other
 )
```

## Parameters

*other*

EWorldShape object to be copied

# EWorldShape::GetPanX

Current horizontal panning factor for drawing operations.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetPanX()
```

# EWorldShape::GetPanY

Current vertical panning factor for drawing operations.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetPanY()
```

# EWorldShape::GetPerspectiveStrength

Perspective effect coefficient, that is the inverse of the observation distance.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetPerspectiveStrength()
```

## Remarks

The larger this parameter, the more perceivable the perspective distortion will be. A **NULL** value corresponds to a telecentric lens.

# EWorldShape::GetRatio

# EWorldShape::SetRatio

**XResolution/YResolution** ratio.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetRatio()
void SetRatio(float ratio)
```

### Remarks

If **Ratio** equals **-1** (or **1**), pixels are square. Beware there is a restriction pertaining to the allowed image anisotropy. The resulting pixels aspect ratio (**XResolution/YResolution**) should be in the range **[-4/3, -3/4]** (or **[3/4, 4/3]**), otherwise the calibration process could fail.

# EWorldShape::RebuildGrid

Reconstructs the grid of points from the given dot centers, to compute the World coordinates of the points.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 RebuildGrid(
 float colPitch,
 float rowPitch,
 OEV_UINT32 centerIndex,
 BOOL direct
 )
```

```
OEV_UINT32 RebuildGrid(
 float colPitch,
 float rowPitch,
 const EPoint& worldCenter,
 OEV_UINT32 centerIndex,
 BOOL direct
 )
```

### Parameters

*colPitch*

Actual pitches of the grid, that are distances between vertical and horizontal rows of the grid.

*rowPitch*

Actual pitches of the grid, that are distances between vertical and horizontal rows of the grid.

*centerIndex*

Index of the grid point chosen as coordinate origin point. By default, the most central grid point.

*direct*

**TRUE** if the world reference frame points upwards.

*worldCenter*

World coordinates of the starting grid point.

### Remarks

This member function also returns the number of grid points that were connected. This prepares the calibration using landmarks (for use by member EWorldShape::Calibrate). Landmark calibration is the process of computing the calibration parameters by means of a set of known points for which the coordinates are available in both World and Sensor spaces. Usually, such points are chosen as salient features on the part or target in view. They must be such that appropriate image processing techniques allow measuring their positions from the image (directly or indirectly by geometric constructions), while at the same time their coordinates in a reference frame are known. See also Dot-Grid-Based Calibration for the grid construction algorithm.

# EWorldShape::RemoveLandmark

Removes a landmark.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void RemoveLandmark(
 OEV_UINT32 index
 )
```

### Parameters

*index*

Index of the landmark to be removed.

# EWorldShape::GetScale

# EWorldShape::SetScale

The scale of the frame.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetScale() const
void SetScale(float f32Scale)
```

# EWorldShape::GetSensorHeight

Logical image height, that is the number of pixels vertically.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 GetSensorHeight()
```

# EWorldShape::SensorToWorld

Performs coordinate transform for arbitrary points from Sensor space to World space.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
EPoint SensorToWorld(
 const EPoint& sensorPoint
 )
```

### Parameters

*sensorPoint*

Sensor point.

# EWorldShape::GetSensorWidth

Logical image width, that is the number of pixels horizontally.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 GetSensorWidth()
```

# EWorldShape::SetCenterXY

Position of the origin point of the reference frame as projected onto the image, i.e. the Sensor position of the point at World coordinates **(0,0)**.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetCenterXY(
 float centerX,
 float centerY
 )
```

### Parameters

*centerX*

Horizontal position (abscissa)

*centerY*

Vertical position (ordinate)

# EWorldShape::SetDistortion

Sets the optical distortion parameters

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetDistortion(
 float distortionStrength,
 float distortionStrength2
 )
```

### Parameters

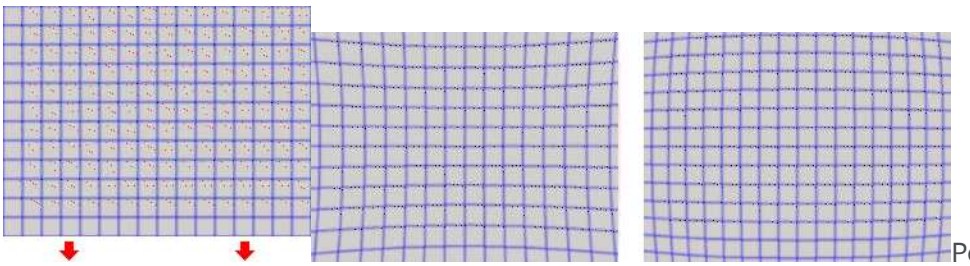
*distortionStrength*

Optical distortion strength, i.e. the ratio of the image diagonal length with and without optical distortion introduced by the lens.

*distortionStrength2*

Optical distortion strength of the second order.

### Remarks


Positive distortion and negative distortion

# EWorldShape::SetFieldSize

Sets the field of view size in physical units.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetFieldSize(
 float width,
 float height
 )
```

### Parameters

*width*

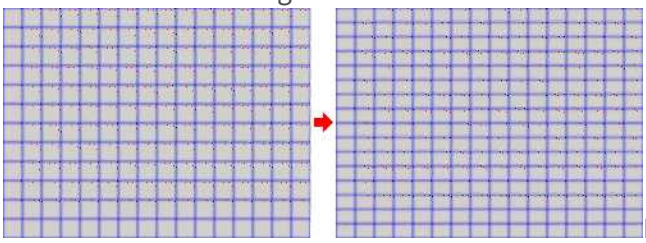
Full image physical width, in length units.

*height*

Full image physical height, in length units. If not specified, same as physical width.

### Remarks

Field size not matching the sensor size results in non-square pixels. By default, the pixels are square.

 Pixels having non-square aspect ratioBeware there is a restriction pertaining to the allowed image anisotropy. The resulting pixels aspect ratio (**XResolution/YResolution**) should be in the range **[-4/3, -3/4]** (or **[3/4, 4/3]**), otherwise the calibration process could fail.

# EWorldShape::SetPan

Sets the horizontal and vertical panning factors for drawing operations.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetPan(
 float panX,
 float panY
 )
```

### Parameters

*panX*

Horizontal panning factor. By default, no panning occurs.

*panY*

Vertical panning factor. By default, no panning occurs.

### Remarks

All objects attached to an EWorldShape object inherit of the same panning factor.

# EWorldShape::SetPerspective

Sets the perspective effect coefficient, i.e. the inverse of the observation distance.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetPerspective(
 float tiltXAngle,
 float tiltYAngle,
 float perspectiveStrength
 )
```

## Parameters

*tiltXAngle*

Tilt angles, i.e. the amplitudes of the rotations applied around the X and Y axis of the sensor to bring the optical (Z) axis perpendicular to the field of view.

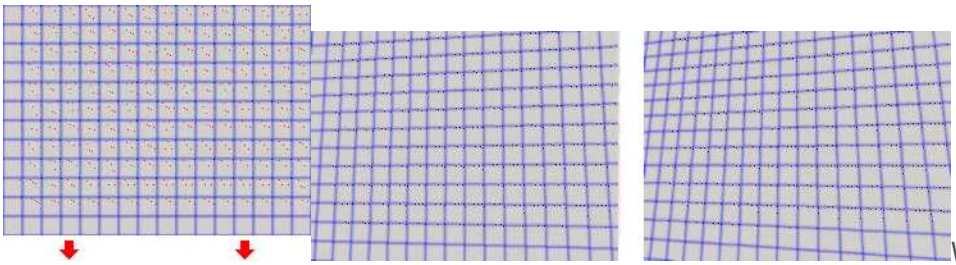
*tiltYAngle*

Tilt angles, i.e. the amplitudes of the rotations applied around the X and Y axis of the sensor to bring the optical (Z) axis perpendicular to the field of view.

*perspectiveStrength*

Perspective effect coefficient.

## Remarks

The larger this parameter, the more perceivable the perspective distortion will be. A **NULL** value corresponds to a telecentric lens.  Weak Perspective and Strong Perspective

# EWorldShape::SetResolution

Sets the sensor resolution in pixels per unit in both directions.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetResolution(
 float resolutionX,
 float resolutionY
 )
```

### Parameters

*resolutionX*

Horizontal resolution in pixels per units

*resolutionY*

Vertical resolution in pixels per units. If not specified, same as horizontal resolution.

### Remarks

By default, the pixels are square.

# EWorldShape::SetSensor

Initializes the calibration object using all given parameters.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
void SetSensor(
 OEV_INT32 sensorWidth,
 OEV_INT32 sensorHeight,
 float fieldWidth,
 float fieldHeight,
 float centerX,
 float centerY,
 float angle,
 float tiltXAngle,
 float tiltYAngle,
 float perspectiveStrength,
 float distortionStrength,
 float distortionStrength2,
 float opticalCenterX,
 float opticalCenterY,
 OEV_UINT32 calibrationModes
 )
```

### Parameters

*sensorWidth*

Logical size of the field of view, i.e. image size, in pixels.

*sensorHeight*

Logical size of the field of view, i.e. image size, in pixels.

*fieldWidth*

Physical size of the field of view. By default (argument omitted), the pixels are square.

*fieldHeight*

Physical size of the field of view. By default (argument omitted), the pixels are square.

*centerX*

Position of the "intersection" between the optical axis and the field of view in the image. By default, if the calibration modes contain ECalibrationMode_Raw, it is set to 0. Otherwise, it is set to the image center.

*centerY*

Position of the "intersection" between the optical axis and the field of view in the image. By default, if the calibration modes contain ECalibrationMode_Raw, it is set to 0 (or to the bottommost pixel index if the calibration modes also contain ECalibrationMode_Inverse). Otherwise, it is set to the image center.

*angle*

Skew angle, i.e. angle formed by the axis of reference and the image edges. By default (argument omitted), no skewing effect is assumed.

*tiltXAngle*

Rotation angles on the X axis to bring the optical axis perpendicular to the image plane. By default (argument omitted), no perspective effect is assumed.

*tiltYAngle*

Rotation angles on the Y axis to bring the optical axis perpendicular to the image plane. By default (argument omitted), no perspective effect is assumed.

*perspectiveStrength*

Relative importance of the perspective effect. By default, no perspective effect is assumed, as if the lens was telecentric.

*distortionStrength*

Relative importance of the lens radial distortion. Positive for barrel, negative for cushion. By default (argument omitted), no optical distortion is assumed.

*distortionStrength2*

Relative importance of the lens radial distortion of the second order. By default (argument omitted), no optical distortion of second order is assumed.

*opticalCenterX*

X Position of the "intersection" between the optical axis and the field of view in the image. By default (argument omitted) the image center.

*opticalCenterY*

Y Position of the "intersection" between the optical axis and the field of view in the image. By default (argument omitted) the image center.

*calibrationModes*

Desired calibration mode effects to be combined, as defined by ECalibrationMode. By default (argument omitted), the simplest model compatible with the given parameters is chosen.

### Remarks

The function automatically selects the appropriate calibration model by checking the parameters. The use of a more complex calibration mode can be enforced by means of parameter EWorldShape::CalibrationModes, not a simpler one.

# EWorldShape::SetSensorSize

Sets the logical image size, i.e. the number of pixels horizontally and vertically.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void SetSensorSize(
 OEV_INT32 width,
 OEV_INT32 height
 )
```

### Parameters

*width*

Full image logical sizes, in pixels.

*height*

Full image logical sizes, in pixels.

# EWorldShape::SetSize

Sets the frame size.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetSize(
 float sizeX,
 float sizeY
 )
```

### Parameters

*sizeX*

Frame X-axis length. The default value is **100**.

*sizeY*

Frame Y-axis length. By default, both axes have the same length.

### Remarks

By default, both frame axis value are set to **100**, which means 100 pixels when the field of view is not calibrated and 100 "units" in case of a calibrated field of view.

# EWorldShape::SetupUnwarp

Prepares a lookup table for fast image unwarping.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetupUnwarp(
 EUnwarpingLut* lookupTable,
 EROIBW8* sourceImage,
 BOOL interpolate
 )
void SetupUnwarp(
 EUnwarpingLut* lookupTable,
 EROIC24* sourceImage,
 BOOL interpolate
 )
```

### Parameters

*lookupTable*

Pointer to the lookup table.

*sourceImage*

Pointer to the source image/ROI.

*interpolate*

Interpolation mode. Default value is **FALSE**.

### Remarks

The function should be called each time the system is re-calibrated (after the optical setup has been changed, for instance). A sample source image has to be supplied to EWorldShape::SetupUnwarp, and its row pitch is recorded in order to speedup the unwarping process. This implies that the following calls to EWorldShape::Unwarp are not allowed to use images with row pitches different from the source image initially supplied to EWorldShape::SetupUnwarp.

# EWorldShape::SetZoom

Sets the horizontal and vertical zooming factors for drawing operations.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void SetZoom(
 float zoomX,
 float zoomY
 )
```

### Parameters

*zoomX*

Horizontal zooming factor. By default, true scale is used.

*zoomY*

Vertical zooming factor. If set to **0**, the default value, the horizontal zooming factor is used instead, so as to provide isotropic zooming.

### Remarks

All objects attached to an EWorldShape inherit of the same zooming factor.

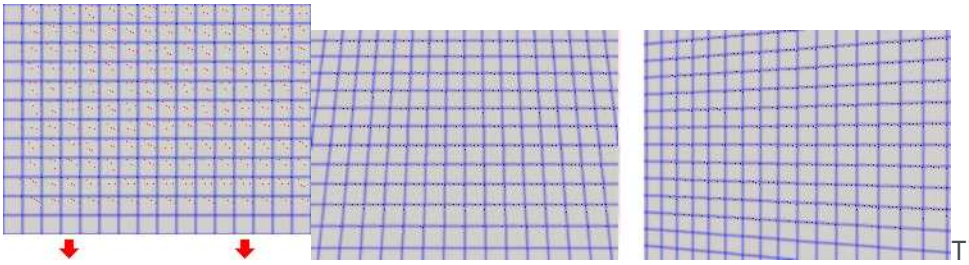
# EWorldShape::GetTiltXAngle

Tilt X angle, that is the amplitude of the rotation applied around the X-axis of the sensor to bring the optical (Z) axis perpendicular to the field of view.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetTiltXAngle()
```

### Remarks


Tilt X Angle and Tilt Y Angle

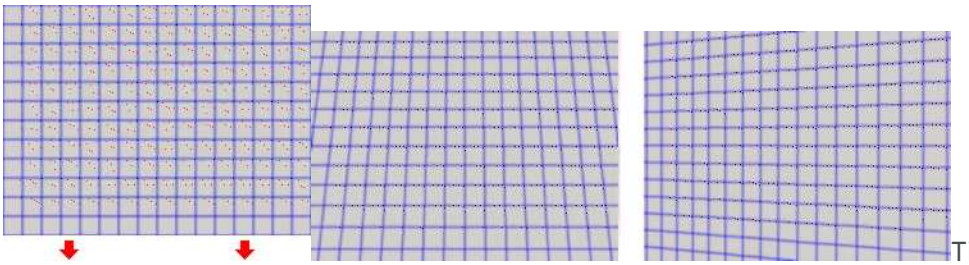
# EWorldShape::GetTiltYAngle

Tilt Y angle, that is the amplitude of the rotation applied around the Y-axis of the sensor to bring the optical (Z) axis perpendicular to the field of view.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetTiltYAngle()
```

### Remarks


Tilt X Angle and Tilt Y Angle

# EWorldShape::GetType

Shape type.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::EShapeType GetType()
```

# EWorldShape::Unwarp

Unwarps a distorted image using the current calibration model.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void Unwarp(
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 BOOL interpolate
 )

void Unwarp(
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 BOOL interpolate
 )

void Unwarp(
 EUnwarpingLut* lookupTable,
 EROIBW8* sourceImage,
 EROIBW8* destinationImage,
 BOOL interpolate
 )

void Unwarp(
 EUnwarpingLut* lookupTable,
 EROIC24* sourceImage,
 EROIC24* destinationImage,
 BOOL interpolate
 )
```

## Parameters

*sourceImage*

Pointer to the source image/ROI.

*destinationImage*

Pointer to the destination unwarped image.

*interpolate*

Interpolation mode. Default value is **FALSE**.

*lookupTable*

Pointer to the lookup table.

### Remarks

Using a pre-computed lookup table allows speeding up the unwarping process. The lookup table is initialized by means of the EWorldShape::SetupUnwarp function.

# EWorldShape::WorldToSensor

Performs coordinate transform for arbitrary points from World space to Sensor space.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EPoint WorldToSensor(
 const EPoint& worldPoint
 )
```

### Parameters

*worldPoint*

World point.

# EWorldShape::GetXResolution

Horizontal sensor resolution, in pixels per unit.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetXResolution()
```

# EWorldShape::GetYResolution

Vertical sensor resolution, in pixels per unit.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetYResolution()
```

# EWorldShape::GetZoomX

Current horizontal zooming factor for drawing operations.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GetZoomX()
```

# EWorldShape::GetZoomY

Current vertical zooming factor for drawing operations.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float GetZoomY()
```

# 4.194. EZMap Class

Represents a generic ZMap type interface.

**Derived Class(es):** EZMap8 EZMap16 EZMap32f

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| AddMetadata | Adds a metadata key (name) and value.<br>If the metadata key already exists, its value will be overwritten. |
| Clear | Clears the ZMap: replaces all pixels with the undefined value. |
| Create | Factory method to allocates and reads a ZMap from a file. Depending of the serialized EZMap type, it returns the corresponding EZMap8, EZMap16 or EZMap32f object. The allocated EZMap must be released after use. |
| Draw | Draws a EZMap in a device context. |
| DrawImage | Displays the internal image buffer |

| | |
|---|---|
| GetBufferPtr | Retrieves the pointer to the internal pixel buffer. |
| GetCheckedBufferPtr | Retrieves the pointer to the pixel buffer. |
| GetHeight | Access ZMap Height. |
| GetMapToWorldMatrix | Returns an E3DTransformMatrix that transforms positions from the EZMap space to the world space. This transformation is composed of rotation and translation only, so it is a rigid transformation and preserves distances. |
| GetMetadata | Returns the string value of the given metadata.<br>Throws an exception if it does not exist. |
| GetResolution | Gets the resolution of the EZMap along the X, Y and Z axis.<br>On the Z axis, the resolution is the number of metric units per grey value.<br>On the X and Y axis, the resolution is the number of metric units per pixel. |
| GetRowPitch | Returns the buffer row pitch. |
| GetSizeInWorld | Returns the dimensions of the EZMap in real world space (e.g metric unit). |
| GetType | Pixel accessor type. |

| | |
|---|---|
| GetWidth | Access ZMap Width. |
| GetWorldPositionFromPixelPosition | Returns the 3D world position corresponding to a EZMap pixel position. The world position is in the original point cloud space. The pixel space origin is at the upper left corner of the image and ranges to (width-1, height-1). The transformation to the world position is calculated using the center of the pixel. The pixel value at position (u,v) must not be undefined, otherwise an exception will be thrown. |
| GetWorldShape | Returns the EWorldShape for conversion between 2D image and 2D world space coordinates. This EWorldShape can be used by EasyGauge to do measurements on a EZMap in real space coordinates (e.g mm). |
| GetWorldToMapMatrix | Returns an E3DTransformMatrix that transforms positions from the world space to the EZMap space. This transformation is composed of rotation and translation only, so it is a rigid transformation and it preserves distances. |
| GetXResolution | Resolution of the EZMap along the X axis<br>The resolution is the number of metric units per pixel. |
| GetYResolution | Resolution of the EZMap along the Y axis<br>The resolution is the number of metric units per pixel. |

| | |
|---|---|
| GetZMapPositionFromPixelPosition | Returns the corresponding EZMap 3D position of a ZMap pixel. The ZMap position is in the original point ZMap space. The pixel position origin is the upper left corner of the image and ranges to (width-1, height-1). The center of the pixel is used for the transformation to the ZMap position. Pixel at position (u,v) must be defined, otherwise an exception will be thrown. |
| GetZResolution | Resolution of the EZMap along the Z axis<br>The resolution is the number of grey values per pixel. |
| ImageToWorld | Transforms a floating point (sub)pixel image position to a 3D world position.<br>The image space origin is at the upper left corner of the image and the X/Y unit size is one pixel.<br>The pixel Z value is in grey scale values (its range depends on the ZMap type). |
| ImageToZMap | Converts a 2D image (sub)pixel coordinate to the EZMap space.<br>(u,v) is the pixel position (with its origin in the upper left corner of the image).<br>(x,y) is the corresponding ZMap position (which has the same scale as the world space).<br>All values are expressed in floating point numbers. |
| IsVoid | Tests if the EZMap object size is zero. |
| Load | Restores the EZMap stored in the given Open eVision file. |
| LoadImage | Restores the EZMap image stored in the given image file. |

| | |
|---|---|
| LoadImageAndMetadata | Loads image format and Metadata in JSON format. |
| LoadMetadata | Loads Metadata in JSON format. |
| Save | Saves the EZMap object to the given Open eVision file. |
| SaveImage | Saves the EZMap image to the given image file. |
| SaveImageAndMetadata | Saves image format and Metadata JSON format. |
| SaveMetadata | Saves Metadata in JSON format. |
| Serialize | Serializes the EZMap object with all its attributes. |
| SerializeImage | Serializes the image associated to EZMap. |
| SetBufferPtr | Sets the pointer to an externally allocated image buffer. |
| SetHeight | Access ZMap Height. |
| SetResolution | Sets the resolution of the EZMap along the X, Y and Z axis.<br>On the Z axis, the resolution is the number of metric units per grey value.<br>On the X and Y axis, the resolution is the number of metric units per pixel. |

| | |
|---|---|
| SetSize | Sets the width and height of the EZMap. |
| SetWidth | Access ZMap Width. |
| SetXResolution | Resolution of the EZMap along the X axis<br>The resolution is the number of metric units per pixel. |
| SetYResolution | Resolution of the EZMap along the Y axis<br>The resolution is the number of metric units per pixel. |
| SetZResolution | Resolution of the EZMap along the Z axis<br>The resolution is the number of grey values per pixel. |
| WorldToImage | Transforms a 3D world position to a floating point (sub)pixel image position.<br>The image space origin is at the upper left corner of the image and the X/Y unit size is one pixel.<br>The Z value is given in grey scale value.<br>Returns TRUE if the pixel position is inside the image limits and the Z value is positive.<br>The parameter pixelPt is filled even if the point is outside the image. |
| WorldToZMap | Transforms a 3D world position to a 3D EZMap position.<br>The ZMap space origin is at the lower left corner of the image.<br>The scales of the world space and ZMap space are the same. |

| | |
|---|---|
| ZMapToImage | Converts a 2D coordinate in the EZMap space to image (sub)pixel space.<br>(x,y) is the ZMap position (which has the same scale as the world space).<br>(u,v) is the corresponding pixel position (with its origin in the upper left corner of the image).<br>All values are expressed in floating point numbers.<br>Returns TRUE if the pixel position is inside the image limits. |
| ZMapToWorld | Transforms a 3D EZMap position to a 3D world space position.<br>The ZMap space origin is at the lower left corner of the image. |

E

Z

# Map::AddMetadata

Adds a metadata key (name) and value.
If the metadata key already exists, its value will be overwritten.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void AddMetadata(
 const std::string& Key,
 const std::string& value
 )
```

## Parameters

*Key*

The name of the metadata. Names are unique.

*value*

The value for the given metadata.

# EZMap::Clear

Clears the ZMap: replaces all pixels with the undefined value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Clear(
 )
```

# EZMap::Create

Factory method to allocates and reads a ZMap from a file. Depending of the serialized EZMap type, it returns the corresponding EZMap8, EZMap16 or EZMap32f object. The allocated EZMap must be released after use.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EZMap* Create(
 const std::string& path
 )
```

## Parameters

*path*

Full path to the file.

# EZMap::Draw

Draws a EZMap in a device context.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void Draw(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

```
void Draw(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the ZMap is displayed in 1:1 scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Pan offset (in pixels) in the horizontal direction. By default, no panning is applied.

*panY*

Pan offset (in pixels) in the vertical direction. By default, no panning is applied.

*colorUndefinedPixel*

An optional parameter to choose the drawing color of undefined pixels.

*c24Vector*

When supplied, this parameter allows using a LUT that maps from Depth to C24 when drawing (false colors).

*bw8Vector*

When supplied, this parameter allows using a LUT that maps from Depth to BW8 when drawing.

### Remarks

A ZMap can be drawn (its pixels rendered) using a device context.
The horizontal and vertical zooming factors can be different but must be in the **1/16..16** range.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EZMap::DrawImage

Displays the internal image buffer

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void DrawImage(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

```
void DrawImage(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

```
void DrawImage(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

### Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the image is displayed in 1:1 scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Pan offset (in pixels) in the horizontal direction. By default, no panning is applied.

*panY*

Pan offset (in pixels) in the vertical direction. By default, no panning is applied.

*colorUndefinedPixel*

An optional parameter to choose the drawing color of undefined pixels.

*c24Vector*

When supplied, this parameter allows using a LUT that maps from Depth to C24 when drawing (false colors).

*bw8Vector*

When supplied, this parameter allows using a LUT that maps from Depth to BW8 when drawing.

### Remarks

An image can be drawn (its pixels rendered) using a device context.
The horizontal and vertical zooming factors can be different but must be in the **1/16..16** range.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EZMap::GetBufferPtr

Retrieves the pointer to the internal pixel buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void* GetBufferPtr(
 )

void* GetBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )

const void* GetBufferPtr(
 )

const void* GetBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel which we want the address.

*y*

Row of the pixel which we want the address.

### Remarks

This function does not check the value of the parameters.
Use carefully.

# EZMap::GetCheckedBufferPtr

Retrieves the pointer to the pixel buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void* GetCheckedBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )

const void* GetCheckedBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel of which we want the address.

*y*

Row of the pixel of which we want the address.

# EZMap::GetMetadata

Returns the string value of the given metadata.
Throws an exception if it does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
std::string GetMetadata(
 const std::string& Key
 )
```

### Parameters

*Key*

The name of an exising metadata.

# EZMap::GetResolution

Gets the resolution of the EZMap along the X, Y and Z axis.
On the Z axis, the resolution is the number of metric units per grey value.
On the X and Y axis, the resolution is the number of metric units per pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPoint GetResolution(
 )
void GetResolution(
 float& sx,
 float& sy,
 float& sz
 )
```

### Parameters

*sx*

Contains the resolution along the X axis.

*sy*

Contains the resolution along the Y axis.

*sz*

Contains the resolution along the Z axis.

# EZMap::GetSizeInWorld

Returns the dimensions of the EZMap in real world space (e.g metric unit).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void GetSizeInWorld(
 float& worldWidth,
 float& worldHeight
 )
```

### Parameters

*worldWidth*

Contains the size of the ZMap along the X axis (column).

*worldHeight*

Contains the size of the ZMap along the Y axis (row).

# EZMap::GetWorldPositionFromPixelPosition

Returns the 3D world position corresponding to a EZMap pixel position. The world position is in the original point cloud space. The pixel space origin is at the upper left corner of the image and ranges to (width-1, height-1). The transformation to the world position is calculated using the center of the pixel. The pixel value at position (u,v) must not be undefined, otherwise an exception will be thrown.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPoint GetWorldPositionFromPixelPosition(
 OEV_INT32 u,
 OEV_INT32 v
 )
```

### Parameters

*u*

Column of the pixel (bounds: [0,width[).

*v*

Row of the pixel (bounds: [0,height[).

# EZMap::GetZMapPositionFromPixelPosition

Returns the corresponding EZMap 3D position of a ZMap pixel. The ZMap position is in the original point ZMap space. The pixel position origin is the upper left corner of the image and ranges to (width-1, height-1). The center of the pixel is used for the transformation to the ZMap position. Pixel at position (u,v) must be defined, otherwise an exception will be thrown.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DPoint GetZMapPositionFromPixelPosition(
 OEV_INT32 u,
 OEV_INT32 v
 )
```

### Parameters

*u*

Column of the pixel (bounds: [0,width[).

*v*

Row of the pixel (bounds: [0,height[).

# EZMap::GetHeight

# EZMap::SetHeight

Access ZMap Height.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
OEV_INT32 GetHeight() const
void SetHeight(OEV_INT32 height)
```

# EZMap::ImageToWorld

Transforms a floating point (sub)pixel image position to a 3D world position.
The image space origin is at the upper left corner of the image and the X/Y unit size is one pixel.
The pixel Z value is in grey scale values (its range depends on the ZMap type).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ImageToWorld(
 const E3DPoint& pixelPt,
 E3DPoint& worldPt
 )
```

## Parameters

*pixelPt*

Position in the image space.

*worldPt*

Position in the 3D world space.

# EZMap::ImageToZMap

Converts a 2D image (sub)pixel coordinate to the EZMap space.
(u,v) is the pixel position (with its origin in the upper left corner of the image).
(x,y) is the corresponding ZMap position (which has the same scale as the world space).
All values are expressed in floating point numbers.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ImageToZMap(
 float u,
 float v,
 float& x,
 float& y
 )
```

### Parameters

*u*

X Coordinate of the pixel as a floating point value.

*v*

Y Coordinate of the pixel as a floating point value.

*x*

Position along horizontal axis in the ZMap space.

*y*

Position along vertical axis in the ZMap space.

# EZMap::IsVoid

Tests if the EZMap object size is zero.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool IsVoid(
 )
```

### Remarks

Returns **TRUE** if the ZMap size is zero.

# EZMap::Load

Restores the EZMap stored in the given Open eVision file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

### Remarks

When loading, the ZMap is resized if needed. This function restores all the ZMap attributes.

# EZMap::LoadImage

Restores the EZMap image stored in the given image file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void LoadImage(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

### Remarks

When loading, the ZMap is resized if needed. This function does not restore the ZMap attributes, only the image associated with the EZMap is updated.

# EZMap::LoadImageAndMetadata

Loads image format and Metadata in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void LoadImageAndMetadata(
 const std::string& pathImage,
 const std::string& pathMetadata
 )
```

### Parameters

*pathImage*

Full path to the file.

*pathMetadata*

Full path to the file.

# EZMap::LoadMetadata

Loads Metadata in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
void LoadMetadata(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

# EZMap::GetMapToWorldMatrix

Returns an E3DTransformMatrix that transforms positions from the EZMap space to the world space. This transformation is composed of rotation and translation only, so it is a rigid transformation and preserves distances.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
const E3DTransformMatrix& GetMapToWorldMatrix() const
```

# EZMap::GetRowPitch

Returns the buffer row pitch.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
OEV_INT32 GetRowPitch() const
```

# EZMap::Save

Saves the EZMap object to the given Open eVision file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 const std::string& path
 )
```

### Parameters

*path*

The full path to the destination file.

### Remarks

This format save the EZMap in a Open eVision file. This function stores all the ZMap attributes.

# EZMap::SaveImage

Saves the EZMap image to the given image file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveImage(
 const std::string& path,
 Euresys::Open_eVision_2_10::EImageFileType type
 )
```

### Parameters

*path*

The full path to the destination file.

*type*

File format, as defined by EImageFileType. If not specified, the file format is determined from the file extension.

### Remarks

This format save the image associated to EZMap in a standard image file and thus does not store ZMap attributes.

# EZMap::SaveImageAndMetadata

Saves image format and Metadata JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveImageAndMetadata(
 const std::string& pathImage,
 const std::string& pathMetadata,
 Euresys::Open_eVision_2_10::EImageFileType type
 )
```

### Parameters

*pathImage*

The full path to the destination file.

*pathMetadata*

The full path to the destination file.

*type*

File format, as defined by EImageFileType. If not specified, the file format is determined from the file extension.

# EZMap::SaveMetadata

Saves Metadata in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveMetadata(
 const std::string& path
 )
```

### Parameters

*path*

The full path to the destination file.

# EZMap::Serialize

Serializes the EZMap object with all its attributes.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or writen to.

# EZMap::SerializeImage

Serializes the image associated to EZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SerializeImage(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or writen to.

# EZMap::SetBufferPtr

Sets the pointer to an externally allocated image buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetBufferPtr(
 OEV_INT32 width,
 OEV_INT32 height,
 void* imagePointer,
 OEV_INT32 bitsPerRow
 )
```

### Parameters

*width*

The width of the supplied buffer, in pixels.

*height*

The height of the supplied buffer, in pixels.

*imagePointer*

The pointer (aligned on 4 bytes) to the buffer, which must be large enough to hold the data.

*bitsPerRow*

The total number of bits contained in a row, padding included.
Using the value **0** (default) means that this size is computed from the buffer width and the pixel size plus a padding with the smallest possible value that leads to a multiple of 4 bytes (32 bits), which is the minimum padding accepted by EZMap::SetBufferPtr.

# EZMap::SetResolution

Sets the resolution of the EZMap along the X, Y and Z axis.
On the Z axis, the resolution is the number of metric units per grey value.
On the X and Y axis, the resolution is the number of metric units per pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetResolution(
 E3DPoint resolution
 )
void SetResolution(
 float rx,
 float ry,
 float rz
 )
```

## Parameters

*resolution*

Contains the resolution along the X,Y and Z axis.

*rx*

Contains the resolution along the X axis.

*ry*

Contains the resolution along the Y axis.

*rz*

Contains the resolution along the Z axis.

# EZMap::SetSize

Sets the width and height of the EZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetSize(
 OEV_INT32 width,
 OEV_INT32 height
 )
void SetSize(
 const EZMap& other
 )
```

## Parameters

*width*

The new requested width.

*height*

The new requested height.

*other*

The other ZMap whose dimensions have to be used for the current object.

## Remarks

Open eVision will allocate a new image buffer (deallocate the old image buffer) if the supplied width and height are different from the existing ones.
If an external buffer has been specified by means of **SetImagePtr**, it will be kept only if the size does not change.
Creating a new Open eVision image buffer and setting its size creates a 4-byte aligned buffer, by default. The *size of an ZMap* is specified as a number of columns (width) and rows (height).
The maximum image dimensions are 32767 by 32767. Furthermore, it must fit into the available memory, that depends upon the physical memory, the operating system and the memory already allocated by the process in other modules or libraries.

# EZMap::GetType

Pixel accessor type.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
Euresys::Open_eVision_2_10::EImageType GetType() const
```

# EZMap::GetWidth

# EZMap::SetWidth

Access ZMap Width.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetWidth() const
void SetWidth(OEV_INT32 width)
```

# EZMap::GetWorldShape

Returns the EWorldShape for conversion between 2D image and 2D world space coordinates. This EWorldShape can be used by EasyGauge to do measurements on a EZMap in real space coordinates (e.g mm).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const EWorldShape& GetWorldShape() const
```

# EZMap::WorldToImage

Transforms a 3D world position to a floating point (sub)pixel image position.
The image space origin is at the upper left corner of the image and the X/Y unit size is one pixel.
The Z value is given in grey scale value.
Returns TRUE if the pixel position is inside the image limits and the Z value is positive.
The parameter pixelPt is filled even if the point is outside the image.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool WorldToImage(
 const E3DPoint& worldPt,
 E3DPoint& pixelPt
 )
```

## Parameters

*worldPt*

Position in the 3D world space.

*pixelPt*

Position in the image space.

# EZMap::GetWorldToMapMatrix

Returns an E3DTransformMatrix that transforms positions from the world space to the EZMap space. This transformation is composed of rotation and translation only, so it is a rigid transformation and it preserves distances.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const E3DTransformMatrix& GetWorldToMapMatrix() const
```

# EZMap::WorldToZMap

Transforms a 3D world position to a 3D EZMap position.
The ZMap space origin is at the lower left corner of the image.
The scales of the world space and ZMap space are the same.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void WorldToZMap(
 const E3DPoint& worldPt,
 E3DPoint& zmapPt
 )
```

## Parameters

*worldPt*

Position in the 3D world space.

*zmapPt*

Position in the ZMap space.

## Remarks

Do not use this method with the same variable as input and output. It might lead to incorrect results.

# EZMap::GetXResolution

# EZMap::SetXResolution

Resolution of the EZMap along the X axis
The resolution is the number of metric units per pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetXResolution() const
void SetXResolution(float resolution)
```

# EZMap::GetYResolution

# EZMap::SetYResolution

Resolution of the EZMap along the Y axis
The resolution is the number of metric units per pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetYResolution() const
void SetYResolution(float resolution)
```

# EZMap::ZMapToImage

Converts a 2D coordinate in the EZMap space to image (sub)pixel space.
(x,y) is the ZMap position (which has the same scale as the world space).
(u,v) is the corresponding pixel position (with its origin in the upper left corner of the image).
All values are expressed in floating point numbers.
Returns TRUE if the pixel position is inside the image limits.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool ZMapToImage(
 float x,
 float y,
 float& u,
 float& v
 )
```

### Parameters

*x*

Position along horizontal axis in the ZMap space.

*y*

Position along vertical axis in the ZMap space.

*u*

Column of the pixel as a floating point value.

*v*

Row of the pixel as a floating point value.

# EZMap::ZMapToWorld

Transforms a 3D EZMap position to a 3D world space position.
The ZMap space origin is at the lower left corner of the image.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ZMapToWorld(
 const E3DPoint& zmapPt,
 E3DPoint& worldPt
 )
```

### Parameters

*zmapPt*

Position in the ZMap space.

*worldPt*

Position in the 3D world space.

### Remarks

Do not use this method with the same variable as input and output. It might lead to incorrect results.

# EZMap::GetZResolution

# EZMap::SetZResolution

Resolution of the EZMap along the Z axis
The resolution is the number of grey values per pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetZResolution() const
void SetZResolution(float resolution)
```

# 4.195. EZMap16 Class

A ZMap16 is a 16bits corrected 2.5D image.
ZMap Pixel values (16 bits integers) represent distances from a 3D reference plane.
Distances are positive, during the ZMap generation all points below the reference plane are discarded.
The EZMap class also stores the transformation from the pixel coordinates to the real world coordinate system.
There could be undefined pixels in the ZMap.

**Base Class:** EZMap

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| AddMetadata | Adds a metadata key (name) and value.<br>If the metadata key already exists, its value will be overwritten. |
| AsEImage | Returns the EZMap16 as an EImageBW16 (16 bits gray scale) to use with existing eVision 2D tools. |
| Clear | Clears the ZMap: replaces all pixels with the undefined value. |
| ClearMetadata | Deletes all metadata. |
| ConvertCoordinatesMapToPixel | Converts 3D Map coordinates to Buffer coordinates |
| ConvertCoordinatesPixelToMap | Converts Buffer coordinates to 3D Map coordinates |

| | |
|---|---|
| CopyMetadataTo | Copies all metadata. |
| DeleteMetadata | Deletes value of this existing metadata key . Throws an exception if it does not exist. |
| Draw | Draws a EZMap16 in a device context. |
| DrawImage | Displays the internal image buffer |
| EZMap16 | Creates a 16 bits EZMap. |
| FillUndefinedPixels | Fills undefined pixels, used to fill the "holes" in the ZMap. |
| GetBufferPtr | Clears the ZMap: replaces all pixels with the undefined value. |
| GetCheckedBufferPtr | Retrieves the pointer to the pixel buffer. |
| GetHeight | Access ZMap Height. |
| GetMapToWorldMatrix | Returns an E3DTransformMatrix that transforms positions from the EZMap16 space to the world space. This transformation is composed of rotation and translation only, so it is a rigid transformation and preserves distances. |
| GetMetadata | Returns the string value of the given metadata.<br>Throws an exception if it does not exist. |

| | |
|---|---|
| GetPixel | Gets the value of a pixel . |
| GetPixelPositionFromWorldPosition | Returns in the u, v and value parameters the EZMap16 values corresponding to a 3D world position.<br>The world position is projected on the ZMap reference plane to get a position in the ZMap.<br>If the projected position is outside the ZMap, the method returns FALSE. |
| GetResolution | Gets the resolution. For the X and Y axes, the resolution is expressed in metric units per pixel.<br>For the Z axis it is expressed in metric units per grey scale value. |
| GetRowPitch | Returns the buffer row pitch. |
| GetSizeInWorld | Returns the dimensions of the EZMap16 in real world space (e.g metric unit). |
| GetType | Pixel accessor type. |
| GetUndefinedValue | Returns the Undefined value. That value is used to set to mark pixels with no valid depth value. |
| GetWidth | Access ZMap Width. |

| | |
|---|---|
| GetWorldPositionFromPixelPosition | Returns the 3D world position corresponding to a EZMap16 pixel position. The world position is in the original point cloud space. The pixel space origin is at the upper left corner of the image and ranges to (width-1, height-1). The transformation to the world position is calculated using the center of the pixel. The pixel value at position (u,v) must not be undefined, otherwise an exception will be thrown. |
| GetWorldShape | Returns the EWorldShape for conversion between 2D image and 2D world space coordinates. This EWorldShape can be used by EasyGauge to do measurements on a EZMap16 in real space coordinates (e.g mm). |
| GetWorldToMapMatrix | Returns an E3DTransformMatrix that transforms positions from the world space to the EZMap16 space. This transformation is composed of rotation and translation only, so it is a rigid transformation and it preserves distances. |
| GetXResolution | Resolution of the EZMap16 along the X axis<br>The resolution is the number of metric units per pixel. |
| GetYResolution | Resolution of the EZMap16 along the Y axis<br>The resolution is the number of metric units per pixel. |
| GetZMapPositionFromPixelPosition | Returns the corresponding EZMap16 3D position of a ZMap pixel. The ZMap position is in the original point ZMap space. The pixel position origin is the upper left corner of the image and ranges to (width-1, height-1). The center of the pixel is used for the transformation to the ZMap position. Pixel at position (u,v) must be defined, otherwise an exception will be thrown. |

| | |
|---|---|
| GetZResolution | Resolution of the EZMap16 along the Z axis<br>The resolution is the number of grey values per pixel. |
| GetZValue | Gets Z value (in metric coordinate) at pixel coordinates. |
| ImageToWorld | Transforms a floating point (sub)pixel image position to a 3D world position.<br>The image space origin is at the upper left corner of the image and the X/Y unit size is one pixel.<br>The pixel Z value is in grey scale values (its range depends on the ZMap type). |
| ImageToZMap | Converts a 2D image (sub)pixel coordinate to the EZMap16 space.<br>(u,v) is the pixel position (with its origin in the upper left corner of the image).<br>(x,y) is the corresponding ZMap position (which has the same scale as the world space).<br>All values are expressed in floating point numbers. |
| IsVoid | Tests if the EZMap16 object size is zero. |
| Load | Restores the EZMap16 stored in the given Open eVision file. |
| LoadImage | Restores the EZMap16 image stored in the given image file. |
| LoadImageAndMetadata | Loads image format and Metadata in JSON format. |
| LoadMetadata | Loads Metadata in JSON format. |

| | |
|---|---|
| ModifyMetadata | Changes an existing metadata key and value. Throws an exception if it does not exist. |
| operator= | Assignment operator. |
| Save | Saves the EZMap16 object to the given Open eVision file. |
| SaveImage | Saves the EZMap16 image to the given image file. |
| SaveImageAndMetadata | Saves image format and Metadata JSON format. |
| SaveMetadata | Saves Metadata in JSON format. |
| Serialize | Serializes the EZMap16 object with all its attributes. |
| SerializeImage | Serializes the image associated to EZMap16. |
| SetBufferPtr | Sets the pointer to an externally allocated image buffer. |
| SetHeight | Access ZMap Height. |
| SetPixel | Sets the value of a pixel . |

| | |
|---|---|
| SetResolution | Sets the resolution. For the X and Y axes, the resolution is expressed in metric units per pixel.<br>For the Z axis it is expressed in metric units per grey scale value. |
| SetSize | Sets the width and height of the EZMap16. |
| SetWidth | Access ZMap Width. |
| SetXResolution | Resolution of the EZMap16 along the X axis<br>The resolution is the number of metric units per pixel. |
| SetYResolution | Resolution of the EZMap16 along the Y axis<br>The resolution is the number of metric units per pixel. |
| SetZResolution | Resolution of the EZMap16 along the Z axis<br>The resolution is the number of grey values per pixel. |
| SetZValue | Sets Z value (in metric coordinate) at pixel coordinates. |
| WorldToImage | Transforms a 3D world position to a floating point (sub)pixel image position.<br>The image space origin is at the upper left corner of the image and the X/Y unit size is one pixel.<br>The Z value is given in grey scale value.<br>Returns TRUE if the pixel position is inside the image limits and the Z value is positive.<br>The parameter pixelPt is filled even if the point is outside the image. |

| | |
|---|---|
| WorldToZMap | Transforms a 3D world position to a 3D EZMap16 position.<br>The ZMap space origin is at the lower left corner of the image.<br>The scales of the world space and ZMap space are the same. |
| ZMapToImage | Converts a 2D coordinate in the EZMap16 space to image (sub-)pixel space.<br>(x,y) is the ZMap position (which has the same scale as the world space).<br>(u,v) is the corresponding pixel position (with its origin in the upper left corner of the image).<br>All values are expressed in floating point numbers.<br>Returns TRUE if the pixel position is inside the image limits. |
| ZMapToWorld | Transforms a 3D EZMap16 position to a 3D world space position.<br>The ZMap space origin is at the lower left corner of the image. |

E

Z

# Map16::AddMetadata

Adds a metadata key (name) and value.
If the metadata key already exists, its value will be overwritten.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void AddMetadata(
 const std::string& Key,
 const std::string& value
 )
```

## Parameters

*Key*

The name of the metadata. Names are unique.

*value*

The value for the given metadata.

# EZMap16::AsEImage

Returns the EZMap16 as an EImageBW16 (16 bits gray scale) to use with existing eVision 2D tools.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
EImageBW16& AsEImage(
 )

const EImageBW16& AsEImage(
 )
```

# EZMap16::Clear

Clears the ZMap: replaces all pixels with the undefined value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Clear(
 )
```

# EZMap16::ClearMetadata

Deletes all metadata.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ClearMetadata(
 )
```

# EZMap16::ConvertCoordinatesMapToPixel

Converts 3D Map coordinates to Buffer coordinates

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool ConvertCoordinatesMapToPixel(
 float x3D,
 float y3D,
 int& xBuffer,
 int& yBuffer
 )
```

## Parameters

*x3D*

The Map X coordinate.

*y3D*

The Map Y coordinate.

*xBuffer*

The returned Pixel X coordinate.

*yBuffer*

The returned Pixel Y coordinate.

# EZMap16::ConvertCoordinatesPixelToMap

Converts Buffer coordinates to 3D Map coordinates

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ConvertCoordinatesPixelToMap(
 int xBuffer,
 int yBuffer,
 float& x3D,
 float& y3D
 )
```

## Parameters

*xBuffer*

The pixel X coordinate.

*yBuffer*

The pixel Y coordinate.

*x3D*

The returned Map X coordinate.

*y3D*

The returned Map Y coordinate.

# EZMap16::CopyMetadataTo

Copies all metadata.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void CopyMetadataTo(
 EZMap16& other
 )
```

### Parameters

*other*

An other EZMap16.

# EZMap16::DeleteMetadata

Deletes value of this existing metadata key . Throws an exception if it does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void DeleteMetadata(
 const std::string& Key
 )
```

### Parameters

*Key*

-

# EZMap16::Draw

Draws a EZMap16 in a device context.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void Draw(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

```
void Draw(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the ZMap is displayed in 1:1 scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Pan offset (in pixels) in the horizontal direction. By default, no panning is applied.

*panY*

Pan offset (in pixels) in the vertical direction. By default, no panning is applied.

*colorUndefinedPixel*

An optional parameter to choose the drawing color of undefined pixels.

*c24Vector*

When supplied, this parameter allows using a LUT that maps from Depth to C24 when drawing (false colors).

*bw8Vector*

When supplied, this parameter allows using a LUT that maps from Depth to BW8 when drawing.

### Remarks

A ZMap can be drawn (its pixels rendered) using a device context.
The horizontal and vertical zooming factors can be different but must be in the **1/16..16** range.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EZMap16::DrawImage

Displays the internal image buffer

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void DrawImage(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

```
void DrawImage(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the image is displayed in 1:1 scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Pan offset (in pixels) in the horizontal direction. By default, no panning is applied.

*panY*

Pan offset (in pixels) in the vertical direction. By default, no panning is applied.

*colorUndefinedPixel*

An optional parameter to choose the drawing color of undefined pixels.

*c24Vector*

When supplied, this parameter allows using a LUT that maps from Depth to C24 when drawing (false colors).

*bw8Vector*

When supplied, this parameter allows using a LUT that maps from Depth to BW8 when drawing.

### Remarks

An image can be drawn (its pixels rendered) using a device context.
The horizontal and vertical zooming factors can be different but must be in the **1/16..16** range.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EZMap16::EZMap16

Creates a 16 bits EZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EZMap16(
 )
void EZMap16(
 OEV_INT32 width,
 OEV_INT32 height
 )
void EZMap16(
 const EZMap16& other
 )
```

### Parameters

*width*

The width of the new Zmap.

*height*

The height of the new Zmap.

*other*

Another Zmap.

# EZMap16::FillUndefinedPixels

Fills undefined pixels, used to fill the "holes" in the ZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void FillUndefinedPixels(
 EZMap16& outMap,
 Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsDirection direction,
 Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsMethod method
 )
```

### Parameters

*outMap*

The destination ZMap.

*direction*

Direction in which the undefined pixels are filled in a ZMap from EFillUndefinedPixelsDirection.

*method*

Which values used to fill the undefined pixels in a ZMap from EFillUndefinedPixelsMethod.

# EZMap16::GetBufferPtr

Clears the ZMap: replaces all pixels with the undefined value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void* GetBufferPtr(
 )
```

```
void* GetBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
const void* GetBufferPtr(
 )
const void* GetBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

-

*y*

-

# EZMap16::GetCheckedBufferPtr

Retrieves the pointer to the pixel buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void* GetCheckedBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
const void* GetCheckedBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel of which we want the address.

*y*

Row of the pixel of which we want the address.

# EZMap16::GetMetadata

Returns the string value of the given metadata.
Throws an exception if it does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
std::string GetMetadata(
 const std::string& Key
 )
```

### Parameters

*Key*

The name of an exising metadata.

# EZMap16::GetPixel

Gets the value of a pixel .

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
EDepth16 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel.

*y*

Row of the pixel.

# EZMap16::GetPixelPositionFromWorldPosition

Returns in the u, v and value parameters the EZMap16 values corresponding to a 3D world position.
The world position is projected on the ZMap reference plane to get a position in the ZMap.
If the projected position is outside the ZMap, the method returns FALSE.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool GetPixelPositionFromWorldPosition(
 const E3DPoint& world_position,
 OEV_INT32& u,
 OEV_INT32& v,
 EDepth16& value
 )
```

### Parameters

*world_position*

The 3D coordinates of a world position.

*u*

Column of the ZMap pixel in [0,width[.

*v*

Row of the ZMap pixel in [0,height[.

*value*

Value of the pixel.

# EZMap16::GetResolution

Gets the resolution. For the X and Y axes, the resolution is expressed in metric units per pixel.
For the Z axis it is expressed in metric units per grey scale value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void GetResolution(
 float& sx,
 float& sy,
 float& sz
 )
E3DPoint GetResolution(
 )
```

## Parameters

*sx*

-

*sy*

-

*sz*

-

# EZMap16::GetSizeInWorld

Returns the dimensions of the EZMap16 in real world space (e.g metric unit).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void GetSizeInWorld(
 float& worldWidth,
 float& worldHeight
 )
```

### Parameters

*worldWidth*

Contains the size of the ZMap along the X axis (column).

*worldHeight*

Contains the size of the ZMap along the Y axis (row).

# EZMap16::GetWorldPositionFromPixelPosition

Returns the 3D world position corresponding to a EZMap16 pixel position. The world position is in the original point cloud space. The pixel space origin is at the upper left corner of the image and ranges to (width-1, height-1). The transformation to the world position is calculated using the center of the pixel. The pixel value at position (u,v) must not be undefined, otherwise an exception will be thrown.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DPoint GetWorldPositionFromPixelPosition(
 OEV_INT32 u,
 OEV_INT32 v
 )
```

### Parameters

*u*

Column of the pixel (bounds: [0,width[).

*v*

Row of the pixel (bounds: [0,height[).

# EZMap16::GetZMapPositionFromPixelPosition

Returns the corresponding EZMap16 3D position of a ZMap pixel. The ZMap position is in the original point ZMap space. The pixel position origin is the upper left corner of the image and ranges to (width-1, height-1). The center of the pixel is used for the transformation to the ZMap position. Pixel at position (u,v) must be defined, otherwise an exception will be thrown.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPoint GetZMapPositionFromPixelPosition(
 OEV_INT32 u,
 OEV_INT32 v
 )
```

### Parameters

*u*

Column of the pixel (bounds: [0,width[).

*v*

Row of the pixel (bounds: [0,height[).

# EZMap16::GetZValue

Gets Z value (in metric coordinate) at pixel coordinates.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
float GetZValue(
 const OEV_INT32 x,
 const OEV_INT32 y
 )
```

### Parameters

*x*

X Coordinate.

*y*

Y Coordinate.

# EZMap16::GetHeight

# EZMap16::SetHeight

Access ZMap Height.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetHeight() const
void SetHeight(OEV_INT32 height)
```

# EZMap16::ImageToWorld

Transforms a floating point (sub)pixel image position to a 3D world position.
The image space origin is at the upper left corner of the image and the X/Y unit size is one pixel.
The pixel Z value is in grey scale values (its range depends on the ZMap type).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ImageToWorld(
 const E3DPoint& pixelPt,
 E3DPoint& worldPt
 )
```

### Parameters

*pixelPt*

Position in the image space.

*worldPt*

Position in the 3D world space.

# EZMap16::ImageToZMap

Converts a 2D image (sub)pixel coordinate to the EZMap16 space.
(u,v) is the pixel position (with its origin in the upper left corner of the image).
(x,y) is the corresponding ZMap position (which has the same scale as the world space).
All values are expressed in floating point numbers.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ImageToZMap(
 float u,
 float v,
 float& x,
 float& y
 )
```

### Parameters

*u*

X Coordinate of the pixel as a floating point value.

*v*

Y Coordinate of the pixel as a floating point value.

*x*

Position along horizontal axis in the ZMap space.

*y*

Position along vertical axis in the ZMap space.

# EZMap16::IsVoid

Tests if the EZMap16 object size is zero.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
bool IsVoid(
 )
```

## Remarks

Returns **TRUE** if the ZMap size is zero.

# EZMap16::Load

Restores the EZMap16 stored in the given Open eVision file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Load(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

### Remarks

When loading, the ZMap is resized if needed. This function restores all the ZMap attributes.

# EZMap16::LoadImage

Restores the EZMap16 image stored in the given image file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void LoadImage(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

### Remarks

When loading, the ZMap is resized if needed. This function does not restore the ZMap attributes, only the image associated with the EZMap16 is updated.

# EZMap16::LoadImageAndMetadata

Loads image format and Metadata in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void LoadImageAndMetadata(
 const std::string& path,
 const std::string& pathMetadata
 )
```

### Parameters

*path*

-

*pathMetadata*

Full path to the file.

# EZMap16::LoadMetadata

Loads Metadata in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void LoadMetadata(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

# EZMap16::GetMapToWorldMatrix

Returns an E3DTransformMatrix that transforms positions from the EZMap16 space to the world space. This transformation is composed of rotation and translation only, so it is a rigid transformation and preserves distances.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const E3DTransformMatrix& GetMapToWorldMatrix() const
```

# EZMap16::ModifyMetadata

Changes an existing metadata key and value. Throws an exception if it does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ModifyMetadata(
 const std::string& Key,
 const std::string& value
 )
```

## Parameters

*Key*

-

*value*

-

# EZMap16::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EZMap16& operator=(
 const EZMap16& other
 )
```

### Parameters

*other*

The source EZMap16.

# EZMap16::GetRowPitch

Returns the buffer row pitch.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetRowPitch() const
```

# EZMap16::Save

Saves the EZMap16 object to the given Open eVision file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 const std::string& path
 )
```

### Parameters

*path*

The full path to the destination file.

### Remarks

This format save the EZMap16 in a Open eVision file. This function stores all the ZMap attributes.

# EZMap16::SaveImage

Saves the EZMap16 image to the given image file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveImage(
 const std::string& path,
 Euresys::Open_eVision_2_10::EImageFileType type
 )
```

### Parameters

*path*

The full path to the destination file.

*type*

File format, as defined by EImageFileType. If not specified, the file format is determined from the file extension.

### Remarks

This format save the image associated to EZMap16 in a standard image file and thus does not store ZMap attributes.

# EZMap16::SaveImageAndMetadata

Saves image format and Metadata JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SaveImageAndMetadata(
 const std::string& path,
 const std::string& pathMetadata,
 Euresys::Open_eVision_2_10::EImageFileType type
 )
```

### Parameters

*path*

-

*pathMetadata*

The full path to the destination file.

*type*

File format, as defined by EImageFileType. If not specified, the file format is determined from the file extension.

# EZMap16::SaveMetadata

Saves Metadata in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SaveMetadata(
 const std::string& path
 )
```

### Parameters

*path*

The full path to the destination file.

# EZMap16::Serialize

Serializes the EZMap16 object with all its attributes.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or writen to.

# EZMap16::SerializeImage

Serializes the image associated to EZMap16.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SerializeImage(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or writen to.

# EZMap16::SetBufferPtr

Sets the pointer to an externally allocated image buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetBufferPtr(
 OEV_INT32 width,
 OEV_INT32 height,
 void* imagePointer,
 OEV_INT32 bitsPerRow
 )
```

### Parameters

*width*

The width of the supplied buffer, in pixels.

*height*

The height of the supplied buffer, in pixels.

*imagePointer*

The pointer (aligned on 4 bytes) to the buffer, which must be large enough to hold the data.

*bitsPerRow*

The total number of bits contained in a row, padding included.
Using the value **0** (default) means that this size is computed from the buffer width and the pixel size plus a padding with the smallest possible value that leads to a multiple of 4 bytes (32 bits), which is the minimum padding accepted by EZMap16::SetBufferPtr.

# EZMap16::SetPixel

Sets the value of a pixel .

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SetPixel(
 EDepth16 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

Value of the pixel.

*x*

Column of the pixel.

*y*

Row of the pixel.

# EZMap16::SetResolution

Sets the resolution. For the X and Y axes, the resolution is expressed in metric units per pixel.
For the Z axis it is expressed in metric units per grey scale value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SetResolution(
 float rx,
 float ry,
 float rz
 )

void SetResolution(
 E3DPoint resolution
 )
```

### Parameters

*rx*

-

*ry*

-

*rz*

-

*resolution*

-

# EZMap16::SetSize

Sets the width and height of the EZMap16.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetSize(
 OEV_INT32 width,
 OEV_INT32 height
 )
void SetSize(
 const EZMap& other
 )
```

## Parameters

*width*

The new requested width.

*height*

The new requested height.

*other*

The other ZMap whose dimensions have to be used for the current object.

### Remarks

Open eVision will allocate a new image buffer (deallocate the old image buffer) if the supplied width and height are different from the existing ones.
If an external buffer has been specified by means of **SetImagePtr**, it will be kept only if the size does not change.
Creating a new Open eVision image buffer and setting its size creates a 4-byte aligned buffer, by default. The *size of an ZMap* is specified as a number of columns (width) and rows (height).
The maximum image dimensions are 32767 by 32767. Furthermore, it must fit into the available memory, that depends upon the physical memory, the operating system and the memory already allocated by the process in other modules or libraries.

# EZMap16::SetZValue

Sets Z value (in metric coordinate) at pixel coordinates.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetZValue(
 const float value,
 const OEV_INT32 x,
 const OEV_INT32 y
 )
```

### Parameters

*value*

Value of the pixel in metric space.

*x*

X Coordinate.

*y*

Y Coordinate.

# EZMap16::GetType

Pixel accessor type.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
Euresys::Open_eVision_2_10::EImageType GetType() const
```

# EZMap16::GetUndefinedValue

Returns the Undefined value. That value is used to set to mark pixels with no valid depth value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EDepth16 GetUndefinedValue() const
```

# EZMap16::GetWidth

# EZMap16::SetWidth

Access ZMap Width.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetWidth() const
void SetWidth(OEV_INT32 width)
```

# EZMap16::GetWorldShape

Returns the EWorldShape for conversion between 2D image and 2D world space coordinates. This EWorldShape can be used by EasyGauge to do measurements on a EZMap16 in real space coordinates (e.g mm).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const EWorldShape& GetWorldShape() const
```

# EZMap16::WorldToImage

Transforms a 3D world position to a floating point (sub)pixel image position.
The image space origin is at the upper left corner of the image and the X/Y unit size is one pixel.
The Z value is given in grey scale value.
Returns TRUE if the pixel position is inside the image limits and the Z value is positive.
The parameter pixelPt is filled even if the point is outside the image.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool WorldToImage(
 const E3DPoint& worldPt,
 E3DPoint& pixelPt
 )
```

## Parameters

*worldPt*

Position in the 3D world space.

*pixelPt*

Position in the image space.

# EZMap16::GetWorldToMapMatrix

Returns an E3DTransformMatrix that transforms positions from the world space to the EZMap16 space. This transformation is composed of rotation and translation only, so it is a rigid transformation and it preserves distances.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const E3DTransformMatrix& GetWorldToMapMatrix() const
```

# EZMap16::WorldToZMap

Transforms a 3D world position to a 3D EZMap16 position.
The ZMap space origin is at the lower left corner of the image.
The scales of the world space and ZMap space are the same.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void WorldToZMap(
 const E3DPoint& worldPt,
 E3DPoint& zmapPt
 )
```

## Parameters

*worldPt*

Position in the 3D world space.

*zmapPt*

Position in the ZMap space.

### Remarks

Do not use this method with the same variable as input and output. It might lead to incorrect results.

# EZMap16::GetXResolution

# EZMap16::SetXResolution

Resolution of the EZMap16 along the X axis
The resolution is the number of metric units per pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetXResolution() const
void SetXResolution(float resolution)
```

# EZMap16::GetYResolution

# EZMap16::SetYResolution

Resolution of the EZMap16 along the Y axis
The resolution is the number of metric units per pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetYResolution() const
void SetYResolution(float resolution)
```

# EZMap16::ZMapToImage

Converts a 2D coordinate in the EZMap16 space to image (sub)pixel space.
(x,y) is the ZMap position (which has the same scale as the world space).
(u,v) is the corresponding pixel position (with its origin in the upper left corner of the image).
All values are expressed in floating point numbers.
Returns TRUE if the pixel position is inside the image limits.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool ZMapToImage(
 float x,
 float y,
 float& u,
 float& v
 )
```

### Parameters

*x*

Position along horizontal axis in the ZMap space.

*y*

Position along vertical axis in the ZMap space.

*u*

Column of the pixel as a floating point value.

*v*

Row of the pixel as a floating point value.

# EZMap16::ZMapToWorld

Transforms a 3D EZMap16 position to a 3D world space position.
The ZMap space origin is at the lower left corner of the image.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ZMapToWorld(
 const E3DPoint& zmapPt,
 E3DPoint& worldPt
 )
```

### Parameters

*zmapPt*

Position in the ZMap space.

*worldPt*

Position in the 3D world space.

### Remarks

Do not use this method with the same variable as input and output. It might lead to incorrect results.

# EZMap16::GetZResolution

# EZMap16::SetZResolution

Resolution of the EZMap16 along the Z axis
The resolution is the number of grey values per pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetZResolution() const
void SetZResolution(float resolution)
```

# 4.196. EZMap32f Class

A ZMap32f is a 32bits corrected 2.5D image.
ZMap pixel values (32 bits floating point numbers) represent distances from a 3D reference plane.
Distances are positive, during the ZMap generation all points below the reference plane are discarded.
The EZMap class also stores the transformation from the pixel coordinates to the real world coordinate system.
There could be undefined pixels in the ZMap.

**Base Class:** EZMap

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| AddMetadata | Adds a metadata key (name) and value.<br>If the metadata key already exists, its value will be overwritten. |
| AsEImage | Returns the EZMap32f as an EImageBW32 (32 bits gray scale) to use with existing eVision 2D tools. |
| Clear | Clears the ZMap: replaces all pixels with the undefined value. |
| ClearMetadata | Deletes all metadata. |
| ConvertCoordinatesMapToPixel | Converts 3D Map coordinates to Buffer coordinates |
| ConvertCoordinatesPixelToMap | Converts Buffer coordinates to 3D Map coordinates |

| | |
|---|---|
| CopyMetadataTo | Copies all metadata. |
| DeleteMetadata | Deletes value of this existing metadata key . Throws an exception if it does not exist. |
| Draw | Draws a EZMap32f in a device context. |
| DrawImage | Displays the internal image buffer |
| EZMap32f | Creates a 32 bits EZMap. |
| FillUndefinedPixels | Fills undefined pixels, used to fill the "holes" in the ZMap. |
| GetBufferPtr | Clears the ZMap: replaces all pixels with the undefined value. |
| GetCheckedBufferPtr | Retrieves the pointer to the pixel buffer. |
| GetHeight | Access ZMap Height. |
| GetMapToWorldMatrix | Returns an E3DTransformMatrix that transforms positions from the EZMap32f space to the world space. This transformation is composed of rotation and translation only, so it is a rigid transformation and preserves distances. |
| GetMetadata | Returns the string value of the given metadata.<br>Throws an exception if it does not exist. |

| | |
|---|---|
| GetPixel | Gets the value of a pixel . |
| GetPixelPositionFromWorldPosition | Returns in the u, v and value parameters the EZMap32f values corresponding to a 3D world position.<br>The world position is projected on the ZMap reference plane to get a position in the ZMap.<br>If the projected position is outside the ZMap, the method returns FALSE. |
| GetResolution | Gets the resolution. For the X and Y axes, the resolution is expressed in metric units per pixel.<br>For the Z axis it is expressed in metric units per grey scale value. |
| GetRowPitch | Returns the buffer row pitch. |
| GetSizeInWorld | Returns the dimensions of the EZMap32f in real world space (e.g metric unit). |
| GetType | Pixel accessor type. |
| GetUndefinedValue | Returns the Undefined value. That value is used to set to mark pixels with no valid depth value. |
| GetWidth | Access ZMap Width. |

| | |
|---|---|
| GetWorldPositionFromPixelPosition | Returns the 3D world position corresponding to a EZMap32f pixel position. The world position is in the original point cloud space. The pixel space origin is at the upper left corner of the image and ranges to (width-1, height-1). The transformation to the world position is calculated using the center of the pixel. The pixel value at position (u,v) must not be undefined, otherwise an exception will be thrown. |
| GetWorldShape | Returns the EWorldShape for conversion between 2D image and 2D world space coordinates. This EWorldShape can be used by EasyGauge to do measurements on a EZMap32f in real space coordinates (e.g mm). |
| GetWorldToMapMatrix | Returns an E3DTransformMatrix that transforms positions from the world space to the EZMap32f space. This transformation is composed of rotation and translation only, so it is a rigid transformation and it preserves distances. |
| GetXResolution | Resolution of the EZMap32f along the X axis<br>The resolution is the number of metric units per pixel. |
| GetYResolution | Resolution of the EZMap32f along the Y axis<br>The resolution is the number of metric units per pixel. |
| GetZMapPositionFromPixelPosition | Returns the corresponding EZMap32f 3D position of a ZMap pixel. The ZMap position is in the original point ZMap space. The pixel position origin is the upper left corner of the image and ranges to (width-1, height-1). The center of the pixel is used for the transformation to the ZMap position. Pixel at position (u,v) must be defined, otherwise an exception will be thrown. |

| | |
|---|---|
| GetZResolution | Resolution of the EZMap32f along the Z axis<br>The resolution is the number of grey values per pixel. |
| GetZValue | Gets Z value (in metric coordinate) at pixel coordinates. |
| ImageToWorld | Transforms a floating point (sub)pixel image position to a 3D world position.<br>The image space origin is at the upper left corner of the image and the X/Y unit size is one pixel.<br>The pixel Z value is in grey scale values (its range depends on the ZMap type). |
| ImageToZMap | Converts a 2D image (sub)pixel coordinate to the EZMap32f space.<br>(u,v) is the pixel position (with its origin in the upper left corner of the image).<br>(x,y) is the corresponding ZMap position (which has the same scale as the world space).<br>All values are expressed in floating point numbers. |
| IsVoid | Tests if the EZMap32f object size is zero. |
| Load | Restores the EZMap32f stored in the given Open eVision file. |
| LoadImage | Restores the EZMap32f image stored in the given image file. |
| LoadImageAndMetadata | Loads image format and Metadata in JSON format. |
| LoadMetadata | Loads Metadata in JSON format. |

| | |
|---|---|
| ModifyMetadata | Changes an existing metadata key and value. Throws an exception if it does not exist. |
| operator= | Assignment operator. |
| Save | Saves the EZMap32f object to the given Open eVision file. |
| SaveImage | Saves the EZMap32f image to the given image file. |
| SaveImageAndMetadata | Saves image format and Metadata JSON format. |
| SaveMetadata | Saves Metadata in JSON format. |
| Serialize | Serializes the EZMap32f object with all its attributes. |
| SerializeImage | Serializes the image associated to EZMap32f. |
| SetBufferPtr | Sets the pointer to an externally allocated image buffer. |
| SetHeight | Access ZMap Height. |
| SetPixel | Sets the value of a pixel . |

| | |
|---|---|
| SetResolution | Sets the resolution. For the X and Y axes, the resolution is expressed in metric units per pixel.<br>For the Z axis it is expressed in metric units per grey scale value. |
| SetSize | Sets the width and height of the EZMap32f. |
| SetWidth | Access ZMap Width. |
| SetXResolution | Resolution of the EZMap32f along the X axis<br>The resolution is the number of metric units per pixel. |
| SetYResolution | Resolution of the EZMap32f along the Y axis<br>The resolution is the number of metric units per pixel. |
| SetZResolution | Resolution of the EZMap32f along the Z axis<br>The resolution is the number of grey values per pixel. |
| SetZValue | Sets Z value (in metric coordinate) at pixel coordinates. |
| WorldToImage | Transforms a 3D world position to a floating point (sub)pixel image position.<br>The image space origin is at the upper left corner of the image and the X/Y unit size is one pixel.<br>The Z value is given in grey scale value.<br>Returns TRUE if the pixel position is inside the image limits and the Z value is positive.<br>The parameter pixelPt is filled even if the point is outside the image. |

| | |
|---|---|
| WorldToZMap | Transforms a 3D world position to a 3D EZMap32f position.<br>The ZMap space origin is at the lower left corner of the image.<br>The scales of the world space and ZMap space are the same. |
| ZMapToImage | Converts a 2D coordinate in the EZMap32f space to image (sub-)pixel space.<br>(x,y) is the ZMap position (which has the same scale as the world space).<br>(u,v) is the corresponding pixel position (with its origin in the upper left corner of the image).<br>All values are expressed in floating point numbers.<br>Returns TRUE if the pixel position is inside the image limits. |
| ZMapToWorld | Transforms a 3D EZMap32f position to a 3D world space position.<br>The ZMap space origin is at the lower left corner of the image. |

E

Z

# Map32f::AddMetadata

Adds a metadata key (name) and value.
If the metadata key already exists, its value will be overwritten.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void AddMetadata(
 const std::string& Key,
 const std::string& value
 )
```

## Parameters

*Key*

The name of the metadata. Names are unique.

*value*

The value for the given metadata.

# EZMap32f::AsEImage

Returns the EZMap32f as an EImageBW32 (32 bits gray scale) to use with existing eVision 2D tools.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
EImageBW32f& AsEImage(
 )

const EImageBW32f& AsEImage(
 )
```

# EZMap32f::Clear

Clears the ZMap: replaces all pixels with the undefined value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Clear(
 )
```

# EZMap32f::ClearMetadata

Deletes all metadata.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ClearMetadata(
 )
```

# EZMap32f::ConvertCoordinatesMapToPixel

Converts 3D Map coordinates to Buffer coordinates

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool ConvertCoordinatesMapToPixel(
 float x3D,
 float y3D,
 int& xBuffer,
 int& yBuffer
 )
```

## Parameters

*x3D*

The Map X coordinate.

*y3D*

The Map Y coordinate.

*xBuffer*

The returned Pixel X coordinate.

*yBuffer*

The returned Pixel Y coordinate.

# EZMap32f::ConvertCoordinatesPixelToMap

Converts Buffer coordinates to 3D Map coordinates

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ConvertCoordinatesPixelToMap(
 int xBuffer,
 int yBuffer,
 float& x3D,
 float& y3D
 )
```

## Parameters

*xBuffer*

The pixel X coordinate.

*yBuffer*

The pixel Y coordinate.

*x3D*

The returned Map X coordinate.

*y3D*

The returned Map Y coordinate.

# EZMap32f::CopyMetadataTo

Copies all metadata.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void CopyMetadataTo(
 EZMap32f& other
 )
```

### Parameters

*other*

An other EZMap32f.

# EZMap32f::DeleteMetadata

Deletes value of this existing metadata key . Throws an exception if it does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void DeleteMetadata(
 const std::string& Key
 )
```

### Parameters

*Key*

-

# EZMap32f::Draw

Draws a EZMap32f in a device context.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void Draw(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

```
void Draw(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the ZMap is displayed in 1:1 scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Pan offset (in pixels) in the horizontal direction. By default, no panning is applied.

*panY*

Pan offset (in pixels) in the vertical direction. By default, no panning is applied.

*colorUndefinedPixel*

An optional parameter to choose the drawing color of undefined pixels.

*c24Vector*

When supplied, this parameter allows using a LUT that maps from Depth to C24 when drawing (false colors).

*bw8Vector*

When supplied, this parameter allows using a LUT that maps from Depth to BW8 when drawing.

### Remarks

A ZMap can be drawn (its pixels rendered) using a device context.
The horizontal and vertical zooming factors can be different but must be in the **1/16..16** range.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EZMap32f::DrawImage

Displays the internal image buffer

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void DrawImage(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

```
void DrawImage(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the image is displayed in 1:1 scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Pan offset (in pixels) in the horizontal direction. By default, no panning is applied.

*panY*

Pan offset (in pixels) in the vertical direction. By default, no panning is applied.

*colorUndefinedPixel*

An optional parameter to choose the drawing color of undefined pixels.

*c24Vector*

When supplied, this parameter allows using a LUT that maps from Depth to C24 when drawing (false colors).

*bw8Vector*

When supplied, this parameter allows using a LUT that maps from Depth to BW8 when drawing.

### Remarks

An image can be drawn (its pixels rendered) using a device context.
The horizontal and vertical zooming factors can be different but must be in the **1/16..16** range.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EZMap32f::EZMap32f

Creates a 32 bits EZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EZMap32f(
 )
void EZMap32f(
 OEV_INT32 width,
 OEV_INT32 height
 )
void EZMap32f(
 const EZMap32f& other
 )
```

### Parameters

*width*

The width of the new Zmap.

*height*

The height of the new Zmap.

*other*

Another Zmap.

# EZMap32f::FillUndefinedPixels

Fills undefined pixels, used to fill the "holes" in the ZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void FillUndefinedPixels(
 EZMap32f& outMap,
 Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsDirection direction,
 Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsMethod method
 )
```

### Parameters

*outMap*

The destination ZMap.

*direction*

Direction in which the undefined pixels are filled in a ZMap from EFillUndefinedPixelsDirection.

*method*

Which values used to fill the undefined pixels in a ZMap from EFillUndefinedPixelsMethod.

# EZMap32f::GetBufferPtr

Clears the ZMap: replaces all pixels with the undefined value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void* GetBufferPtr(
 )
```

```
void* GetBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )

const void* GetBufferPtr(
 )

const void* GetBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

-

*y*

-

# EZMap32f::GetCheckedBufferPtr

Retrieves the pointer to the pixel buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void* GetCheckedBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )

const void* GetCheckedBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel of which we want the address.

*y*

Row of the pixel of which we want the address.

# EZMap32f::GetMetadata

Returns the string value of the given metadata.
Throws an exception if it does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
std::string GetMetadata(
 const std::string& Key
 )
```

### Parameters

*Key*

The name of an exising metadata.

# EZMap32f::GetPixel

Gets the value of a pixel .

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
EDepth32f GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel.

*y*

Row of the pixel.

# EZMap32f::GetPixelPositionFromWorldPosition

Returns in the u, v and value parameters the EZMap32f values corresponding to a 3D world position.
The world position is projected on the ZMap reference plane to get a position in the ZMap.
If the projected position is outside the ZMap, the method returns FALSE.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool GetPixelPositionFromWorldPosition(
 const E3DPoint& world_position,
 OEV_INT32& u,
 OEV_INT32& v,
 EDepth32f& value
 )
```

### Parameters

*world_position*

The 3D coordinates of a world position.

*u*

Column of the ZMap pixel in [0,width[.

*v*

Row of the ZMap pixel in [0,height[.

*value*

Value of the pixel.

# EZMap32f::GetResolution

Gets the resolution. For the X and Y axes, the resolution is expressed in metric units per pixel.
For the Z axis it is expressed in metric units per grey scale value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void GetResolution(
 float& sx,
 float& sy,
 float& sz
 )
E3DPoint GetResolution(
 )
```

## Parameters

*sx*

-

*sy*

-

*sz*

-

# EZMap32f::GetSizeInWorld

Returns the dimensions of the EZMap32f in real world space (e.g metric unit).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void GetSizeInWorld(
 float& worldWidth,
 float& worldHeight
 )
```

### Parameters

*worldWidth*

Contains the size of the ZMap along the X axis (column).

*worldHeight*

Contains the size of the ZMap along the Y axis (row).

# EZMap32f::GetWorldPositionFromPixelPosition

Returns the 3D world position corresponding to a EZMap32f pixel position. The world position is in the original point cloud space. The pixel space origin is at the upper left corner of the image and ranges to (width-1, height-1). The transformation to the world position is calculated using the center of the pixel. The pixel value at position (u,v) must not be undefined, otherwise an exception will be thrown.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DPoint GetWorldPositionFromPixelPosition(
 OEV_INT32 u,
 OEV_INT32 v
 )
```

### Parameters

*u*

Column of the pixel (bounds: [0,width[).

*v*

Row of the pixel (bounds: [0,height[).

# EZMap32f::GetZMapPositionFromPixelPosition

Returns the corresponding EZMap32f 3D position of a ZMap pixel. The ZMap position is in the original point ZMap space. The pixel position origin is the upper left corner of the image and ranges to (width-1, height-1). The center of the pixel is used for the transformation to the ZMap position. Pixel at position (u,v) must be defined, otherwise an exception will be thrown.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
E3DPoint GetZMapPositionFromPixelPosition(
 OEV_INT32 u,
 OEV_INT32 v
 )
```

### Parameters

*u*

Column of the pixel (bounds: [0,width[).

*v*

Row of the pixel (bounds: [0,height[).

# EZMap32f::GetZValue

Gets Z value (in metric coordinate) at pixel coordinates.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
```

```
float GetZValue(
 const OEV_INT32 x,
 const OEV_INT32 y
 )
```

### Parameters

*x*

X Coordinate.

*y*

Y Coordinate.

# EZMap32f::GetHeight

# EZMap32f::SetHeight

Access ZMap Height.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
OEV_INT32 GetHeight() const
void SetHeight(OEV_INT32 height)
```

# EZMap32f::ImageToWorld

Transforms a floating point (sub)pixel image position to a 3D world position.
The image space origin is at the upper left corner of the image and the X/Y unit size is one pixel.
The pixel Z value is in grey scale values (its range depends on the ZMap type).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ImageToWorld(
 const E3DPoint& pixelPt,
 E3DPoint& worldPt
 )
```

### Parameters

*pixelPt*

Position in the image space.

*worldPt*

Position in the 3D world space.

# EZMap32f::ImageToZMap

Converts a 2D image (sub)pixel coordinate to the EZMap32f space.
(u,v) is the pixel position (with its origin in the upper left corner of the image).
(x,y) is the corresponding ZMap position (which has the same scale as the world space).
All values are expressed in floating point numbers.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ImageToZMap(
 float u,
 float v,
 float& x,
 float& y
 )
```

### Parameters

*u*

X Coordinate of the pixel as a floating point value.

*v*

Y Coordinate of the pixel as a floating point value.

*x*

Position along horizontal axis in the ZMap space.

*y*

Position along vertical axis in the ZMap space.

# EZMap32f::IsVoid

Tests if the EZMap32f object size is zero.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool IsVoid(
 )
```

## Remarks

Returns **TRUE** if the ZMap size is zero.

# EZMap32f::Load

Restores the EZMap32f stored in the given Open eVision file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

### Remarks

When loading, the ZMap is resized if needed. This function restores all the ZMap attributes.

# EZMap32f::LoadImage

Restores the EZMap32f image stored in the given image file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void LoadImage(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

### Remarks

When loading, the ZMap is resized if needed. This function does not restore the ZMap attributes, only the image associated with the EZMap32f is updated.

# EZMap32f::LoadImageAndMetadata

Loads image format and Metadata in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void LoadImageAndMetadata(
 const std::string& path,
 const std::string& pathMetadata
 )
```

### Parameters

*path*

-

*pathMetadata*

Full path to the file.

# EZMap32f::LoadMetadata

Loads Metadata in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void LoadMetadata(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

# EZMap32f::GetMapToWorldMatrix

Returns an E3DTransformMatrix that transforms positions from the EZMap32f space to the world space. This transformation is composed of rotation and translation only, so it is a rigid transformation and preserves distances.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const E3DTransformMatrix& GetMapToWorldMatrix() const
```

# EZMap32f::ModifyMetadata

Changes an existing metadata key and value. Throws an exception if it does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ModifyMetadata(
 const std::string& Key,
 const std::string& value
 )
```

## Parameters

*Key*

-

*value*

-

# EZMap32f::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EZMap32f& operator=(
 const EZMap32f& other
 )
```

### Parameters

*other*

The source EZMap32f.

# EZMap32f::GetRowPitch

Returns the buffer row pitch.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetRowPitch() const
```

# EZMap32f::Save

Saves the EZMap32f object to the given Open eVision file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 const std::string& path
 )
```

### Parameters

*path*

The full path to the destination file.

### Remarks

This format save the EZMap32f in a Open eVision file. This function stores all the ZMap attributes.

# EZMap32f::SaveImage

Saves the EZMap32f image to the given image file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveImage(
 const std::string& path,
 Euresys::Open_eVision_2_10::EImageFileType type
 )
```

### Parameters

*path*

The full path to the destination file.

*type*

File format, as defined by EImageFileType. If not specified, the file format is determined from the file extension.

### Remarks

This format save the image associated to EZMap32f in a standard image file and thus does not store ZMap attributes.

# EZMap32f::SaveImageAndMetadata

Saves image format and Metadata JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveImageAndMetadata(
 const std::string& path,
 const std::string& pathMetadata,
 Euresys::Open_eVision_2_10::EImageFileType type
 )
```

### Parameters

*path*

-

*pathMetadata*

The full path to the destination file.

*type*

File format, as defined by EImageFileType. If not specified, the file format is determined from the file extension.

# EZMap32f::SaveMetadata

Saves Metadata in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveMetadata(
 const std::string& path
 )
```

### Parameters

*path*

The full path to the destination file.

# EZMap32f::Serialize

Serializes the EZMap32f object with all its attributes.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or writen to.

# EZMap32f::SerializeImage

Serializes the image associated to EZMap32f.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SerializeImage(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or writen to.

# EZMap32f::SetBufferPtr

Sets the pointer to an externally allocated image buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetBufferPtr(
 OEV_INT32 width,
 OEV_INT32 height,
 void* imagePointer,
 OEV_INT32 bitsPerRow
 )
```

### Parameters

*width*

The width of the supplied buffer, in pixels.

*height*

The height of the supplied buffer, in pixels.

*imagePointer*

The pointer (aligned on 4 bytes) to the buffer, which must be large enough to hold the data.

*bitsPerRow*

The total number of bits contained in a row, padding included.
Using the value **0** (default) means that this size is computed from the buffer width and the pixel size plus a padding with the smallest possible value that leads to a multiple of 4 bytes (32 bits), which is the minimum padding accepted by EZMap32f::SetBufferPtr.

# EZMap32f::SetPixel

Sets the value of a pixel .

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetPixel(
 EDepth32f value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

Value of the pixel.

*x*

Column of the pixel.

*y*

Row of the pixel.

# EZMap32f::SetResolution

Sets the resolution. For the X and Y axes, the resolution is expressed in metric units per pixel.
For the Z axis it is expressed in metric units per grey scale value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetResolution(
 float rx,
 float ry,
 float rz
 )
void SetResolution(
 E3DPoint resolution
 )
```

### Parameters

*rx*

-

*ry*

-

*rz*

-

*resolution*

-

# EZMap32f::SetSize

Sets the width and height of the EZMap32f.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetSize(
 OEV_INT32 width,
 OEV_INT32 height
 )
void SetSize(
 const EZMap& other
 )
```

## Parameters

*width*

The new requested width.

*height*

The new requested height.

*other*

The other ZMap whose dimensions have to be used for the current object.

### Remarks

Open eVision will allocate a new image buffer (deallocate the old image buffer) if the supplied width and height are different from the existing ones.
If an external buffer has been specified by means of **SetImagePtr**, it will be kept only if the size does not change.
Creating a new Open eVision image buffer and setting its size creates a 4-byte aligned buffer, by default. The *size of an ZMap* is specified as a number of columns (width) and rows (height).
The maximum image dimensions are 32767 by 32767. Furthermore, it must fit into the available memory, that depends upon the physical memory, the operating system and the memory already allocated by the process in other modules or libraries.

# EZMap32f::SetZValue

Sets Z value (in metric coordinate) at pixel coordinates.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetZValue(
 const float value,
 const OEV_INT32 x,
 const OEV_INT32 y
 )
```

### Parameters

*value*

Value of the pixel in metric space.

*x*

X Coordinate.

*y*

Y Coordinate.

# EZMap32f::GetType

Pixel accessor type.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
Euresys::Open_eVision_2_10::EImageType GetType() const
```

# EZMap32f::GetUndefinedValue

Returns the Undefined value. That value is used to set to mark pixels with no valid depth value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EDepth32f GetUndefinedValue() const
```

# EZMap32f::GetWidth

# EZMap32f::SetWidth

Access ZMap Width.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetWidth() const
void SetWidth(OEV_INT32 width)
```

# EZMap32f::GetWorldShape

Returns the EWorldShape for conversion between 2D image and 2D world space coordinates. This EWorldShape can be used by EasyGauge to do measurements on a EZMap32f in real space coordinates (e.g mm).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const EWorldShape& GetWorldShape() const
```

# EZMap32f::WorldToImage

Transforms a 3D world position to a floating point (sub)pixel image position.
The image space origin is at the upper left corner of the image and the X/Y unit size is one pixel.
The Z value is given in grey scale value.
Returns TRUE if the pixel position is inside the image limits and the Z value is positive.
The parameter pixelPt is filled even if the point is outside the image.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool WorldToImage(
 const E3DPoint& worldPt,
 E3DPoint& pixelPt
 )
```

## Parameters

*worldPt*

Position in the 3D world space.

*pixelPt*

Position in the image space.

# EZMap32f::GetWorldToMapMatrix

Returns an E3DTransformMatrix that transforms positions from the world space to the EZMap32f space. This transformation is composed of rotation and translation only, so it is a rigid transformation and it preserves distances.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const E3DTransformMatrix& GetWorldToMapMatrix() const
```

# EZMap32f::WorldToZMap

Transforms a 3D world position to a 3D EZMap32f position.
The ZMap space origin is at the lower left corner of the image.
The scales of the world space and ZMap space are the same.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void WorldToZMap(
 const E3DPoint& worldPt,
 E3DPoint& zmapPt
 )
```

## Parameters

*worldPt*

Position in the 3D world space.

*zmapPt*

Position in the ZMap space.

### Remarks

Do not use this method with the same variable as input and output. It might lead to incorrect results.

# EZMap32f::GetXResolution

# EZMap32f::SetXResolution

Resolution of the EZMap32f along the X axis
The resolution is the number of metric units per pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetXResolution() const
void SetXResolution(float resolution)
```

# EZMap32f::GetYResolution

# EZMap32f::SetYResolution

Resolution of the EZMap32f along the Y axis
The resolution is the number of metric units per pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetYResolution() const
void SetYResolution(float resolution)
```

# EZMap32f::ZMapToImage

Converts a 2D coordinate in the EZMap32f space to image (sub)pixel space.
(x,y) is the ZMap position (which has the same scale as the world space).
(u,v) is the corresponding pixel position (with its origin in the upper left corner of the image).
All values are expressed in floating point numbers.
Returns TRUE if the pixel position is inside the image limits.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool ZMapToImage(
 float x,
 float y,
 float& u,
 float& v
 )
```

### Parameters

*x*

Position along horizontal axis in the ZMap space.

*y*

Position along vertical axis in the ZMap space.

*u*

Column of the pixel as a floating point value.

*v*

Row of the pixel as a floating point value.

# EZMap32f::ZMapToWorld

Transforms a 3D EZMap32f position to a 3D world space position.
The ZMap space origin is at the lower left corner of the image.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void ZMapToWorld(
 const E3DPoint& zmapPt,
 E3DPoint& worldPt
 )
```

### Parameters

*zmapPt*

Position in the ZMap space.

*worldPt*

Position in the 3D world space.

### Remarks

Do not use this method with the same variable as input and output. It might lead to incorrect results.

# EZMap32f::GetZResolution

# EZMap32f::SetZResolution

Resolution of the EZMap32f along the Z axis
The resolution is the number of grey values per pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetZResolution() const
```

```
void SetZResolution(float resolution)
```

# 4.197. EZMap8 Class

A ZMap8 is a 8bits corrected 2.5D image.
ZMap Pixel values (8 bits integers) represent distances from a 3D reference plane.
Distances are positive, during the ZMap generation all points below the reference plane are discarded.
The EZMap class also stores the transformation from the pixel coordinates to the real world coordinate system.
There could be undefined pixels in the ZMap.

**Base Class:** EZMap

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| AddMetadata | Adds a metadata key (name) and value.<br>If the metadata key already exists, its value will be overwritten. |
| AsEImage | Returns the EZMap8 as an EImageBW8 (8 bits gray scale) to use with existing eVision 2D tools. |
| Clear | Clears the ZMap: replaces all pixels with the undefined value. |
| ClearMetadata | Deletes all metadata. |
| ConvertCoordinatesMapToPixel | Converts 3D Map coordinates to Buffer coordinates |
| ConvertCoordinatesPixelToMap | Converts Buffer coordinates to 3D Map coordinates |

| | |
|---|---|
| CopyMetadataTo | Copies all metadata. |
| DeleteMetadata | Deletes value of this existing metadata key . Throws an exception if it does not exist. |
| Draw | Draws a EZMap8 in a device context. |
| DrawImage | Displays the internal image buffer |
| EZMap8 | Creates a 8 bits EZMap. |
| FillUndefinedPixels | Fills undefined pixels, used to fill the "holes" in the ZMap. |
| GetBufferPtr | Retrieves the pointer to the internal pixel buffer. |
| GetCheckedBufferPtr | Retrieves the pointer to the pixel buffer. |
| GetHeight | Access ZMap Height. |
| GetMapToWorldMatrix | Returns an E3DTransformMatrix that transforms positions from the EZMap8 space to the world space. This transformation is composed of rotation and translation only, so it is a rigid transformation and preserves distances. |
| GetMetadata | Returns the string value of the given metadata.<br>Throws an exception if it does not exist. |

| | |
|---|---|
| GetPixel | Gets the value of a pixel . |
| GetPixelPositionFromWorldPosition | Returns in the u, v and value parameters the EZMap8 values corresponding to a 3D world position.<br>The world position is projected on the ZMap reference plane to get a position in the ZMap.<br>If the projected position is outside the ZMap, the method returns FALSE. |
| GetResolution | Gets the resolution. For the X and Y axes, the resolution is expressed in metric units per pixel.<br>For the Z axis it is expressed in metric units per grey scale value. |
| GetRowPitch | Returns the buffer row pitch. |
| GetSizeInWorld | Returns the dimensions of the EZMap8 in real world space (e.g metric unit). |
| GetType | Pixel accessor type. |
| GetUndefinedValue | Returns the Undefined value. That value is used to set to mark pixels with no valid depth value. |
| GetWidth | Access ZMap Width. |

| | |
|---|---|
| GetWorldPositionFromPixelPosition | Returns the 3D world position corresponding to a EZMap8 pixel position. The world position is in the original point cloud space. The pixel space origin is at the upper left corner of the image and ranges to (width-1, height-1). The transformation to the world position is calculated using the center of the pixel. The pixel value at position (u,v) must not be undefined, otherwise an exception will be thrown. |
| GetWorldShape | Returns the EWorldShape for conversion between 2D image and 2D world space coordinates. This EWorldShape can be used by EasyGauge to do measurements on a EZMap8 in real space coordinates (e.g mm). |
| GetWorldToMapMatrix | Returns an E3DTransformMatrix that transforms positions from the world space to the EZMap8 space. This transformation is composed of rotation and translation only, so it is a rigid transformation and it preserves distances. |
| GetXResolution | Resolution of the EZMap8 along the X axis<br>The resolution is the number of metric units per pixel. |
| GetYResolution | Resolution of the EZMap8 along the Y axis<br>The resolution is the number of metric units per pixel. |
| GetZMapPositionFromPixelPosition | Returns the corresponding EZMap8 3D position of a ZMap pixel. The ZMap position is in the original point ZMap space. The pixel position origin is the upper left corner of the image and ranges to (width-1, height-1). The center of the pixel is used for the transformation to the ZMap position. Pixel at position (u,v) must be defined, otherwise an exception will be thrown. |

| | |
|---|---|
| GetZResolution | Resolution of the EZMap8 along the Z axis<br>The resolution is the number of grey values per pixel. |
| GetZValue | Gets Z value (in metric coordinate) at pixel coordinates. |
| ImageToWorld | Transforms a floating point (sub)pixel image position to a 3D world position.<br>The image space origin is at the upper left corner of the image and the X/Y unit size is one pixel.<br>The pixel Z value is in grey scale values (its range depends on the ZMap type). |
| ImageToZMap | Converts a 2D image (sub)pixel coordinate to the EZMap8 space.<br>(u,v) is the pixel position (with its origin in the upper left corner of the image).<br>(x,y) is the corresponding ZMap position (which has the same scale as the world space).<br>All values are expressed in floating point numbers. |
| IsVoid | Tests if the EZMap8 object size is zero. |
| Load | Restores the EZMap8 stored in the given Open eVision file. |
| LoadImage | Restores the EZMap8 image stored in the given image file. |
| LoadImageAndMetadata | Loads image format and Metadata in JSON format. |
| LoadMetadata | Loads Metadata in JSON format. |

| | |
|---|---|
| ModifyMetadata | Changes an existing metadata key and value. Throws an exception if it does not exist. |
| operator= | Assignment operator. |
| Save | Saves the EZMap8 object to the given Open eVision file. |
| SaveImage | Saves the EZMap8 image to the given image file. |
| SaveImageAndMetadata | Saves image format and Metadata JSON format. |
| SaveMetadata | Saves Metadata in JSON format. |
| Serialize | Serializes the EZMap8 object with all its attributes. |
| SerializeImage | Serializes the image associated to EZMap8. |
| SetBufferPtr | Sets the pointer to an externally allocated image buffer. |
| SetHeight | Access ZMap Height. |
| SetPixel | Sets the value of a pixel . |

| | |
|---|---|
| SetResolution | Sets the resolution. For the X and Y axes, the resolution is expressed in metric units per pixel.<br>For the Z axis it is expressed in metric units per grey scale value. |
| SetSize | Sets the width and height of the EZMap8. |
| SetWidth | Access ZMap Width. |
| SetXResolution | Resolution of the EZMap8 along the X axis<br>The resolution is the number of metric units per pixel. |
| SetYResolution | Resolution of the EZMap8 along the Y axis<br>The resolution is the number of metric units per pixel. |
| SetZResolution | Resolution of the EZMap8 along the Z axis<br>The resolution is the number of grey values per pixel. |
| SetZValue | Sets Z value (in metric coordinate) at pixel coordinates. |
| WorldToImage | Transforms a 3D world position to a floating point (sub)pixel image position.<br>The image space origin is at the upper left corner of the image and the X/Y unit size is one pixel.<br>The Z value is given in grey scale value.<br>Returns TRUE if the pixel position is inside the image limits and the Z value is positive.<br>The parameter pixelPt is filled even if the point is outside the image. |

| | |
|---|---|
| WorldToZMap | Transforms a 3D world position to a 3D EZMap8 position.<br>The ZMap space origin is at the lower left corner of the image.<br>The scales of the world space and ZMap space are the same. |
| ZMapToImage | Converts a 2D coordinate in the EZMap8 space to image (sub)pixel space.<br>(x,y) is the ZMap position (which has the same scale as the world space).<br>(u,v) is the corresponding pixel position (with its origin in the upper left corner of the image).<br>All values are expressed in floating point numbers.<br>Returns TRUE if the pixel position is inside the image limits. |
| ZMapToWorld | Transforms a 3D EZMap8 position to a 3D world space position.<br>The ZMap space origin is at the lower left corner of the image. |

E

Z

# Map8::AddMetadata

Adds a metadata key (name) and value.
If the metadata key already exists, its value will be overwritten.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void AddMetadata(
 const std::string& Key,
 const std::string& value
 )
```

## Parameters

*Key*

The name of the metadata. Names are unique.

*value*

The value for the given metadata.

# EZMap8::AsEImage

Returns the EZMap8 as an EImageBW8 (8 bits gray scale) to use with existing eVision 2D tools.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
EImageBW8& AsEImage(
 )

const EImageBW8& AsEImage(
 )
```

# EZMap8::Clear

Clears the ZMap: replaces all pixels with the undefined value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Clear(
 )
```

# EZMap8::ClearMetadata

Deletes all metadata.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ClearMetadata(
 )
```

# EZMap8::ConvertCoordinatesMapToPixel

Converts 3D Map coordinates to Buffer coordinates

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool ConvertCoordinatesMapToPixel(
 float x3D,
 float y3D,
 int& xBuffer,
 int& yBuffer
 )
```

## Parameters

*x3D*

The Map X coordinate.

*y3D*

The Map Y coordinate.

*xBuffer*

The returned Pixel X coordinate.

*yBuffer*

The returned Pixel Y coordinate.

# EZMap8::ConvertCoordinatesPixelToMap

Converts Buffer coordinates to 3D Map coordinates

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ConvertCoordinatesPixelToMap(
 int xBuffer,
 int yBuffer,
 float& x3D,
 float& y3D
 )
```

## Parameters

*xBuffer*

The pixel X coordinate.

*yBuffer*

The pixel Y coordinate.

*x3D*

The returned Map X coordinate.

*y3D*

The returned Map Y coordinate.

# EZMap8::CopyMetadataTo

Copies all metadata.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void CopyMetadataTo(
 EZMap8& other
 )
```

### Parameters

*other*

An other EZMap8.

# EZMap8::DeleteMetadata

Deletes value of this existing metadata key . Throws an exception if it does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void DeleteMetadata(
 const std::string& Key
 )
```

### Parameters

*Key*

-

# EZMap8::Draw

Draws a EZMap8 in a device context.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )
```

```
void Draw(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void Draw(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

```
void Draw(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY
 )

void Draw(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the ZMap is displayed in 1:1 scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Pan offset (in pixels) in the horizontal direction. By default, no panning is applied.

*panY*

Pan offset (in pixels) in the vertical direction. By default, no panning is applied.

*colorUndefinedPixel*

An optional parameter to choose the drawing color of undefined pixels.

*c24Vector*

When supplied, this parameter allows using a LUT that maps from Depth to C24 when drawing (false colors).

*bw8Vector*

When supplied, this parameter allows using a LUT that maps from Depth to BW8 when drawing.

### Remarks

A ZMap can be drawn (its pixels rendered) using a device context.
The horizontal and vertical zooming factors can be different but must be in the **1/16..16** range.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EZMap8::DrawImage

Displays the internal image buffer

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void DrawImage(
 EDrawAdapter* graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 EDrawAdapter* graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 EDrawAdapter* graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

```
void DrawImage(
 HDC graphicContext,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 HDC graphicContext,
 EC24Vector* c24Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )

void DrawImage(
 HDC graphicContext,
 EBW8Vector* bw8Vector,
 float zoomX,
 float zoomY,
 float panX,
 float panY,
 EC24 colorUndefinedPixel
 )
```

## Parameters

*graphicContext*

Handle to the device context of the destination window.

*zoomX*

Magnification factor for zooming in or out in the horizontal direction. By default, the image is displayed in 1:1 scale.

*zoomY*

Magnification factor for zooming in or out in the vertical direction. Setting a **0** value (which is the default) will result in isotropic scaling (i.e. equal horizontal and vertical factors).

*panX*

Pan offset (in pixels) in the horizontal direction. By default, no panning is applied.

*panY*

Pan offset (in pixels) in the vertical direction. By default, no panning is applied.

*colorUndefinedPixel*

An optional parameter to choose the drawing color of undefined pixels.

*c24Vector*

When supplied, this parameter allows using a LUT that maps from Depth to C24 when drawing (false colors).

*bw8Vector*

When supplied, this parameter allows using a LUT that maps from Depth to BW8 when drawing.

### Remarks

An image can be drawn (its pixels rendered) using a device context.
The horizontal and vertical zooming factors can be different but must be in the **1/16..16** range.
(MFC users can use the **CDC::GetSafeHdc()** method to obtain a suitable device context handle from a **CDC** instance.)

# EZMap8::EZMap8

Creates a 8 bits EZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EZMap8(
 )
void EZMap8(
 OEV_INT32 width,
 OEV_INT32 height
 )
void EZMap8(
 const EZMap8& other
 )
```

### Parameters

*width*

The width of the new Zmap.

*height*

The height of the new Zmap.

*other*

Another Zmap.

# EZMap8::FillUndefinedPixels

Fills undefined pixels, used to fill the "holes" in the ZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void FillUndefinedPixels(
 EZMap8& outMap,
 Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsDirection direction,
 Euresys::Open_eVision_2_10::Easy3D::EFillUndefinedPixelsMethod method
 )
```

### Parameters

*outMap*

The destination ZMap.

*direction*

Direction in which the undefined pixels are filled in a ZMap from EFillUndefinedPixelsDirection.

*method*

Which values used to fill the undefined pixels in a ZMap from EFillUndefinedPixelsMethod.

# EZMap8::GetBufferPtr

Retrieves the pointer to the internal pixel buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void* GetBufferPtr(
 )
```

```
void* GetBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )

const void* GetBufferPtr(
 )

const void* GetBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel which we want the address.

*y*

Row of the pixel which we want the address.

### Remarks

This function does not check the value of the parameters.
Use carefully.

# EZMap8::GetCheckedBufferPtr

Retrieves the pointer to the pixel buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void* GetCheckedBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

```
const void* GetCheckedBufferPtr(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel of which we want the address.

*y*

Row of the pixel of which we want the address.

# EZMap8::GetMetadata

Returns the string value of the given metadata.
Throws an exception if it does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
std::string GetMetadata(
 const std::string& Key
 )
```

### Parameters

*Key*

The name of an exising metadata.

# EZMap8::GetPixel

Gets the value of a pixel .

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
EDepth8 GetPixel(
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*x*

Column of the pixel.

*y*

Row of the pixel.

# EZMap8::GetPixelPositionFromWorldPosition

Returns in the u, v and value parameters the EZMap8 values corresponding to a 3D world position.
The world position is projected on the ZMap reference plane to get a position in the ZMap.
If the projected position is outside the ZMap, the method returns FALSE.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
bool GetPixelPositionFromWorldPosition(
 const E3DPoint& world_position,
 OEV_INT32& u,
 OEV_INT32& v,
 EDepth8& value
 )
```

### Parameters

*world_position*

The 3D coordinates of a world position.

*u*

Column of the ZMap pixel in [0,width[.

*v*

Row of the ZMap pixel in [0,height[.

*value*

Value of the pixel.

# EZMap8::GetResolution

Gets the resolution. For the X and Y axes, the resolution is expressed in metric units per pixel.
For the Z axis it is expressed in metric units per grey scale value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void GetResolution(
 float& sx,
 float& sy,
 float& sz
 )

E3DPoint GetResolution(
 )
```

## Parameters

*sx*

-

*sy*

-

*sz*

-

# EZMap8::GetSizeInWorld

Returns the dimensions of the EZMap8 in real world space (e.g metric unit).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void GetSizeInWorld(
 float& worldWidth,
 float& worldHeight
 )
```

### Parameters

*worldWidth*

Contains the size of the ZMap along the X axis (column).

*worldHeight*

Contains the size of the ZMap along the Y axis (row).

# EZMap8::GetWorldPositionFromPixelPosition

Returns the 3D world position corresponding to a EZMap8 pixel position. The world position is in the original point cloud space. The pixel space origin is at the upper left corner of the image and ranges to (width-1, height-1). The transformation to the world position is calculated using the center of the pixel. The pixel value at position (u,v) must not be undefined, otherwise an exception will be thrown.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DPoint GetWorldPositionFromPixelPosition(
 OEV_INT32 u,
 OEV_INT32 v
 )
```

### Parameters

*u*

Column of the pixel (bounds: [0,width[).

*v*

Row of the pixel (bounds: [0,height[).

# EZMap8::GetZMapPositionFromPixelPosition

Returns the corresponding EZMap8 3D position of a ZMap pixel. The ZMap position is in the original point ZMap space. The pixel position origin is the upper left corner of the image and ranges to (width-1, height-1). The center of the pixel is used for the transformation to the ZMap position. Pixel at position (u,v) must be defined, otherwise an exception will be thrown.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
E3DPoint GetZMapPositionFromPixelPosition(
 OEV_INT32 u,
 OEV_INT32 v
 )
```

### Parameters

*u*

Column of the pixel (bounds: [0,width[).

*v*

Row of the pixel (bounds: [0,height[).

# EZMap8::GetZValue

Gets Z value (in metric coordinate) at pixel coordinates.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetZValue(
 const OEV_INT32 x,
 const OEV_INT32 y
 )
```

### Parameters

*x*

X Coordinate.

*y*

Y Coordinate.

# EZMap8::GetHeight

# EZMap8::SetHeight

Access ZMap Height.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetHeight() const
void SetHeight(OEV_INT32 height)
```

# EZMap8::ImageToWorld

Transforms a floating point (sub)pixel image position to a 3D world position.
The image space origin is at the upper left corner of the image and the X/Y unit size is one pixel.
The pixel Z value is in grey scale values (its range depends on the ZMap type).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ImageToWorld(
 const E3DPoint& pixelPt,
 E3DPoint& worldPt
 )
```

### Parameters

*pixelPt*

Position in the image space.

*worldPt*

Position in the 3D world space.

# EZMap8::ImageToZMap

Converts a 2D image (sub)pixel coordinate to the EZMap8 space.
(u,v) is the pixel position (with its origin in the upper left corner of the image).
(x,y) is the corresponding ZMap position (which has the same scale as the world space).
All values are expressed in floating point numbers.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ImageToZMap(
 float u,
 float v,
 float& x,
 float& y
 )
```

### Parameters

*u*

X Coordinate of the pixel as a floating point value.

*v*

Y Coordinate of the pixel as a floating point value.

*x*

Position along horizontal axis in the ZMap space.

*y*

Position along vertical axis in the ZMap space.

# EZMap8::IsVoid

Tests if the EZMap8 object size is zero.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool IsVoid(
 )
```

## Remarks

Returns **TRUE** if the ZMap size is zero.

# EZMap8::Load

Restores the EZMap8 stored in the given Open eVision file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Load(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

### Remarks

When loading, the ZMap is resized if needed. This function restores all the ZMap attributes.

# EZMap8::LoadImage

Restores the EZMap8 image stored in the given image file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void LoadImage(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

### Remarks

When loading, the ZMap is resized if needed. This function does not restore the ZMap attributes, only the image associated with the EZMap8 is updated.

# EZMap8::LoadImageAndMetadata

Loads image format and Metadata in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void LoadImageAndMetadata(
 const std::string& pathImage,
 const std::string& pathMetadata
 )
```

### Parameters

*pathImage*

Full path to the file.

*pathMetadata*

Full path to the file.

# EZMap8::LoadMetadata

Loads Metadata in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void LoadMetadata(
 const std::string& path
 )
```

### Parameters

*path*

Full path to the file.

# EZMap8::GetMapToWorldMatrix

Returns an E3DTransformMatrix that transforms positions from the EZMap8 space to the world space. This transformation is composed of rotation and translation only, so it is a rigid transformation and preserves distances.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const E3DTransformMatrix& GetMapToWorldMatrix() const
```

# EZMap8::ModifyMetadata

Changes an existing metadata key and value. Throws an exception if it does not exist.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ModifyMetadata(
 const std::string& Key,
 const std::string& value
 )
```

## Parameters

*Key*

-

*value*

-

# EZMap8::operator=

Assignment operator.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EZMap8& operator=(
 const EZMap8& other
 )
```

### Parameters

*other*

The source EZMap8.

# EZMap8::GetRowPitch

Returns the buffer row pitch.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetRowPitch() const
```

# EZMap8::Save

Saves the EZMap8 object to the given Open eVision file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Save(
 const std::string& path
 )
```

### Parameters

*path*

The full path to the destination file.

### Remarks

This format save the EZMap8 in a Open eVision file. This function stores all the ZMap attributes.

# EZMap8::SaveImage

Saves the EZMap8 image to the given image file.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SaveImage(
 const std::string& path,
 Euresys::Open_eVision_2_10::EImageFileType type
 )
```

### Parameters

*path*

The full path to the destination file.

*type*

File format, as defined by EImageFileType. If not specified, the file format is determined from the file extension.

### Remarks

This format save the image associated to EZMap8 in a standard image file and thus does not store ZMap attributes.

# EZMap8::SaveImageAndMetadata

Saves image format and Metadata JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SaveImageAndMetadata(
 const std::string& pathImage,
 const std::string& pathMetadata,
 Euresys::Open_eVision_2_10::EImageFileType type
 )
```

### Parameters

*pathImage*

The full path to the destination file.

*pathMetadata*

The full path to the destination file.

*type*

File format, as defined by EImageFileType. If not specified, the file format is determined from the file extension.

# EZMap8::SaveMetadata

Saves Metadata in JSON format.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void SaveMetadata(
 const std::string& path
 )
```

### Parameters

*path*

The full path to the destination file.

# EZMap8::Serialize

Serializes the EZMap8 object with all its attributes.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or writen to.

# EZMap8::SerializeImage

Serializes the image associated to EZMap8.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SerializeImage(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or writen to.

# EZMap8::SetBufferPtr

Sets the pointer to an externally allocated image buffer.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetBufferPtr(
 OEV_INT32 width,
 OEV_INT32 height,
 void* imagePointer,
 OEV_INT32 bitsPerRow
 )
```

### Parameters

*width*

The width of the supplied buffer, in pixels.

*height*

The height of the supplied buffer, in pixels.

*imagePointer*

The pointer (aligned on 4 bytes) to the buffer, which must be large enough to hold the data.

*bitsPerRow*

The total number of bits contained in a row, padding included.
Using the value **0** (default) means that this size is computed from the buffer width and the pixel size plus a padding with the smallest possible value that leads to a multiple of 4 bytes (32 bits), which is the minimum padding accepted by EZMap8::SetBufferPtr.

# EZMap8::SetPixel

Sets the value of a pixel .

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetPixel(
 EDepth8 value,
 OEV_INT32 x,
 OEV_INT32 y
 )
```

### Parameters

*value*

Value of the pixel.

*x*

Column of the pixel.

*y*

Row of the pixel.

# EZMap8::SetResolution

Sets the resolution. For the X and Y axes, the resolution is expressed in metric units per pixel.
For the Z axis it is expressed in metric units per grey scale value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetResolution(
 float rx,
 float ry,
 float rz
 )
void SetResolution(
 E3DPoint resolution
 )
```

### Parameters

*rx*

-

*ry*

-

*rz*

-

*resolution*

-

# EZMap8::SetSize

Sets the width and height of the EZMap8.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetSize(
 OEV_INT32 width,
 OEV_INT32 height
 )
void SetSize(
 const EZMap& other
 )
```

## Parameters

*width*

The new requested width.

*height*

The new requested height.

*other*

The other ZMap whose dimensions have to be used for the current object.

### Remarks

Open eVision will allocate a new image buffer (deallocate the old image buffer) if the supplied width and height are different from the existing ones.
If an external buffer has been specified by means of **SetImagePtr**, it will be kept only if the size does not change.
Creating a new Open eVision image buffer and setting its size creates a 4-byte aligned buffer, by default. The *size of an ZMap* is specified as a number of columns (width) and rows (height).
The maximum image dimensions are 32767 by 32767. Furthermore, it must fit into the available memory, that depends upon the physical memory, the operating system and the memory already allocated by the process in other modules or libraries.

# EZMap8::SetZValue

Sets Z value (in metric coordinate) at pixel coordinates.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void SetZValue(
 const float value,
 const OEV_INT32 x,
 const OEV_INT32 y
 )
```

### Parameters

*value*

Value of the pixel in metric space.

*x*

X Coordinate.

*y*

Y Coordinate.

# EZMap8::GetType

Pixel accessor type.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
Euresys::Open_eVision_2_10::EImageType GetType() const
```

# EZMap8::GetUndefinedValue

Returns the Undefined value. That value is used to set to mark pixels with no valid depth value.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EDepth8 GetUndefinedValue() const
```

# EZMap8::GetWidth

# EZMap8::SetWidth

Access ZMap Width.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
OEV_INT32 GetWidth() const
void SetWidth(OEV_INT32 width)
```

# EZMap8::GetWorldShape

Returns the EWorldShape for conversion between 2D image and 2D world space coordinates. This EWorldShape can be used by EasyGauge to do measurements on a EZMap8 in real space coordinates (e.g mm).

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const EWorldShape& GetWorldShape() const
```

# EZMap8::WorldToImage

Transforms a 3D world position to a floating point (sub)pixel image position.
The image space origin is at the upper left corner of the image and the X/Y unit size is one pixel.
The Z value is given in grey scale value.
Returns TRUE if the pixel position is inside the image limits and the Z value is positive.
The parameter pixelPt is filled even if the point is outside the image.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool WorldToImage(
 const E3DPoint& worldPt,
 E3DPoint& pixelPt
 )
```

## Parameters

*worldPt*

Position in the 3D world space.

*pixelPt*

Position in the image space.

# EZMap8::GetWorldToMapMatrix

Returns an E3DTransformMatrix that transforms positions from the world space to the EZMap8 space. This transformation is composed of rotation and translation only, so it is a rigid transformation and it preserves distances.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
const E3DTransformMatrix& GetWorldToMapMatrix() const
```

# EZMap8::WorldToZMap

Transforms a 3D world position to a 3D EZMap8 position.
The ZMap space origin is at the lower left corner of the image.
The scales of the world space and ZMap space are the same.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void WorldToZMap(
 const E3DPoint& worldPt,
 E3DPoint& zmapPt
 )
```

## Parameters

*worldPt*

Position in the 3D world space.

*zmapPt*

Position in the ZMap space.

### Remarks

Do not use this method with the same variable as input and output. It might lead to incorrect results.

# EZMap8::GetXResolution

# EZMap8::SetXResolution

Resolution of the EZMap8 along the X axis
The resolution is the number of metric units per pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetXResolution() const
void SetXResolution(float resolution)
```

# EZMap8::GetYResolution

# EZMap8::SetYResolution

Resolution of the EZMap8 along the Y axis
The resolution is the number of metric units per pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
float GetYResolution() const
void SetYResolution(float resolution)
```

# EZMap8::ZMapToImage

Converts a 2D coordinate in the EZMap8 space to image (sub)pixel space.
(x,y) is the ZMap position (which has the same scale as the world space).
(u,v) is the corresponding pixel position (with its origin in the upper left corner of the image).
All values are expressed in floating point numbers.
Returns TRUE if the pixel position is inside the image limits.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool ZMapToImage(
 float x,
 float y,
 float& u,
 float& v
 )
```

### Parameters

*x*

Position along horizontal axis in the ZMap space.

*y*

Position along vertical axis in the ZMap space.

*u*

Column of the pixel as a floating point value.

*v*

Row of the pixel as a floating point value.

# EZMap8::ZMapToWorld

Transforms a 3D EZMap8 position to a 3D world space position.
The ZMap space origin is at the lower left corner of the image.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ZMapToWorld(
 const E3DPoint& zmapPt,
 E3DPoint& worldPt
 )
```

### Parameters

*zmapPt*

Position in the ZMap space.

*worldPt*

Position in the 3D world space.

### Remarks

Do not use this method with the same variable as input and output. It might lead to incorrect results.

# EZMap8::GetZResolution

# EZMap8::SetZResolution

Resolution of the EZMap8 along the Z axis
The resolution is the number of grey values per pixel.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float GetZResolution() const
void SetZResolution(float resolution)
```

# 4.198. EZMapToPointCloudConverter Class

Generates an EPointCloud from a ZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

## Methods

| | |
|---|---|
| Convert | Generates an EPointCloud from a ZMap (EZMap8, EZMap16 or EZMap32f). ZMap pixel positions/values (U,V,W) are converted to 3D positions (X,Y,Z) and written to the given point cloud. By default, the target coordinate system is the original world space. Optional parameter inZMapSpace can switch to the ZMap space as the target coordinate system. |
| EZMapToPointCloudConverter | Creates an EZMapToPointCloudConverter object. |

E

Z

# MapToPointCloudConverter::Convert

Generates an EPointCloud from a ZMap (EZMap8, EZMap16 or EZMap32f). ZMap pixel positions/values (U,V,W) are converted to 3D positions (X,Y,Z) and written to the given point cloud. By default, the target coordinate system is the original world space. Optional parameter inZMapSpace can switch to the ZMap space as the target coordinate system.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

[C++]

```
void Convert(
 const EZMap8& srcZMap,
 EPointCloud& pointCloud,
 bool inZMapSpace
 )

void Convert(
 const EZMap8& srcZMap,
 ERegion& region,
 EPointCloud& pointCloud,
 bool inZMapSpace
 )

void Convert(
 const EZMap16& srcZMap,
 EPointCloud& pointCloud,
 bool inZMapSpace
 )

void Convert(
 const EZMap16& srcZMap,
 ERegion& region,
 EPointCloud& pointCloud,
 bool inZMapSpace
 )

void Convert(
 const EZMap32f& srcZMap,
 EPointCloud& pointCloud,
 bool inZMapSpace
 )

void Convert(
 const EZMap32f& srcZMap,
 ERegion& region,
 EPointCloud& pointCloud,
 bool inZMapSpace
 )

void Convert(
 const EZMap* srcZMap,
 EPointCloud& pointCloud,
 bool inZMapSpace
 )
```

```
void Convert(
 const EZMap* srcZMap,
 ERegion& region,
 EPointCloud& pointCloud,
 bool inZMapSpace
 )
```

### Parameters

*srcZMap*

The ZMap to convert.

*pointCloud*

The destination point cloud.

*inZMapSpace*

When TRUE, converts to 3D ZMap space instead of world space (default is FALSE).

*region*

The region of interest (default is no region).

### Remarks

The destination point cloud will be cleared before being (re-)populated.

# EZMapToPointCloudConverter::EZMapToPointCloudConverter

Creates an EZMapToPointCloudConverter object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void EZMapToPointCloudConverter(
 )

void EZMapToPointCloudConverter(
 const EZMapToPointCloudConverter& other
 )
```

## Parameters

*other*

-

# 5. Structures

## 5.1. E3DPoint Struct

Represents a 3D point with floating point coordinates.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

### Properties

| | |
|---|---|
| X | X coordinate of the point. |
| Y | Y coordinate of the point. |
| Z | Z coordinate of the point. |

### Methods

| | |
|---|---|
| DistanceTo | Returns the euclidean distance to another E3DPoint. |
| E3DPoint | Constructs a default E3DPoint object. |
| operator!= | Checks if two E3DPoint are stricly different. |
| operator== | Checks if two E3DPoint are stricly equal. |
| Serialize | Serializes the E3DPoint. |

# E3DPoint::DistanceTo

Returns the euclidean distance to another E3DPoint.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float DistanceTo(
 E3DPoint point
 )
```

## Parameters

*point*

The other point.

# E3DPoint::E3DPoint

Constructs a default E3DPoint object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void E3DPoint(
 )

void E3DPoint(
 float x,
 float y,
 float z
 )
```

## Parameters

*x*

X coordinate of the point.

*y*

Y coordinate of the point.

*z*

Z coordinate of the point.

### Remarks

If the default constructor is used, the point is initialized to (0, 0, 0).

# E3DPoint::operator!=

Checks if two E3DPoint are stricly different.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool operator!=(
 E3DPoint point
 )
```

### Parameters

*point*

The other point.

# E3DPoint::operator==

Checks if two E3DPoint are stricly equal.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool operator==(
 E3DPoint point
 )
```

### Parameters

*point*

The other point.

# E3DPoint::Serialize

Serializes the E3DPoint.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void Serialize(
 ESerializer* serializer
 )
```

### Parameters

*serializer*

The ESerializer object that is read from or written to.

# E3DPoint::X

X coordinate of the point.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float X
```

## E3DPoint::Y

Y coordinate of the point.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float Y
```

## E3DPoint::Z

Z coordinate of the point.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
float Z
```

# 5.2. EBW1 Struct

Black and white pixel value, coded as an unsigned 32-bit integer.

### Remarks

Every pixel is coded on 1 bit, allowing to represent 2 different values. The value **0** stands for black (background), and the value **1** stands for white (foreground).

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| Value | The value of the pixel. |

## Methods

| | |
|---|---|
| EBW1 | Constructs a EBW1 object. |
| GetSize | The number of bits in a pixel |

# EBW1::EBW1

Constructs a EBW1 object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EBW1(
 )
void EBW1(
 OEV_UINT32 value
 )
```

### Parameters

*value*

The value of the pixel.

## EBW1::GetSize

The number of bits in a pixel

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetSize()
```

## EBW1::Value

The value of the pixel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 Value
```

# 5.3. EBW16 Struct

Gray-level pixel value, coded as an unsigned 16-bit integer.

### Remarks

High-quality cameras or scanners are able to digitize on 10 or 12 bits. Sometimes too, to avoid numerical truncation errors, intermediate processing results require more than 8 bits of storage. In such situations, 8 bits gray-level images are no longer sufficient. 16 bits gray-level images are similar to 8 bits ones, but each pixel is, in this case, coded on 16 bits, which effect is to increase the levels of gray to 65,536. It is not possible to show the difference between a gray-level image quantized on 16 bits rather than 8. Under Windows, no display device is able to display 16-bit gray levels. Windows doesn't allow you to display more than 256 gray levels.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| Value | The value of the pixel. |

## Methods

| | |
|---|---|
| EBW16 | Constructs a EBW16 object. |
| GetSize | The number of bits in a pixel |

# EBW16::EBW16

Constructs a EBW16 object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EBW16(
 )
void EBW16(
 OEV_UINT16 value
 )
```

### Parameters

*value*

The value of the pixel.

## EBW16::GetSize

The number of bits in a pixel

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetSize()
```

## EBW16::Value

The value of the pixel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT16 Value
```

# 5.4. EBW16Path Struct

Path from a EBW16 image: image pixel coordinates, and associated gray-level pixel value.

**Namespace:** Euresys::Open_eVision_2_10

### Properties

| | |
|---|---|
| Pixel | The value of the pixel at this coordinate. |
| X | Coordinate along the horizontal direction. |
| Y | Coordinate along the vertical direction. |

# EBW16Path::Pixel

The value of the pixel at this coordinate.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW16 Pixel
```

# EBW16Path::X

Coordinate along the horizontal direction.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 X
```

# EBW16Path::Y

Coordinate along the vertical direction.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 Y
```

# 5.5. EBW32 Struct

Gray-level pixel value, coded as an unsigned 32-bit integer.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| Value | The value of the pixel. |

## Methods

| | |
|---|---|
| EBW32 | Constructs a EBW32 object. |
| GetSize | The number of bits in a pixel |

# EBW32::EBW32

Constructs a EBW32 object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EBW32(
 )
void EBW32(
 OEV_UINT32 value
 )
```

### Parameters

*value*

The value of the pixel.

# EBW32::GetSize

The number of bits in a pixel

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 GetSize()
```

# EBW32::Value

The value of the pixel.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT32 Value
```

# 5.6. EBW8 Struct

Gray-level pixel value, coded as an unsigned 8-bit integer.

### Remarks

Every pixel is coded on 8 bits, allowing to represent 256 different values. The value **0** stands for black (background) and the value **255** stands for white (foreground). The 254 remaining values stand for shades of gray. This is sufficient for most applications. Most of the Open eVision gray-level operations apply to this pixel type.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| Value | The value of the pixel. |

## Methods

| | |
|---|---|
| EBW8 | Constructs a EBW8 object. |
| GetSize | The number of bits in a pixel |

# EBW8::EBW8

Constructs a EBW8 object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EBW8(
 )
```

```
void EBW8(
 OEV_UINT8 value
 )
```

### Parameters

*value*

The value of the pixel.

# EBW8::GetSize

The number of bits in a pixel

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 GetSize()
```

# EBW8::Value

The value of the pixel.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT8 Value
```

# 5.7. EBW8Path Struct

Path from a EBW8 image: image pixel coordinates, and associated gray-level pixel value.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| Pixel | The value of the pixel at this coordinate. |
| X | Coordinate along the horizontal direction. |
| Y | Coordinate along the vertical direction. |

## EBW8Path::Pixel

The value of the pixel at this coordinate.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EBW8 Pixel
```

## EBW8Path::X

Coordinate along the horizontal direction.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 X
```

## EBW8Path::Y

Coordinate along the vertical direction.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 Y
```

# 5.8. EC15 Struct

Color pixel value, coded as 3 fields of 5 bits each (red, green, blue components) and 1 field of 1 bit for padding.

### Remarks

This class is suited to handle the Windows RGB15 color images. The pixel values are coded on 15 bits, leaving 32 possible levels per color component (red, green or blue).

**Namespace:** Euresys::Open_eVision_2_10

### Properties

| | |
|---|---|
| C0 | The value of the first component of the pixel (red channel). |
| C1 | The value of the second component of the pixel (green channel). |
| C2 | The value of the third component of the pixel (blue channel). |

## Methods

| | |
|---|---|
| EC15 | Constructs a EC15 object. |
| GetSize | The number of bits in a pixel |

# EC15::C0

The value of the first component of the pixel (red channel).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT16 C0
```

# EC15::C1

The value of the second component of the pixel (green channel).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT16 C1
```

# EC15::C2

The value of the third component of the pixel (blue channel).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT16 C2
```

# EC15::EC15

Constructs a EC15 object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EC15(
 )

void EC15(
 OEV_UINT8 c0,
 OEV_UINT8 c1,
 OEV_UINT8 c2
 )
```

## Parameters

*c0*

The value of the first component of the pixel (red channel).

*c1*

The value of the second component of the pixel (green channel).

*c2*

The value of the third component of the pixel (blue channel).

## EC15::GetSize

The number of bits in a pixel

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetSize()
```

# 5.9. EC16 Struct

Color pixel value, coded as 3 fields of 5 bits, 6 bits and 5 bits (red, green and blue components).

### Remarks

This class is suited to handle the Windows RGB16 color images. The pixel values are coded on 16 bits (5-6-5), leaving 32 possible levels for R and B components, and 64 possible levels for G component.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| C0 | The value of the first component of the pixel (red channel). |
| C1 | The value of the second component of the pixel (green channel). |
| C2 | The value of the third component of the pixel (blue channel). |

## Methods

| | |
|---|---|
| EC16 | Constructs a EC16 object. |
| | |

| | |
|---|---|
| GetSize | The number of bits in a pixel |

E

C

# 16::C0

The value of the first component of the pixel (red channel).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT16 C0
```

# EC16::C1

The value of the second component of the pixel (green channel).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT16 C1
```

# EC16::C2

The value of the third component of the pixel (blue channel).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT16 C2
```

# EC16::EC16

Constructs a EC16 object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EC16(
 )
void EC16(
 OEV_UINT8 c0,
 OEV_UINT8 c1,
 OEV_UINT8 c2
 )
```

### Parameters

*c0*

The value of the first component of the pixel (red channel).

*c1*

The value of the second component of the pixel (green channel).

*c2*

The value of the third component of the pixel (blue channel).

# EC16::GetSize

The number of bits in a pixel

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetSize()
```

# 5.10. EC24 Struct

Color pixel value coded as 3 unsigned 8-bit integers (red, green and blue components).

### Remarks

(RGB triplet, windows 24 bpp bitmap format) The pixel values are coded on 24 bits, providing 256 possible levels per color component. This way, RGB images can represent 16,777,216 different colors. This is sufficient for most applications. Most of the Open eVision color operations apply to this pixel type.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| C0 | The value of the first component of the pixel (red channel). |
| C1 | The value of the second component of the pixel (green channel). |
| C2 | The value of the third component of the pixel (blue channel). |

## Methods

| | |
|---|---|
| EC24 | Constructs a EC24 object. |
| GetSize | The number of bits in a pixel |

# EC24::C0

The value of the first component of the pixel (red channel).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT8 C0
```

# EC24::C1

The value of the second component of the pixel (green channel).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT8 C1
```

# EC24::C2

The value of the third component of the pixel (blue channel).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
OEV_UINT8 C2
```

# EC24::EC24

Constructs a EC24 object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EC24(
 )
void EC24(
 const ERGBColor& rgbColor
 )
void EC24(
 OEV_UINT8 c0,
 OEV_UINT8 c1,
 OEV_UINT8 c2
 )
```

## Parameters

*rgbColor*

-

*c0*

The value of the first component of the pixel (red channel).

*c1*

The value of the second component of the pixel (green channel).

*c2*

The value of the third component of the pixel (blue channel).

## EC24::GetSize

The number of bits in a pixel

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetSize()
```

# 5.11. EC24A Struct

Color pixel value coded as 4 unsigned 8-bit integers (red, green, blue and alpha components).

### Remarks

This class is suited to handle the Windows RGB32 color format. The pixel values are coded on 32 bits, leaving 256 possible levels per color component (red, green or blue), and 8 more bits for an alpha channel. Currently, the alpha channel is not used for any purpose in the Open eVision processing functions. Users are free to use it to store an additional gray-level content.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| A | The value of the alpha component of the pixel. |
| C0 | The value of the first component of the pixel (red channel). |
| C1 | The value of the second component of the pixel (green channel). |
| C2 | The value of the third component of the pixel (blue channel). |

## Methods

| | |
|---|---|
| EC24A | Constructs a EC24A object. |
| GetSize | The number of bits in a pixel |

# EC24A::A

The value of the alpha component of the pixel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT8 A
```

# EC24A::C0

The value of the first component of the pixel (red channel).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT8 C0
```

# EC24A::C1

The value of the second component of the pixel (green channel).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT8 C1
```

# EC24A::C2

The value of the third component of the pixel (blue channel).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT8 C2
```

# EC24A::EC24A

Constructs a EC24A object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EC24A(
 )
```

```
void EC24A(
 OEV_UINT8 c0,
 OEV_UINT8 c1,
 OEV_UINT8 c2,
 OEV_UINT8 a
 )
```

### Parameters

*c0*

The value of the first component of the pixel (red channel).

*c1*

The value of the second component of the pixel (green channel).

*c2*

The value of the third component of the pixel (blue channel).

*a*

-

## EC24A::GetSize

The number of bits in a pixel

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetSize()
```

# 5.12. EC24Path Struct

Path from a EC24 image: image pixel coordinates, and associated color pixel value.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| Pixel | The value of the pixel at this coordinate. |
| X | Coordinate along the horizontal direction. |
| Y | Coordinate along the vertical direction. |

# EC24Path::Pixel

The value of the pixel at this coordinate.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
EC24 Pixel
```

# EC24Path::X

Coordinate along the horizontal direction.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 X
```

# EC24Path::Y

Coordinate along the vertical direction.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 Y
```

# 5.13. EC48 Struct

Color pixel value coded as 3 unsigned 16-bit integers (red, green, blue components).

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| C0 | The value of the first component of the pixel (red channel). |
| C1 | The value of the second component of the pixel (green channel). |
| C2 | The value of the third component of the pixel (blue channel). |

## Methods

| | |
|---|---|
| EC48 | Constructs a EC48 object. |
| GetSize | The size of a pixel. |

# EC48::C0

The value of the first component of the pixel (red channel).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT16 C0
```

# EC48::C1

The value of the second component of the pixel (green channel).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT16 C1
```

# EC48::C2

The value of the third component of the pixel (blue channel).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
OEV_UINT16 C2
```

# EC48::EC48

Constructs a EC48 object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EC48(
 )
void EC48(
 OEV_UINT16 c0,
 OEV_UINT16 c1,
 OEV_UINT16 c2
 )
```

## Parameters

*c0*

The value of the first component of the pixel (red channel).

*c1*

The value of the second component of the pixel (green channel).

*c2*

The value of the third component of the pixel (blue channel).

# EC48::GetSize

The size of a pixel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetSize()
```

# 5.14. EColor Struct

Triple of floating-point numbers that encode a color in a given color system.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| C0 | First color component. |
| C1 | Second color component. |
| C2 | Third color component. |

## Methods

| | |
|---|---|
| EColor | Constructor for EColor objects. |

# EColor::C0

First color component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float C0
```

# EColor::C1

Second color component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float C1
```

# EColor::C2

Third color component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float C2
```

# EColor::EColor

Constructor for EColor objects.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EColor(
 )
void EColor(
 float c0,
 float c1,
 float c2
 )
```

### Parameters

*c0*

value for the first color component

*c1*

value for the second color component

*c2*

value for the third color component

# 5.15. EDepth16 Struct

Depth value of the pixel, coded as an unsigned 16-bit integer.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| Value | The depth value of the pixel. |

## Methods

| | |
|---|---|
| EDepth16 | Constructs a EDepth16 object. |
| GetSize | The number of bits in a pixel |

# EDepth16::EDepth16

Constructs a EDepth16 object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EDepth16(
 )
void EDepth16(
 OEV_UINT16 value
 )
```

### Parameters

*value*

The depth value of the pixel.

# EDepth16::GetSize

The number of bits in a pixel

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetSize()
```

# EDepth16::Value

The depth value of the pixel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT16 Value
```

# 5.16. EDepth32f Struct

Depth value of the pixel, coded as a 32-bits floating point value.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| Value | The depth value of the pixel. |

## Methods

| | |
|---|---|
| EDepth32f | Constructs a EDepth32f object. |
| GetSize | The number of bits in a pixel |

# EDepth32f::EDepth32f

Constructs a EDepth32f object.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EDepth32f(
 )
void EDepth32f(
 float value
 )
```

### Parameters

*value*

The depth value of the pixel.

# EDepth32f::GetSize

The number of bits in a pixel

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 GetSize()
```

# EDepth32f::Value

The depth value of the pixel.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float Value
```

# 5.17. EDepth8 Struct

Depth value of the pixel, coded as an unsigned 8-bit integer.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| Value | The depth value of the pixel. |

## Methods

| | |
|---|---|
| EDepth8 | Constructs a EDepth8 object. |
| GetSize | The number of bits in a pixel |

# EDepth8::EDepth8

Constructs a EDepth8 object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EDepth8(
 )
void EDepth8(
 OEV_UINT8 value
 )
```

### Parameters

*value*

The depth value of the pixel.

## EDepth8::GetSize

The number of bits in a pixel

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GetSize()
```

## EDepth8::Value

The depth value of the pixel.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT8 Value
```

# 5.18. EFeatureData Struct

Describes object features.

### Remarks

A feature is associated to an array of values, each corresponding to an object of given identification number. A feature is also characterized by the size of the array, a feature number, data size/type information and pointers to both ends of the array. The features can be accessed by their number (see EFeature). To obtain the value of a given feature of a given object, just use the class member ECodedImage::GetObjectFeature. This enumeration pertains to the EasyObject legacy API and should not be used for new developments. See ECodedImage2 for the new API.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| FeatDataSize | Data size (see EDataSize). |
| FeatDataType | Data type (see EDataType). |
| FeatNum | Code (see EFeature). |
| Size | Number of objects for which the feature is stored. |

# EFeatureData::FeatDataSize

Data size (see EDataSize).

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
Euresys::Open_eVision_2_10::EDataSize FeatDataSize
```

# EFeatureData::FeatDataType

Data type (see EDataType).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::EDataType FeatDataType
```

## EFeatureData::FeatNum

Code (see EFeature).

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
Euresys::Open_eVision_2_10::ELegacyFeature FeatNum
```

## EFeatureData::Size

Number of objects for which the feature is stored.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 Size
```

# 5.19. EISH Struct

Intensity, Saturation, Hue color system.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| H | Hue component. |
| I | Intensity component. |
| S | Saturation component. |

# EISH::H

Hue component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float H
```

# EISH::I

Intensity component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float I
```

## EISH::S

Saturation component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float S
```

# 5.20. ELAB Struct

CIE Lightness, a*, b* color system.

**Namespace:** Euresys::Open_eVision_2_10

### Properties

| | |
|---|---|
| A | a* component. |
| B | b* component. |
| L | Lightness component. |

## ELAB::A

a* component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float A
```

## ELAB::B

b* component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float B
```

## ELAB::L

Lightness component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float L
```

# 5.21. ELCH Struct

Lightness, Chroma, Hue color system.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| C | Chroma component. |
| H | Hue component. |
| L | Lightness component. |

# ELCH::C

Chroma component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float C
```

# ELCH::H

Hue component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float H
```

## ELCH::L

Lightness component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float L
```

# 5.22. ELSH Struct

Lightness, Saturation, Hue color system.

**Namespace:** Euresys::Open_eVision_2_10

### Properties

| | |
|---|---|
| H | Hue component. |
| L | Lightness component. |
| S | Saturation component. |

## ELSH::H

Hue component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float H
```

## ELSH::L

Lightness component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float L
```

## ELSH::S

Saturation component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float S
```

# 5.23. ELUV Struct

CIE Lightness, u*, v* color system.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| L | Lightness component. |
| U | u* component. |
| V | v* component. |

# ELUV::L

Lightness component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float L
```

# ELUV::U

u* component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float U
```

# ELUV::V

v* component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float V
```

# 5.24. EMatchPosition Struct

Represents a single instance of the pattern in the search field, as returned by the EasyMatch matching process.

### Remarks

EMatcher::GetPosition returns instances of this class. A EMatchPosition object represents one matched instance, with all the needed information about it.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| Angle | Clockwise rotation angle, expressed using the current angle unit, of the pattern found in the image. |
| AreaRatio | Ratio between the found pattern area inside the search ROI and its complete area. |
| CenterX | Abscissa of the center of the pattern found in the image. |
| CenterY | Ordinate of the center of the pattern found in the image. |
| | |

| | |
|---|---|
| Interpolated | Indicates whether sub-pixel interpolation was actually performed on the position. |
| Scale | Scale factor of the pattern found in the image. |
| ScaleX | Measured horizontal scaling of the pattern found in the image, expressed as a dimensionless ratio. |
| ScaleY | Measured vertical scaling of the pattern found in the image, expressed as a dimensionless ratio. |
| Score | Indicates how good a matching was. |

# EMatchPosition::Angle

Clockwise rotation angle, expressed using the current angle unit, of the pattern found in the image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float Angle
```

## Remarks

**0** if no rotation is allowed.

# EMatchPosition::AreaRatio

Ratio between the found pattern area inside the search ROI and its complete area.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float AreaRatio
```

# EMatchPosition::CenterX

Abscissa of the center of the pattern found in the image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float CenterX
```

# EMatchPosition::CenterY

Ordinate of the center of the pattern found in the image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float CenterY
```

# EMatchPosition::Interpolated

Indicates whether sub-pixel interpolation was actually performed on the position.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL Interpolated
```

### Remarks

In some cases, when the pattern is found close to the ROI edge, sub-pixel interpolation cannot be used.

# EMatchPosition::Scale

Scale factor of the pattern found in the image.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float Scale
```

### Remarks

**1** if no scaling is allowed.

# EMatchPosition::ScaleX

Measured horizontal scaling of the pattern found in the image, expressed as a dimensionless ratio.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float ScaleX
```

# EMatchPosition::ScaleY

Measured vertical scaling of the pattern found in the image, expressed as a dimensionless ratio.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float ScaleY
```

# EMatchPosition::Score

Indicates how good a matching was.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float Score
```

## Remarks

**1** means that the matching was perfect. Lower values correspond to approximate matching; **-1** corresponds to a perfect mismatch (pattern superimposed on its negative image).

# 5.25. EMatrixCodeIso15415GradingParameters Struct

Holds all grading parameters pertaining to ISO/IEC 15415

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| AxialNonUniformity | Axial Non Uniformity |
| AxialNonUniformityGrade | Axial Non Uniformity Grade |
| DecodingGrade | Decoding Grade |
| FixedPatternDamageGrade | Fixed Pattern Damage Grade |
| GridNonUniformity | Grid Non Uniformity |
| GridNonUniformityGrade | Grid Non Uniformity Grade |
| HorizontalPrintGrowth | Horizontal Print Growth |
| ModulationGrade | Modulation Grade |
| OverallSymbolGrade | Overall Symbol Grade |
| ReflectanceMarginGrade | Reflectance Margin Grade |
| SymbolContrast | Symbol Contrast |
| SymbolContrastGrade | Symbol Contrast Grade |
| UnusedErrorCorrection | Unused Error Correction |
| UnusedErrorCorrectionGrade | Unused Error Correction Grade |
| VerticalPrintGrowth | Vertical Print Growth |

## Methods

| | |
|---|---|
| EMatrixCodeIso15415GradingParameters | - |

# EMatrixCodeIso15415GradingParameters::AxialNonUniformity

Axial Non Uniformity

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float AxialNonUniformity
```

# EMatrixCodeIso15415GradingParameters::AxialNonUniformityGrade

Axial Non Uniformity Grade

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 AxialNonUniformityGrade
```

# EMatrixCodeIso15415GradingParameters::DecodingGrade

Decoding Grade

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 DecodingGrade
```

# EMatrixCodeIso15415GradingParameters::EMatrixCodeIso15415GradingParameters

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EMatrixCodeIso15415GradingParameters(
 )
```

# EMatrixCodeIso15415GradingParameters::FixedPatternDamageGrade

Fixed Pattern Damage Grade

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 FixedPatternDamageGrade
```

# EMatrixCodeIso15415GradingParameters::GridNonUniformity

Grid Non Uniformity

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float GridNonUniformity
```

# EMatrixCodeIso15415GradingParameters::GridNonUniformityGrade

Grid Non Uniformity Grade

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 GridNonUniformityGrade
```

# EMatrixCodeIso15415GradingParameters::HorizontalPrintGrowth

Horizontal Print Growth

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float HorizontalPrintGrowth
```

# EMatrixCodeIso15415GradingParameters::ModulationGrade

Modulation Grade

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 ModulationGrade
```

# EMatrixCodeIso15415GradingParameters::OverallSymbolGrade

Overall Symbol Grade

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 OverallSymbolGrade
```

# EMatrixCodeIso15415GradingParameters::ReflectanceMarginGrade

Reflectance Margin Grade

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 ReflectanceMarginGrade
```

# EMatrixCodeIso15415GradingParameters::SymbolContrast

Symbol Contrast

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float SymbolContrast
```

# EMatrixCodeIso15415GradingParameters::SymbolContrastGrade

Symbol Contrast Grade

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 SymbolContrastGrade
```

# EMatrixCodeIso15415GradingParameters::UnusedErrorCorrection

Unused Error Correction

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float UnusedErrorCorrection
```

# EMatrixCodeIso15415GradingParameters::UnusedErrorCorrectionGrade

Unused Error Correction Grade

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 UnusedErrorCorrectionGrade
```

# EMatrixCodeIso15415GradingParameters::VerticalPrintGrowth

Vertical Print Growth

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float VerticalPrintGrowth
```

# 5.26. EMatrixCodeIso29158CalibrationParameters Struct

Holds all grading inputs pertaining to ISO/IEC 29158

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| MLcal | Mean of the light from a histogram of the calibrated standard. |
| Rcal | Reported reflectance value, from a calibration standard. |
| SRcal | System response parameters(such as exposure and/or again) used to create an image of the calibration standard. |
| SRtarget | System response parameters(such as exposure and/or again) used to create an image of the symbole under test. |

## Methods

| | |
|---|---|
| EMatrixCodeIso29158CalibrationParameters | - |

# EMatrixCodeIso29158CalibrationParameters::EMatrixCodeIso29158CalibrationParameters

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EMatrixCodeIso29158CalibrationParameters(
 )
```

# EMatrixCodeIso29158CalibrationParameters::MLcal

Mean of the light from a histogram of the calibrated standard.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float MLcal
```

# EMatrixCodeIso29158CalibrationParameters::Rcal

Reported reflectance value, from a calibration standard.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float Rcal
```

# EMatrixCodeIso29158CalibrationParameters::SRcal

System response parameters(such as exposure and/or again) used to create an image of the calibration standard.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float SRcal
```

# EMatrixCodeIso29158CalibrationParameters::SRtarget

System response parameters(such as exposure and/or again) used to create an image of the symbole under test.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float SRtarget
```

# 5.27. EMatrixCodeIso29158GradingParameters Struct

Holds all grading parameters pertaining to ISO/IEC 29158

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| CellContrastGrade | Cell Contrast Grade |
| CellModulationGrade | Cell Modulation Grade |
| FixedPatternDamageGrade | FixedPatternDamageGrade |
| IsMeanLightInRequiredBounds | Is Mean Light In Required Bounds |
| MeanLight | Mean Light |
| MinimumReflectanceGrade | Minimum Reflectance Grade |
| OverallSymbolGrade | Overall Symbol Grade |

## Methods

| | |
|---|---|
| EMatrixCodeIso29158GradingParameters | - |

# EMatrixCodeIso29158GradingParameters::CellContrastGrade

Cell Contrast Grade

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 CellContrastGrade
```

# EMatrixCodeIso29158GradingParameters::CellModulationGrade

Cell Modulation Grade

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 CellModulationGrade
```

# EMatrixCodeIso29158GradingParameters::EMatrixCodeIso29158GradingParameters

-

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EMatrixCodeIso29158GradingParameters(
 )
```

# EMatrixCodeIso29158GradingParameters::FixedPatternDamageGrade

FixedPatternDamageGrade

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 FixedPatternDamageGrade
```

# EMatrixCodeIso29158GradingParameters::IsMeanLightInRequiredBounds

Is Mean Light In Required Bounds

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
BOOL IsMeanLightInRequiredBounds
```

# EMatrixCodeIso29158GradingParameters::MeanLight

Mean Light

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float MeanLight
```

# EMatrixCodeIso29158GradingParameters::MinimumReflectanceGrade

Minimum Reflectance Grade

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 MinimumReflectanceGrade
```

# EMatrixCodeIso29158GradingParameters::OverallSymbolGrade

Overall Symbol Grade

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 OverallSymbolGrade
```

# 5.28. EMatrixCodeSemiT10GradingParameters Struct

Holds all grading parameters pertaining to Semi T10-

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| CellDefects | Cell Defects |
| DataMatrixCellHeight | Data Matrix Cell Height |
| DataMatrixCellWidth | Data Matrix Cell Width |
| FinderPatternDefects | Finder Pattern Defects |
| HorizontalMarkGrowth | Horizontal Mark Growth |
| HorizontalMarkMisplacement | Horizontal Mark Misplacement |
| SymbolContrast | Symbol Contrast |
| SymbolContrastSNR | Symbol Contrast SNR |
| UnusedErrorCorrection | Unused Error Correction |
| VerticalMarkGrowth | Vertical Mark Growth |
| VerticalMarkMisplacement | Vertical Mark Misplacement |

## Methods

| | |
|---|---|
| EMatrixCodeSemiT10GradingParameters | - |

# EMatrixCodeSemiT10GradingParameters::CellDefects

Cell Defects

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float CellDefects
```

# EMatrixCodeSemiT10GradingParameters::DataMatrixCellHeight

Data Matrix Cell Height

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float DataMatrixCellHeight
```

# EMatrixCodeSemiT10GradingParameters::DataMatrixCellWidth

Data Matrix Cell Width

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float DataMatrixCellWidth
```

# EMatrixCodeSemiT10GradingParameters::EMatrixCodeSemiT10GradingParameters

-

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
void EMatrixCodeSemiT10GradingParameters(
 )
```

# EMatrixCodeSemiT10GradingParameters::FinderPatternDefects

Finder Pattern Defects

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float FinderPatternDefects
```

# EMatrixCodeSemiT10GradingParameters::HorizontalMarkGrowth

Horizontal Mark Growth

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float HorizontalMarkGrowth
```

# EMatrixCodeSemiT10GradingParameters::HorizontalMarkMisplacement

Horizontal Mark Misplacement

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float HorizontalMarkMisplacement
```

# EMatrixCodeSemiT10GradingParameters::SymbolContrast

Symbol Contrast

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float SymbolContrast
```

# EMatrixCodeSemiT10GradingParameters::SymbolContrastSNR

Symbol Contrast SNR

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float SymbolContrastSNR
```

# EMatrixCodeSemiT10GradingParameters::UnusedErrorCorrection

Unused Error Correction

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float UnusedErrorCorrection
```

# EMatrixCodeSemiT10GradingParameters::VerticalMarkGrowth

Vertical Mark Growth

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float VerticalMarkGrowth
```

## EMatrixCodeSemiT10GradingParameters::VerticalMarkMisplacement

Vertical Mark Misplacement

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float VerticalMarkMisplacement
```

# 5.29. EMatrixPosition Struct

Represents a position in a 2D Matrix.

**Namespace:** Euresys::Open_eVision_2_10

### Properties

| | |
|---|---|
| X | X position. |
| Y | Y position. |

## Methods

| | |
|---|---|
| EMatrixPosition | Constructs a default EMatrixPosition object. |
| operator!= | Checks if two EMatrixPosition are stricly different |
| operator== | Checks if two EMatrixPosition are stricly equal |

# EMatrixPosition::EMatrixPosition

Constructs a default EMatrixPosition object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EMatrixPosition(
 )
void EMatrixPosition(
 int x,
 int y
 )
```

### Parameters

*x*

X position.

*y*

Y position.

### Remarks

If the default constructor is used, the position is initialized to (0, 0).

# EMatrixPosition::operator!=

Checks if two EMatrixPosition are stricly different

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool operator!=(
 EMatrixPosition position
 )
```

## Parameters

*position*

The other position.

# EMatrixPosition::operator==

Checks if two EMatrixPosition are stricly equal

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool operator==(
 EMatrixPosition position
 )
```

## Parameters

*position*

The other position.

## EMatrixPosition::X

X position.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int X
```

## EMatrixPosition::Y

Y position.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int Y
```

# 5.30. EObjectData Struct

Describes objects.

### Remarks

An object is characterized by a class, a unique identification number, the number of its constituent runs, the number of its holes (if the object is a real object, not a hole), a selection flag, an identification flag (real object or hole) and the list of its constituent runs. After the object construction phase (real objects and eventually holes), all the objects are gathered in a single dynamic list. The objects can be accessed by their absolute position in the list as well as by their identification number. This structure pertains to the EasyObject legacy API and should not be used for new developments.

**Note.** After a sorting operation, the objects retain their identification number, not their absolute position in the list. If need be, the list of runs of an object can be traversed by means of the following functions: **GetObjNbRun**, ECodedImage::GetObjFirstRunPtr, ECodedImage::GetObjLastRunPtr.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| Class | Class code. |
| IsHole | **TRUE** if the object is a hole, **FALSE** otherwise. |
| IsSelected | Selection flag. |
| ObjNbHole | Number of holes. |
| ObjNbRun | Number of runs. |
| ObjNum | Identification number. |

# EObjectData::Class

Class code.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 Class
```

# EObjectData::IsHole

**TRUE** if the object is a hole, **FALSE** otherwise.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
BOOL IsHole
```

# EObjectData::IsSelected

Selection flag.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_UINT8 IsSelected
```

# EObjectData::ObjNbHole

Number of holes.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 ObjNbHole
```

## EObjectData::ObjNbRun

Number of runs.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 ObjNbRun
```

## EObjectData::ObjNum

Identification number.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 ObjNum
```

# 5.31. EOCR2CharacterCandidate Struct

Holds a single recognition score for a detected character from the image

### Remarks

The variable "code" contains the ASCII-representation of the reference character from the database. The variable "score" contains the recognition score between the detected character and the reference character.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| Code | Contains the ASCII-representation of the reference character from the database. |
| Score | Contains the recognition score between the detected character and the reference character. |

## Methods

| | |
|---|---|
| EOCR2CharacterCandidate | Constructs an EOCR2CharacterCandidate context. |

# EOCR2CharacterCandidate::Code

Contains the ASCII-representation of the reference character from the database.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT16 Code
```

# EOCR2CharacterCandidate::EOCR2CharacterCandidate

Constructs an EOCR2CharacterCandidate context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void EOCR2CharacterCandidate(
 )
void EOCR2CharacterCandidate(
 OEV_UINT16 code,
 float score
 )
```

### Parameters

*code*

-

*score*

-

## EOCR2CharacterCandidate::Score

Contains the recognition score between the detected character and the reference character.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float Score
```

# 5.32. EPath Struct

Path from an image: image pixel coordinates.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| X | Coordinate along the horizontal direction. |
| Y | Coordinate along the vertical direction. |

# EPath::X

Coordinate along the horizontal direction.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 X
```

# EPath::Y

Coordinate along the vertical direction.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 Y
```

# 5.33. EPeak Struct

Represents a peak in a profile.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| Amplitude | Amplitude of the peak. |
| Area | Area of the peak. |
| Center | Center of the peak. |
| Length | Length of the peak. |
| Start | Start of the peak. |

# EPeak::Amplitude

Amplitude of the peak.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 Amplitude
```

# EPeak::Area

Area of the peak.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 Area
```

# EPeak::Center

Center of the peak.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float Center
```

# EPeak::Length

Length of the peak.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
OEV_UINT32 Length
```

## EPeak::Start

Start of the peak.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_UINT32 Start
```

# 5.34. ERenderStyle Struct

Represents how to visualize 3D Objects in a E3DViewer

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

### Properties

| | |
|---|---|
| fillRGB | Fill (inside) color of the object |
| hasFill | Indicates that the inside of the object should be rendered. |
| hasLine | Indicates that the edges of the object should be rendered |
| lineRGB | Line (edge) color of the object. |
| pointRGB | Color of the object if it is a point. |

## Methods

| | |
|---|---|
| ERenderStyle | Constructs a default ERenderStyle object. |

# ERenderStyle::ERenderStyle

Constructs a default ERenderStyle object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
void ERenderStyle(
 )
```

# ERenderStyle::fillRGB

Fill (inside) color of the object

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EC24A fillRGB
```

### Remarks

Only applied if the object is a polygon.

# ERenderStyle::hasFill

Indicates that the inside of the object should be rendered.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool hasFill
```

## Remarks

Only applied if the object is a polygon.

# ERenderStyle::hasLine

Indicates that the edges of the object should be rendered

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
bool hasLine
```

## Remarks

Only applied if the object is a polygon.

# ERenderStyle::lineRGB

Line (edge) color of the object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EC24A lineRGB
```

### Remarks

Only applied if the object is a polygon.

## ERenderStyle::pointRGB

Color of the object if it is a point.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

```
[C++]
EC24A pointRGB
```

# 5.35. ERGB Struct

NTSC/PAL/SMPTE Red, Green, Blue color system.

**Namespace:** Euresys::Open_eVision_2_10

### Properties

| | |
|---|---|
| B | Blue component. |
| G | Green component. |
| R | Red component. |

# ERGB::B

Blue component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float B
```

# ERGB::G

Green component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float G
```

# ERGB::R

Red component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
float R
```

# 5.36. ERGBColor Struct

NTSC/PAL/SMPTE Red, Green, Blue color system.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| Blue | Blue component. |
| Green | Green component. |
| Red | Red component. |

## Methods

| | |
|---|---|
| ERGBColor | Constructs an ERGBColor object. |

# ERGBColor::Blue

Blue component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
int Blue
```

# ERGBColor::ERGBColor

Constructs an ERGBColor object.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ERGBColor(
 int red,
 int green,
 int blue
 )
void ERGBColor(
 )
```

### Parameters

*red*
Red component.

*green*
Green component.

*blue*
Blue component.

# ERGBColor::Green

Green component.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
int Green
```

## ERGBColor::Red

Red component.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
int Red
```

# 5.37. ERun Struct

-

**Namespace:** Euresys::Open_eVision_2_10

### Properties

| | |
|---|---|
| Length | Length of the Run |
| OrgX | X origin of the Run |
| Y | Y position of the Run |

### Methods

| |
|---|

| | |
|---|---|
| ERun | Constructs an ERun context. |
| operator!= | Checks if this instance is not strictly equal to another |
| operator== | Checks if this instance is strictly equal to another |

# ERun::ERun

Constructs an ERun context.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
void ERun(
 )
void ERun(
 int orgX,
 int length,
 int y
 )
```

### Parameters

*orgX*

X origin of the Run.

*length*

Length of the Run.

*y*

Y position of the Run.

# ERun::Length

Length of the Run

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int Length
```

# ERun::operator!=

Checks if this instance is not strictly equal to another

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
bool operator!=(
 ERun other
 )
```

## Parameters

*other*

Reference to the other ERun instance

# ERun::operator==

Checks if this instance is strictly equal to another

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
bool operator==(
 ERun other
 )
```

### Parameters

*other*

Reference to the other ERun instance

# ERun::OrgX

X origin of the Run

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int OrgX
```

# ERun::Y

Y position of the Run

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
int Y
```

# 5.38. ERunData Struct

Describes runs.

### Remarks

A run is characterized by a starting point (**OrgX**, **OrgY**), by a length, a class, a unique identification number and the number of the object to which they belong. After the run construction phase, all the runs are gathered in a single dynamic list.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| Class | Class. |
| Len | Length. |
| ObjNum | Identification number of the object to which the run belongs. |
| OrgX | Start point abscissa. |
| OrgY | Start point ordinate. |

# ERunData::Class

Class.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
OEV_INT32 Class
```

# ERunData::Len

Length.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 Len
```

# ERunData::ObjNum

Identification number of the object to which the run belongs.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 ObjNum
```

# ERunData::OrgX

Start point abscissa.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 OrgX
```

## ERunData::OrgY

Start point ordinate.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
OEV_INT32 OrgY
```

# 5.39. EVSH Struct

Value, Saturation, Hue color system.

**Namespace:** Euresys::Open_eVision_2_10

### Properties

| | |
|---|---|
| H | Hue component. |
| S | Saturation component. |
| V | Value component. |

# EVSH::H

Hue component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float H
```

# EVSH::S

Saturation component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float S
```

# EVSH::V

Value component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
```

```
float V
```

# 5.40. EXYZ Struct

CIE XYZ color system.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| X | X component. |
| Y | Y component. |
| Z | Z component. |

## EXYZ::X

X component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float X
```

## EXYZ::Y

Y component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float Y
```

## EXYZ::Z

Z component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float Z
```

# 5.41. EYIQ Struct

CCIR Luma, Inphase, Quadrature color system.

**Namespace:** Euresys::Open_eVision_2_10

### Properties

| | |
|---|---|
| I | Inphase component. |

| | |
|---|---|
| Q | Quadrature component. |
| Y | Luma component. |

# EYIQ::I

Inphase component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float I
```

# EYIQ::Q

Quadrature component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float Q
```

# EYIQ::Y

Luma component.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float Y
```

# 5.42. EYSH Struct

CCIR Luma, Saturation, Hue color system.

**Namespace:** Euresys::Open_eVision_2_10

## Properties

| | |
|---|---|
| H | Hue component. |
| S | Saturation component. |
| Y | Luma component. |

## EYSH::H

Hue component.

**Namespace:** Euresys::Open_eVision_2_10

[C++]

```
float H
```

## EYSH::S

Saturation component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float S
```

## EYSH::Y

Luma component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float Y
```

# 5.43. EYUV Struct

CCIR Luma, U Chroma, V Chroma color system.

**Namespace:** Euresys::Open_eVision_2_10

### Properties

| | |
|---|---|
| U | U Chroma component. |

| | |
|---|---|
| V | V Chroma component. |
| Y | Luma component. |

# EYUV::U

U Chroma component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float U
```

# EYUV::V

V Chroma component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float V
```

# EYUV::Y

Luma component.

**Namespace:** Euresys::Open_eVision_2_10

```
[C++]
float Y
```

# 6. Enumerations

## 6.1. E3DObjectFeature Enum

The list of possible extracted features for E3DObject

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

| | |
|---|---|
| E3DObjectFeature_Length | Length of the object in metric units. |
| E3DObjectFeature_Width | Width of the object in metric units. |
| E3DObjectFeature_LocalHeight | Local height of the object in metric units. |
| E3DObjectFeature_ReferenceHeight | Reference height of the object in metric units. |
| E3DObjectFeature_Orientation | Orientation of the object. |
| E3DObjectFeature_BoundingBox | 3D oriented bounding box of the object. |
| E3DObjectFeature_AveragePosition | 3D average position of the object. |
| E3DObjectFeature_LocalTopPosition | 3D highest position of the object relatively to the object base plane. |
| E3DObjectFeature_ReferenceTopPosition | 3D top position relative to the ZMap origin (this is the position with the highest Z coordinate). |
| E3DObjectFeature_Plane | Plane fitted to the object 3D positions. |
| | |

| | |
|---|---|
| E3DObjectFeature_LocalTilt | Angle between the object plane and the base plane. |
| E3DObjectFeature_ReferenceTilt | Angle between the object plane and the vertical (Z) axis. |
| E3DObjectFeature_Area | Object area in metric units. |
| E3DObjectFeature_Volume | Object volume in metric units. |
| E3DObjectFeature_BasePlane | Base plane for the object. |
| E3DObjectFeature_BaseTilt | Angle between the base plane and the vertical (Z) axis. |
| E3DObjectFeature_ERectangleRegion | ERectangleRegion enclosing the object ZMap pixels. |
| E3DObjectFeature_ERegion | ERegion composed of the object ZMap pixels. |

# 6.2. EAdaptiveThresholdMethod Enum

Adaptive thresholding modes.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EAdaptiveThresholdMethod_Mean | Use the mean as threshold. |
| EAdaptiveThresholdMethod_Median | Use the median as threshold. |
| EAdaptiveThresholdMethod_Middle | Use the middle of the values interval as threshold. |

# 6.3. EAlignmentPolarity Enum

Polarity of an alignment, used in EFeaturesAligner.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

| | |
|---|---|
| EAlignmentPolarity_ModelToMeasured | The transformation from the model data to the measured data. |
| EAlignmentPolarity_MeasuredToModel | The transformation from the measured data to the model data. |

# 6.4. EAngleUnit Enum

The angle units that are supported by Open eVision.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EAngleUnit_Revolutions | Revolutions (**0..1** corresponds to a full revolution). |
| EAngleUnit_Radians | Radians (**0..2Pi** corresponds to a full revolution). |
| EAngleUnit_Degrees | Degrees (**0..360** corresponds to a full revolution). |
| EAngleUnit_Grades | Grades (**0..400** corresponds to a full revolution). |

# 6.5. EArithmeticLogicOperation Enum

Supported arithmetic or logic pixel-wise operators.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EArithmeticLogicOperation_Copy | Sheer copy. |
| EArithmeticLogicOperation_Invert | Complement. (*) |
| EArithmeticLogicOperation_Add | Saturated addition. (*) |
| EArithmeticLogicOperation_Subtract | Saturated subtraction. (*) |
| EArithmeticLogicOperation_Multiply | Saturated multiplication. (*) |
| EArithmeticLogicOperation_Divide | Saturated division. (*) |
| EArithmeticLogicOperation_Modulo | Modulo. (*) |
| EArithmeticLogicOperation_ShiftLeft | Arithmetic left shift (multiplication by a power of 2). (*) |
| EArithmeticLogicOperation_ShiftRight | Arithmetic right shift (division by a power of 2). (*) |
| EArithmeticLogicOperation_ScaledAdd | Non saturating addition ((left + right) / 2). (*) |
| EArithmeticLogicOperation_ScaledSubtract | Non saturating subtraction ((left + complemented right) / 2). (*) |
| EArithmeticLogicOperation_ScaledMultiply | Non saturating multiplication (left * right / 256 in the **BW8** case, and left * right / 65,536 in the **BW16** one). (*) |
| EArithmeticLogicOperation_ScaledDivide | Non saturating division (256 * left / right in the **BW8** case, and 65,536 * left / right in the **BW16** one). (*) |
| EArithmeticLogicOperation_BitwiseAnd | Bitwise AND. |
| EArithmeticLogicOperation_BitwiseOr | Bitwise OR. |
| | |

| | |
|---|---|
| EArithmeticLogicOperation_BitwiseXor | Bitwise exclusive OR. |
| EArithmeticLogicOperation_LogicalAnd | Logical AND (non zero is **TRUE**). (*) |
| EArithmeticLogicOperation_LogicalOr | Logical OR (non zero is **TRUE**). (*) |
| EArithmeticLogicOperation_LogicalXor | Logical exclusive OR (non zero is **TRUE**). (*) |
| EArithmeticLogicOperation_Min | Minimum. (*) |
| EArithmeticLogicOperation_Max | Maximum. (*) |
| EArithmeticLogicOperation_SetZero | Copy the right operand where the left operand is zero. |
| EArithmeticLogicOperation_SetNonZero | Copy the right operand where the left operand is non zero. |
| EArithmeticLogicOperation_Equal | Equality comparison. (*) |
| EArithmeticLogicOperation_NotEqual | Non equality comparison. (*) |
| EArithmeticLogicOperation_GreaterOrEqual | "Greater or equal" comparison. (*) |
| EArithmeticLogicOperation_LesserOrEqual | "Lesser or equal" comparison. |
| EArithmeticLogicOperation_Greater | "Greater" comparison. (*) |
| EArithmeticLogicOperation_Lesser | "Lesser" comparison. (*) |
| EArithmeticLogicOperation_Compare | Absolute value of the difference. (*) |
| EArithmeticLogicOperation_Overlay | Overlay of one image onto a source image giving a destination image. (See note at the end of the topic). (*) (**) |
| | |

| | |
|---|---|
| EArithmeticLogicOperation_BitwiseNot | Same as EArithmeticLogicOperation_Invert. |
| EArithmeticLogicOperation_Average | Same as EArithmeticLogicOperation_ScaledAdd. (*) |

### Remarks

(*) Not applicable for the **BW1** images/ROIs. (**) In the overlay image, black pixels (0 valued) are considered as transparent. If a **C24** image is used as overlay, all pixels (but the black ones) will be copied to the destination image. If a **BW8** image is used as overlay, all non-black pixels will be converted to the color defined by the **OverlayColor** parameter before copy to the destination image. The destination image is always a **C24** image. If no source image is given (only overlay and destination), the destination image is considered as the source image.

# 6.6. EasyOCR2CharacterFilter Enum

This enumeration contains the possible filters for loading fonts in EOCR2.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EasyOCR2CharacterFilter_ASCII | All ASCII characters are loaded. |
| EasyOCR2CharacterFilter_Letters | Only (alphabetic) letters are loaded. |
| EasyOCR2CharacterFilter_Digits | Only digits are loaded. |

# 6.7. EasyOCR2CharSpacingBias Enum

This enumeration contains the possible biases for the optimised character spacing in the detection phase of EOCR2.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EasyOCR2CharSpacingBias_Wide | The optimisation is biased toward wide character spacing. |
| | |

| | |
|---|---|
| EasyOCR2CharSpacingBias_Neutral | The optimisation is not biased. |
| EasyOCR2CharSpacingBias_Narrow | The optimisation is biased toward narrow character spacing. |

# 6.8. EasyOCR2CharWidthBias Enum

This enumeration contains the possible biases for the optimised character width in the detection phase of EOCR2.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EasyOCR2CharWidthBias_Widest | The optimisation is biased toward very wide boxes. |
| EasyOCR2CharWidthBias_Wide | The optimisation is biased toward wide boxes. |
| EasyOCR2CharWidthBias_Neutral | The optimisation is not biased. |
| EasyOCR2CharWidthBias_Narrow | The optimisation is biased toward narrow boxes. |
| EasyOCR2CharWidthBias_Narrowest | The optimisation is biased toward very narrow boxes. |

# 6.9. EasyOCR2DrawDetectionStyle Enum

This enumeration contains the possible drawing styles for the detection results in EOCR2.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EasyOCR2DrawDetectionStyle_DrawChars | A bounding box is drawn around each individual detected character |
| | |

| | |
|---|---|
| EasyOCR2DrawDetectionStyle_DrawWords | A bounding box is drawn around each detacted word, containing all characters in the word |
| EasyOCR2DrawDetectionStyle_DrawLines | A bounding box is drawn around each detacted line, containing all characters/words in the line |
| EasyOCR2DrawDetectionStyle_DrawText | A bounding box is drawn around the detected text, containing all characters/words/lines in the text |

# 6.10. EasyOCR2DrawRecognitionStyle Enum

This enumeration contains the possible drawing styles for the recognition results in EOCR2.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EasyOCR2DrawRecognitionStyle_LeftTop | The recognition result is drawn at the left top of the character |
| EasyOCR2DrawRecognitionStyle_LeftMiddle | The recognition result is drawn at the left middle of the character |
| EasyOCR2DrawRecognitionStyle_LeftBottom | The recognition result is drawn at the left bottom of the character |
| EasyOCR2DrawRecognitionStyle_BottomLeft | The recognition result is drawn at the bottom left of the character |
| EasyOCR2DrawRecognitionStyle_ BottomMiddle | The recognition result is drawn at the bottom middle of the character |
| EasyOCR2DrawRecognitionStyle_BottomRight | The recognition result is drawn at the bottom right of the character |
| EasyOCR2DrawRecognitionStyle_RightBottom | The recognition result is drawn at the right bottom of the character |
| | |

| | |
|---|---|
| EasyOCR2DrawRecognitionStyle_RightMiddle | The recognition result is drawn at the right middle of the character |
| EasyOCR2DrawRecognitionStyle_RightTop | The recognition result is drawn at the right top of the character |
| EasyOCR2DrawRecognitionStyle_TopRight | The recognition result is drawn at the top right of the character |
| EasyOCR2DrawRecognitionStyle_TopMiddle | The recognition result is drawn at the top middle of the character |
| EasyOCR2DrawRecognitionStyle_TopLeft | The recognition result is drawn at the top left of the character |

# 6.11. EasyOCR2DrawSegmentationStyle Enum

This enumeration contains the possible drawing styles for the segmentation results in EOCR2.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EasyOCR2DrawSegmentationStyle_DrawBlobs | The segmented blobs are drawn directly. |

# 6.12. EasyOCR2TextPolarity Enum

This enumeration contains the possible polarities of text searched during segmentation in EOCR2.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EasyOCR2TextPolarity_WhiteOnBlack | The text is light on a dark background |
| EasyOCR2TextPolarity_BlackOnWhite | The text is dark on a light background |

# 6.13. EAxisSystemType Enum

-

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

| | |
|---|---|
| EAxisSystemType_CornerUpperLeft | - |
| EAxisSystemType_CornerLowerLeft | - |
| EAxisSystemType_Unknown | - |

# 6.14. ECalibrationMode Enum

Allowed values for the calibration mode.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ECalibrationMode_Raw | No calibration at all. |
| ECalibrationMode_Inverse | The ordinate axis points downwards instead of upwards. |
| ECalibrationMode_Scaled | The pixels are assigned a physical size. |
| ECalibrationMode_Anisotropic | The physical size of the pixels differ horizontally and vertically. (Beware there is a restriction pertaining to the allowed image anisotropy. The resulting pixels aspect ratio should be in the range **[-4/3, -3/4]** (or **[3/4, 4/3]**), otherwise the calibration process could fail). |
| ECalibrationMode_Skewed | The coordinate axis make an angle with the image edge. |

| | |
|---|---|
| ECalibrationMode_Tilted | The field-of-view plane is not perpendicular to the optical axis. |
| ECalibrationMode_Radial | The lens introduces some amount of radial distortion. |
| ECalibrationMode_Bilinear | This mode can not be combined with other calibration mode. The bilinear calibration is based on a first order polynomial approach. |
| ECalibrationMode_Quadratic | This mode can not be combined with other calibration mode. The quadratic calibration is based on a second order polynomial approach. |

# 6.15. ECalibrationType Enum

The Easy3D calibration type.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

| | |
|---|---|
| ECalibrationType_Scale | Calibration type is EScaleCalibrationModel. |
| ECalibrationType_ExplicitGeometric | Calibration type is EExplicitGeometricCalibrationModel. |
| ECalibrationType_ObjectBased | Calibration type is EObjectBasedCalibrationModel. |
| ECalibrationType_Unknown | Calibration type is not defined. |

# 6.16. ECannyThresholdingMode Enum

The thresholding modes for the Canny edge detector.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ECannyThresholdingMode_Relative | Relative thresholding mode. |
| ECannyThresholdingMode_Absolute | Absolute thresholding mode. |

# 6.17. ECC000Family Enum

-

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

| | |
|---|---|
| ECC000Family_ECC000 | - |
| ECC000Family_ECC050 | - |
| ECC000Family_ECC080 | - |
| ECC000Family_ECC100 | - |
| ECC000Family_ECC140 | - |
| ECC000Family_Unknown | - |

# 6.18. ECharCreationMode Enum

Allowed values for the character creation mode in EasyOCV.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ECharCreationMode_Group | Form a single character comprising all blobs. |
| ECharCreationMode_Overlap | Form single characters out of blobs whose bounding box overlap. |
| ECharCreationMode_Separate | Form one character for each blob. |

# 6.19. EClippingMode Enum

Allows to choose how the fitted segment length and centre are computed

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EClippingMode_CenteredNominal | Regular mode: the fitted segment always has the nominal length of the line gauge. |
| EClippingMode_ClippedToValidSamples | The fitted segment does not extend beyond valid samples. It is clipped to the projection of the valid samples on the fitted line. |
| EClippingMode_ClippedInNominalShape | The segment is built by clipping the fitted line in the rectangular range where the ELineGauge looks for valid transition samples, i.e. the rectangle which is centered and aligned on the ELineGauge nominal line, and which height is two times the ELineGauge::Tolerance. |

# 6.20. EColorQuantization Enum

Allowed values for the quantization mode in EasyColor.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EColorQuantization_FullRange | Values are quantized in range **0..255**. |
| EColorQuantization_Ccir601 | Values are quantized in range **16..235** for the R, G, B or Y component, and in range **16..240** for the I, Q, U and y components. |

### Remarks

When quantizing the color values for the RGB or YIQ/YUV representation, one usually uses the full **0..255** range. Anyway, the CCIR has defined an alternate convention such that some values in this interval are reserved. Before performing a conversion, functions EasyColor::SrcQuantization and EasyColor::DstQuantization can be used to specify the rule used.

# 6.21. EColorRampMode Enum

This enumeration contains the possible values for the parameter of E3DViewer::GenerateColors method.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

| | |
|---|---|
| EColorRampMode_HueFromZ | The Hue color component calculated from Z coordinate. The colors range from Red (minimum Z) to Blue (maximum Z). |
| EColorRampMode_RGBCube | RGB colors directly calculated from XYZ coordinates. |

# 6.22. EColorSystem Enum

The color systems that are supported by Open eVision.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EColorSystem_NoColor | Undefined |
| EColorSystem_Bilevel | Binary black & white. |
| EColorSystem_GrayLevel | Continuous tone black & white. |
| EColorSystem_Xyz | CIE XYZ. |
| EColorSystem_Rgb | NTSC/PAL/SMPTE Red, Green, Blue. |
| EColorSystem_Lab | CIE Lightness, a*, b*. |
| EColorSystem_Luv | CIE Lightness, u*, v*. |
| EColorSystem_Yuv | CCIR Luma, U Chroma, V Chroma. |
| EColorSystem_Yiq | CCIR Luma, Inphase, Quadrature. |
| EColorSystem_Lch | Lightness, Chroma, Hue. |
| EColorSystem_Ish | Intensity, Saturation, Hue |
| EColorSystem_Lsh | Lightness, Saturation, Hue. |
| EColorSystem_Vsh | Value, Saturation, Hue. |

| | |
|---|---|
| EColorSystem_Ysh | CCIR Luma, Saturation, Hue. |

### Remarks

Open eVision supports several color systems. The achromatic ones are related to black and white and gray-level images (EImageBW1 and EImageBW8). The remaining ones apply to color images (EImageC24). Also see unquantized and quantized colors for the allowed ranges of values.

# 6.23. EConnexity Enum

Possible values for the connexity of a contour.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EConnexity_Connexity4 | Pixels touching by an edge are considered connected. |
| EConnexity_Connexity8 | Pixels touching by an edge or a corner are considered connected. |

# 6.24. EContourMode Enum

Possible modes for contour traversal.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EContourMode_Clockwise | The contour is traversed clockwise. |
| EContourMode_ClockwiseAlwaysClosed | The contour is traversed clockwise and the image border may be followed if necessary to close the contour. |
| EContourMode_ClockwiseContinueIfBorder | Contour traversal is restarted counterclockwise when an image border is met. |
| | |

| | |
|---|---|
| EContourMode_Anticlockwise | The contour is traversed counterclockwise. |
| EContourMode_ AnticlockwiseContinueIfBorder | Contour traversal is restarted clockwise when an image border is met. |
| EContourMode_AnticlockwiseAlwaysClosed | The contour is traversed anticlockwise and the image border may be followed if necessary to close the contour. |

# 6.25. EContourThreshold Enum

Allowed thresholding modes for contour traversal.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EContourThreshold_Above | Traverse the pixels just above the threshold. |
| EContourThreshold_Below | Traverse the pixels just below the threshold. |

# 6.26. ECorrelationMode Enum

Allowed values for the EasyMatch correlation mode.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ECorrelationMode_Standard | Correlation sensitive to changes in intensity and/or contrast (computed from the raw image/pattern gray values). |
| ECorrelationMode_OffsetNormalized | Correlation made unsensitive to changes in intensity (computed from the centered image/pattern gray values). |
| | |

| | |
|---|---|
| ECorrelationMode_GainNormalized | Correlation made unsensitive to changes in contrast (computed from the reduced image/pattern gray values). |
| ECorrelationMode_Normalized | Correlation made unsensitive to changes in both intensity and contrast (computed from the centered and reduced image/pattern gray values). Default mode. |

# 6.27. EDataSize Enum

Possible data sizes for an object feature.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EDataSize_BitsPerPixel1 | bit |
| EDataSize_BitsPerPixel8 | byte |
| EDataSize_BitsPerPixel16 | word |
| EDataSize_BitsPerPixel32 | long word |
| EDataSize_BitsPerPixel64 | quad word |
| EDataSize_BitsPerPixel24 | 3 bytes |
| EDataSize_BitsPerPixel96 | 12 bytes |

# 6.28. EDataType Enum

Possible data types for an object feature.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EDataType_UnsignedInt | Unsigned integer. |
| EDataType_SignedInt | Signed integer. |
| EDataType_Float | Floating-point. |

# 6.29. EDegreesOfFreedom Enum

Allowed values for the degrees of freedom in EChecker.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EDegreesOfFreedom_Translation | Translation allowed. |
| EDegreesOfFreedom_Rotation | Rotation allowed. |
| EDegreesOfFreedom_Scaling | Scaling allowed. |

# 6.30. EDiagnostic Enum

Possible defect codes in EasyOCV.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EDiagnostic_CharNotFound | Insufficient location score for a character. |
| EDiagnostic_CharOverprinting | Too much inking on a character. |
| EDiagnostic_CharUnderprinting | Too little inking on a character. |
| EDiagnostic_CharMismatch | Wrong character shape. |
| EDiagnostic_TextNotFound | Insufficient location score for a text. |
| EDiagnostic_TextOverprinting | Too much inking on a text. |
| EDiagnostic_TextUnderprinting | Too little inking on a text. |
| EDiagnostic_TextMismatch | Wrong text shape. |
| EDiagnostic_BadContrast | Global contrast (of inspection ROI) out of range. |
| EDiagnostic_Undefined | - |

# 6.31. EDoubleThresholdMode Enum

The double threshold mode for the selection of coded elements with respect to a given feature.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EDoubleThresholdMode_Inside | The value of the feature must be greater or equal to the low threshold, and strictly less than the high threshold. |
| EDoubleThresholdMode_Outside | |

| | |
|---|---|
| | The value of the feature must be strictly less than the low threshold, or greater or equal to the high threshold. |

# 6.32. EDraggingMode Enum

Defines how the shape could be dragged

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EDraggingMode_Standard | Allows positioning the shape edges symmetrically. |
| EDraggingMode_ToEdges | Allows positioning each shape edge individually. |

# 6.33. EDragHandle Enum

Allowed values for a handle identifier in the context of handle dragging.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EDragHandle_NoHandle | No handle. |
| EDragHandle_Inside | Inside handle. |
| EDragHandle_North | Northern handle. |
| EDragHandle_East | Eastern handle. |
| EDragHandle_South | Southern handle. |
| EDragHandle_West | Western handle. |

| | |
|---|---|
| EDragHandle_NorthWest | North-Western handle. |
| EDragHandle_SouthWest | South-Western handle. |
| EDragHandle_NorthEast | North-Eastern handle. |
| EDragHandle_SouthEast | South-Eastern handle. |
| EDragHandle_Center | Circle, rectangle or wedge center handle. |
| EDragHandle_Org | Line segment or circle arc origin handle. |
| EDragHandle_Mid | Line segment or circle arc middle handle. |
| EDragHandle_End | Line segment or circle arc end handle. |
| EDragHandle_Tol0 | First tolerance handle. |
| EDragHandle_Tol1 | Second tolerance handle. |
| EDragHandle_Tol_x0 | Rectangle leftmost first tolerance handle. |
| EDragHandle_Tol_x1 | Rectangle leftmost second tolerance handle. |
| EDragHandle_Tol_y0 | Rectangle lower first tolerance handle. |
| EDragHandle_Tol_y1 | Rectangle lower second tolerance handle. |
| EDragHandle_Tol_XX0 | Rectangle rightmost first tolerance handle. |
| EDragHandle_Tol_XX1 | Rectangle rightmost second tolerance handle. |
| EDragHandle_Tol_YY0 | Rectangle upper first tolerance handle. |

| | |
|---|---|
| EDragHandle_Tol_YY1 | Rectangle upper second tolerance handle. |
| EDragHandle_Tol_a0 | Wedge leftmost first tolerance handle. |
| EDragHandle_Tol_a1 | Wedge leftmost second tolerance handle. |
| EDragHandle_Tol_AA0 | Wedge rightmost first tolerance handle. |
| EDragHandle_Tol_AA1 | Wedge rightmost second tolerance handle. |
| EDragHandle_Tol_r0 | Wedge inner first tolerance handle. |
| EDragHandle_Tol_r1 | Wedge inner second tolerance handle. |
| EDragHandle_Tol_RR0 | Wedge outer first tolerance handle. |
| EDragHandle_Tol_RR1 | Wedge outer second tolerance handle. |
| EDragHandle_Edge_x | Rectangle leftmost edge handle. |
| EDragHandle_Edge_XX | Rectangle rightmost edge handle. |
| EDragHandle_Edge_y | Rectangle lower edge handle. |
| EDragHandle_Edge_YY | Rectangle upper edge handle. |
| EDragHandle_Edge_a | Wedge leftmost edge handle. |
| EDragHandle_Edge_AA | Wedge rightmost edge handle. |
| EDragHandle_Edge_r | Wedge outer edge handle. |
| EDragHandle_Edge_RR | Wedge inner edge handle. |

# 6.34. EDrawableFeature Enum

The various features that can be drawn for coded elements.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EDrawableFeature_BoundingBox | The bounding box. |
| EDrawableFeature_ConvexHull | The convex hull. |
| EDrawableFeature_Ellipse | The ellipse of inertia. |
| EDrawableFeature_FeretBox22 | The Feret box oriented at 22.5°. |
| EDrawableFeature_FeretBox45 | The Feret box oriented at 45°. |
| EDrawableFeature_FeretBox68 | The Feret box oriented at 67.5°. |
| EDrawableFeature_GravityCenter | The gravity center. |
| EDrawableFeature_ MinimumEnclosingRectangle | The minimum enclosing rectangle. |
| EDrawableFeature_FeretBox | The Feret box oriented at a fixed angle. Only available for selections of coded elements: The angle of interest is set through the EObjectSelection::FeretAngle property. |
| EDrawableFeature_WeightedGravityCenter | The gravity center of the pixels of the attached image over the coded element. Only available for selections of coded elements: The attached image is set through the EObjectSelection::AttachedImage property. |

# 6.35. EDrawingMode Enum

Allowed modes to draw the bounding box of a symbol.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EDrawingMode_Nominal | Draws the nominal point location or model fitting gauge. |
| EDrawingMode_Actual | Draws the located point or the fitted model. |
| EDrawingMode_SampledPaths | Draws the sampled segments along the model. |
| EDrawingMode_SampledPath | Draws the sampled segment specified by ELineGauge::MeasureSample. |
| EDrawingMode_PointsInSkipRange | Draws the skipped sampled points in addition to the non-skipped points. |
| EDrawingMode_SampledPoints | Draws the sampled points along the model. |
| EDrawingMode_SampledPoint | Draws the sampled point specified by ELineGauge::MeasureSample. |
| EDrawingMode_InvalidSampledPoints | Draws the invalid sampled points along the model. |
| EDrawingMode_Learn | Draw the pattern learning ROI(s). |
| EDrawingMode_Match | Draw the pattern searching ROI(s). |
| EDrawingMode_Position | Draw the found pattern ROI(s). |
| EDrawingMode_Inspected | Draw the inspected ROI. |

| | |
|---|---|
| EDrawingMode_MaxInspected | Draw the largest possible inspected ROI. |

# 6.36. EEditionMode Enum

This enumeration is used to select which graphical interactions are allowed.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EEditionMode_None | The object cannot be edited. |
| EEditionMode_Move | The object can be moved. |
| EEditionMode_Rotate | The object can be rotated. |
| EEditionMode_Stretch | The object can be stretched. |
| EEditionMode_EdgeSplit | The object can have one of its edges split. |
| EEditionMode_All | All graphical interactions are allowed. |

# 6.37. EEncodingConnexity Enum

The connexity mode for the encoding process.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EEncodingConnexity_Four | Pixels touching by an edge are considered connected. |
| EEncodingConnexity_Eight | Pixels touching by an edge or a corner are considered connected. |

# 6.38. EError Enum

Possible Open eVision error codes.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EError_Ok | Success |
| EError_EndOfImageSequence | End of image sequence |
| EError_UserDialogFailed | User dialog failed |
| EError_ImageLimitsReached | Image limits reached |
| EError_InvalidAsciiPadding | Invalid ASCII padding |
| EError_InvalidOperation | Invalid operation - See Reference |
| EError_InvalidBitsPerPixel | Invalid depth (bits per pixel) - Check type compatibility |
| EError_InvalidDataType | Invalid data type - Check type compatibility |
| EError_InvalidDataSize | Invalid data size - Check type compatibility |
| EError_ParametersOutOfRange | Parameters out of range - See Reference |
| EError_InvalidMode | Invalid mode - See Reference |
| EError_EndSmallerThanStart | End smaller than start - Adjust indices |
| EError_Parameter1OutOfRange | Parameter 1 out of range - See Reference |

| | |
|---|---|
| EError_Parameter2OutOfRange | Parameter 2 out of range - See Reference |
| EError_Parameter3OutOfRange | Parameter 3 out of range - See Reference |
| EError_Parameter4OutOfRange | Parameter 4 out of range - See Reference |
| EError_Parameter5OutOfRange | Parameter 5 out of range - See Reference |
| EError_Parameter6OutOfRange | Parameter 6 out of range - See Reference |
| EError_Parameter7OutOfRange | Parameter 7 out of range - See Reference |
| EError_Parameter8OutOfRange | Parameter 8 out of range - See Reference |
| EError_Parameter9OutOfRange | Parameter 9 out of range - See Reference |
| EError_Parameter10OutOfRange | Parameter 10 out of range - See Reference |
| EError_WindowsError | Out of GDI handles |
| EError_InvalidPlanesPerPixel | Invalid planes per pixel - Check image type or file contents |
| EError_BW1ImageExpected | 1 bit black & white (**BW1**) image expected - Check image type or file contents |
| EError_BW8ImageExpected | 8 bits black & white (**BW8**) image expected - Check image type or file contents |
| EError_BW16ImageExpected | 16 bits black & white (**BW16**) image expected - Check image type or file contents |
| EError_BW32ImageExpected | 32 bits black & white (**BW32**) image expected - Check image type or file contents |
| | |

| | |
|---|---|
| EError_TemplateCallNeedsSpecialization | Template call needs specialization |
| EError_CannotCreateMutex | Cannot create Mutex |
| EError_CannotLockMutex | Cannot lock Mutex |
| EError_CannotUnlockMutex | Cannot unlock Mutex |
| EError_CannotDeleteMutex | Cannot delete Mutex |
| EError_TimeoutReached | Timeout reached |
| EError_FunctionNotFound | Function not found |
| EError_ProcessStopped | Process stopped |
| EError_CopyNotAllowed | Copy not allowed |
| EError_SingularMatrix | Internal error (code: 1050) |
| EError_DivisionByZero | Division by zero - Check parameters |
| EError_ReadonlyProperty | This property is read-only and cannot be accessed |
| EError_UndefinedProperty | This property is undefined and cannot be accessed |
| EError_ItemNotFound | The data structure does not contain the specified item |
| EError_NextItemNotFound | The specified item has no next sibling |
| EError_InlineAssemblyDisabled | The inline assembly code for MMX is disabled in this build. This error should only be returned for builds compiled with GCC, or if the symbol E_MMX is left undefined. |
| | |

| | |
|---|---|
| EError_FileAccessProblems | File access problems - Check file pathname and device state |
| EError_FileCouldNotBeOpened | File could not be opened - Check file pathname, troubleshoot disk device |
| EError_FilwhileReading | File error while reading - Check file integrity, troubleshoot disk device |
| EError_FilwhileWriting | File error while writing - Check free disk space, troubleshoot disk device |
| EError_BadFileFormat | Bad file format - Check file source or contents |
| EError_FileCouldNotBeClosed | File could not be closed - Check free disk space, troubleshoot disk device |
| EError_UnsupportedFileFormatVersion | Unsupported file format version - Upgrade to newer release |
| EError_MissingOrUnsupportedFileExtension | Missing or unsupported file extension - Check file name/format match |
| EError_FileIsReadOnly | File is read-only - Set to read-write or save under another name |
| EError_UnsupportedObjectTypeInArchive | The archive does not contain the correct object type - Check your archiving routines |
| EError_UnknownArchiveError | An unexpected error occurred during archive access |
| | |

| | |
|---|---|
| EError_SerializerShouldBeInReadMode | Attempting to read with a serializer not open for read access |
| EError_SerializerShouldBeInWriteMode | Attempting to write with a serializer not open for write access |
| EError_FileExists | Attempting to overwrite an existing file while not allowed to do so |
| EError_SerializerNotOpen | Attempting to use a serializer that is not correctly opened |
| EError_UnrecognizedFileFormat | Unrecognized File Format |
| EError_ WrongColorFormatFileFormatCombination | Wrong Color Format File Format Combination |
| EError_FileDoesNotExist | Attempting to open a file which doesn't exist |
| EError_ObjectTooLargeToBeSerialized | Object is too large to be serialized |
| EError_UnsupportedFileFormat | Unsupported File Format |
| EError_UnsupportedTiffFormat | Unsupported TIFF format - Convert TIFF file |
| EError_UnsupportedBmpFormat | Unsupported BMP format - Convert BMP file |
| EError_InvalidPngCompression | Invalid PNG compression level |
| EError_UnsupportedJpegFormat | Unsupported JPEG format - Convert JPEG file |
| EError_BilevelImageExpected | Bi-level image expected - Use **BW1** |
| EError_GrayLevelImageExpected | Gray-level image expected - Use **BW8** |
| EError_ColorImageExpected | Color image expected - Use **C24** |

| | |
|---|---|
| EError_BilevelFormatExpected | Bi-level format expected - Convert file to black & white |
| EError_GrayLevelFormatExpected | Gray-level format expected - Convert file to gray shades |
| EError_ColorFormatExpected | Color format expected - Convert file to true color |
| EError_CannotReadJpegFile | Cannot read JPEG file - Troubleshoot disk device |
| EError_CannotWriteJpegFile | Cannot write JPEG file - Troubleshoot disk device |
| EError_WrongFileExtension | Wrong file extension |
| EError_UnableToAllocateTemporaryMemory | Unable to allocate temporary memory - Fix memory leaks or release memory |
| EError_BufferTooSmall | The supplied buffer is too small. Supply a bigger buffer. |
| EError_UnableToAllocateMemory | Not enough memory for this allocation. |
| EError_UnableToAccessImageMemory | Unable to access image memory - Load or size image |
| EError_RoiTooLarge | ROI too large - Fit ROI to image |
| EError_NotAValidImage | Not a valid image - Check image contents |
| EError_ImagesNotSameSize | Images not of the same size - Adjust image size(s) |
| EError_ImagesNotSameBitsPerPixel | Images not of the same depth (bits per pixel) - Choose compatible types |
| EError_SourceImageTooSmall | Origin image too small - Use a larger image |
| | |

| | |
|---|---|
| EError_PixelsMustHaveFiniteSize | Pixels must have finite size - Use non-zero parameters |
| EError_ConstantIsNull | Constant is NULL - Use non-zero value |
| EError_PixelNullEncountered | NULL pixel encountered - Avoid division by zero |
| EError_ImagesMayNotOverlap | Images cannot overlap - Use distinct images |
| EError_RoiOutOfImageLimits | ROI out of the top parent limits - Resize to fit in image |
| EError_RoiAlreadyHasAParent | ROI already has a parent - Detach ROI first |
| EError_RoiHasNoParentImage | There is no image ancestor for this ROI - Attach the ROI or one of its ancestor to an image |
| EError_CannotApplyToAnImage | Cannot apply to an image - Apply to a ROI instead |
| EError_UnsupportedImageType | Unsupported image type - Check type compatibility |
| EError_InvalidImageType | Invalid image type - Check type compatibility |
| EError_UnsupportedXserverDepth | Unsupported X server depth |
| EError_InconsistentRoiHierarchy | The hierarchy of ROI has been corrupted (inconsistent parent/daughters relationship) |
| EError_SourceImageTooBig | Original image too big - Use a smaller image |
| EError_BW1RoiNotAligned | First bit index of an aligned ROI must be 0 - Use an aligned ROI |
| EError_WrongRoiType | Wrong ROI or image type |
| EError_CyclingParenthoodNotAllowed | Cycling Parenthood not allowed |
| | |

| | |
|---|---|
| EError_WrongBitsPerRow | Bits per row must be a multiple of 32 (4 bytes) and must be enough to hold all the pixels of an image row. |
| EError_MisalignedImagePtr | The supplied image pointer must be aligned to 4 bytes. |
| EError_UnsupportedImageTypeConversion | Unsupported image type conversion |
| EError_ImageFromFileDoesNotFitIntoROI | The ROI is not the same size as the image file. When loading an image from a file into a ROI, the ROI must have the exact required size. On the other, when loading into an image object, it gets resized to the correct size. |
| EError_PixelCoordinatesOutOfROI | The specified coordinate is outside the ROI/Image |
| EError_ROIFromFileDoesNotFitIntoROI | The ROI is not the same size as the ROI previsously saved. ROIs directly linked to an pixel container must have the same size as the container and have a (0, 0) origin. |
| EError_ROIHasZeroArea | The ROI width and height must both be larger than 0 - resize the ROI. |
| EError_RegionTooSmall | The region is too small. |
| EError_ERegionHasNotBeenPrepared | The ERegion has not been prepared, please use function ERegion::Prepare(). |
| EError_ERegionImpossibleCopy | Cannot copy non-prepared geometrical region into a base ERegion instance. |
| EError_PixelOutsidePerimeter | Pixel outside perimeter - Check pixel value |
| | |

| | |
|---|---|
| EError_PixelInsidePerimeter | Pixel inside perimeter - Check pixel value |
| EError_IsolatedPixel | Isolated pixel - Check pixel value |
| EError_MaxPixelInContourReached | Maximum pixels in contour reached |
| EError_NotAValidContour | Not a valid contour - Initialize using a contouring function |
| EError_UnableToAccessVectorMemory | Unable to access vector memory - Check proper vector initialization |
| EError_NotAValidVectorDescriptor | Not a valid vector descriptor |
| EError_VectorTypeIsNotHist | Vector type is not histogram |
| EError_NotEnoughGroupsInVector | Not enough groups in vector |
| EError_InvalidVectorDataSize | Invalid vector data size |
| EError_InvalidVectorDataType | Invalid vector data type |
| EError_InvalidVectorType | Invalid vector type |
| EError_ResultTooBigToFitInVector | Result too big to fit in vector |
| EError_GroupOutOfRange | Group out of range - Adjust group index |
| EError_InvalidVectorGroupLength | Invalid vector group length |
| EError_InvalidNumberOfVectorElements | Invalid number of vector elements - Check proper vector initialization |
| EError_VectorsNotSameSize | Vectors not of the same size - Adjust vector size(s) |

| | |
|---|---|
| EError_UnableToAccessKernelMemory | Unable to access kernel memory - Check proper kernel initialization |
| EError_NotAValidKernelDescriptor | Not a valid kernel descriptor |
| EError_InvalidKernel | Invalid kernel |
| EError_KernelInvalidSize | Invalid kernel size - Check proper kernel initialization |
| EError_KernelNotAllocated | Kernel not allocated - Check proper kernel initialization |
| EError_BadListPosition | Bad list position - Restart list traversal |
| EError_ListIsEmpty | List is empty |
| EError_TopOfList | Top of list - Do not traverse backwards |
| EError_BotOfList | Bottom of list - Do not traverse forwards |
| EError_ListError | List error |
| EError_LicenseMissing | The license for this library is not granted - Launch License Manager |
| EError_EasyImageLicenseMissing | The license for EasyImage is not granted - Launch License Manager |
| EError_EasyColorLicenseMissing | The license for EasyColor is not granted - Launch License Manager |
| EError_EasyObjectLicenseMissing | The license for EasyObject is not granted - Launch License Manager |
| EError_EasyMatchLicenseMissing | The license for EasyMatch is not granted - Launch License Manager |

| | |
|---|---|
| EError_EasyGaugeLicenseMissing | The license for EasyGauge is not granted - Launch License Manager |
| EError_EasyFindLicenseMissing | The license for EasyFind is not granted - Launch License Manager |
| EError_EasyOcrLicenseMissing | The license for EasyOCR is not granted - Launch License Manager |
| EError_EasyOcvLicenseMissing | The license for EasyOCV is not granted - Launch License Manager |
| EError_EasyBarCodeLicenseMissing | The license for EasyBarCode is not granted - Launch License Manager |
| EError_EasyMatrixCodeLicenseMissing | The license for EasyMatrixCode is not granted - Launch License Manager |
| EError_ EasyMatchAlignementModeLicenseMissing | The license for EasyMatch Alignment mode is not granted - Launch License Manager |
| EError_EvisionStudioLicenseMissing | The license for eVison Studio is not granted - Launch License Manager |
| EError_InvalidDongleIndex | The index do not match any available dongle |
| EError_CannotWriteOEMKey | The OEM key cannot be set |
| EError_WarpImagesTooSmall | Warp images too small - Increase image size |
| EError_UnsupportedImageSize | Unknown error code |
| EError_InvalidThresholdValue | The threshold value is not supported |
| EError_UnknownFeature | Unknown feature - Check parameters |
| EError_InvalidSelectionArgument | Invalid selection argument - Check parameters |

| | |
|---|---|
| EError_SortListTooLong | Sort list too long |
| EError_NotAValidOperationCode | Not a valid operation code |
| EError_TooManyObjectsDetected | Too many objects detected - Increase **MaxObjects** |
| EError_InvalidFeature | Invalid feature - Check parameters |
| EError_FeatureNotCalculated | Feature not calculated - Call **AnalyseObjects** method |
| EError_BadObjectNumber | Bad object number - Check parameters |
| EError_NoObjectSelected | No object selected - Blob list is empty |
| EError_ LowThresholdHigherThanHighThreshold | Low threshold higher than high threshold - Adjust thresholds |
| EError_InvalidThresholdMode | Invalid threshold mode - Use appropriate threshold setting method |
| EError_NoImageAttached | No image attached to the selection - Use Attach() |
| EError_OutOfContinuousMode | Invalid call out of continuous mode |
| EError_InvalidImageTypeForSegmenter | The current segmenter can not cope with this type of image |
| EError_LayersOverlapping | Two different layers are associated with the same layer index |
| EError_EndOfIterator | The iterator has reached the end of the enumeration |
| EError_NoThresholdComputedYet | The threshold valued has not been computed yet - First encode an image |
| EError_FeatureNotDrawable | This kind of feature cannot be drawn |

| | |
|---|---|
| EError_OnlyApplicableToObjectSelection | This kind of feature cannot be used out of EObjectSelection |
| EError_MoreThanOneLayerEncoded | Please specify the layer index (several layers are encoded) |
| EError_CodedElementNotSelected | The coded element is not present in the selection |
| EError_NoPatternLearnt | No pattern learnt - Load from file or train pattern |
| EError_PatternTooLarge | Pattern too large - Use a smaller one |
| EError_PatternTooSmall | Pattern too small - Use a larger one |
| EError_NotAnEasyMatchFile | Not an EasyMatch file - Check file source |
| EError_UnsupportedEasyMatchFileVersion | Unsupported EasyMatch file version - Upgrade to a newer release |
| EError_NoImageLearnt | No image learnt - Call **LearnImage**() first |
| EError_WrongNumberOfDegreesOfFreedom | The number of degrees of freedom must be at least one, and no more than five - Use a value in this range |
| EError_InsufficientContrast | Not enough feature points - Use a more contrasted pattern or reduce the Don't Care mask |
| EError_PatternTooCloseToImageBorder | Pattern is too close to image border - Leave a margin around the pattern |
| EError_IncompatibleModes | Incompatible modes (CoarseToFineAnalysisMode and PatternType) |
| EError_ AllowancesAndPatterntypeNotCompatible | Angle and Scale allowances can not be used with the current pattern type |
| EError_ModelNotSuitedForContrastingregions | The model is unsuitable for ContrastingRegions pattern type - Try an other pattern type or increase surface of region(s) |

| | |
|---|---|
| EError_ModelNotSuitedForConsistentedges | The model is unsuitable for ConsistentEdges pattern type - Try another pattern type or increase the model surface |
| EError_NoPatternsLoaded | No patterns loaded - Load font file or train |
| EError_NoPatternsInTheseClasses | No patterns in these classes - Check pattern and text class assignments |
| EError_CharacterTooSmall | Character too small - Enlarge to font size |
| EError_CharacterCodeTooBig8 | Character code too big to fit in a string, use ReadTextWide instead |
| EError_CharacterCodeTooBig16 | Character code too big to fit in a wide string, use GetFirstCharCode instead |
| EError_InvalidTextStructure | Text parameter doesn't fit the text topology |
| EError_InvalidFontFile | The specified font file couldn't be loaded |
| EError_InvalidTopology | The specified topology is invalid |
| EError_InvalidEOCR2File | The file-type and structure could not be verified |
| EError_EOCR2InvalidCharWidth | Character widths must be larger than 0 |
| EError_EOCR2InvalidCharWidthTolerance | Character width tolerance must be between 0 and 1 |
| EError_EOCR2InvalidCharHeight | Character height must be larger than 0 |
| EError_EOCR2InvalidMaxVariation | The 'maximum variation' parameter must be between 0 and 1. |
| EError_EOCR2InvalidDetectionDelta | The 'detection delta' parameter must be between 0 and 128. |
| EError_EOCR2InvalidMaxFragmentation | |

| | |
|---|---|
| EError_EOCR2InvalidSpaceWidth | The 'maximum fragmentation' parameter must be between 0 and 1. |
| s must be larger than or equal to 0. | |
| EError_EOCR2InvalidNumDetectionPasses | NumDetectionPasses must be either 1 or 2. |
| EError_EOCR2CharCodeNotSet | Character code not set |
| EError_EOCR2CharHeightNotSet | Character height not set |
| EError_EOCR2CharWidthNotSet | Character width not set |
| EError_EOCR2TopologyNotSet | Topology not set |
| EError_EOCR2DetectionFailed | The given topology and parameters could not be fitted to the detected blobs. |
| EError_MismatchingColorSystem | Mismatching color system - Check transform compatibility |
| EError_ColorLookupMustBeInitialized | Color lookup must be initialized - Use initialization method |
| EError_UnsupportedColorTransform | Unsupported color transform |
| EError_UnknownSymbolSize | Unknown symbol size - Check size initialization |
| EError_UnknownEccFamily | Unknown ECC family (ECC 000/050/080/100/140/200 only) |
| EError_UncorrectableErrors | Too many errors, cannot correct contents - See Reference |
| EError_CouldNotLocateSymbol | Could not locate the dot matrix symbol (no good candidate object) - See Reference |
| EError_UnknownFormatId | Unknown Format ID in ECC 000-140 symbol (Base 11/27/41/37 and ASCII 7/8 only) - See Reference |

| | |
|---|---|
| EError_InvalidCrc | Invalid CRC after error correction in ECC 000-140 symbol - See Reference |
| EError_NoCodeFound | Could not find any codes in the image |
| EError_TimeoutReachedAndNoCodeFound | Could not find any codes in the image within the timeout |
| EError_CouldNotDecodeSymbol | Could not decode symbol - Try to improve image quality |
| EError_CouldNotGrade | Could not grade symbol - Quiet zone out of bounds |
| EError_CouldNotLocateBarcode | Could not locate bar code symbol - Improve contrast, avoid clutter |
| EError_UnrecognizedSignature | Unrecognized signature - Check enabled symbologies |
| EError_InvalidNumberOfBars | Invalid number of bars - Improve bar/space contrast |
| EError_ExtraEdgesFound | Extra edges found - Improve bar/space contrast or uniformity |
| EError_IncoherentBarSpaceThickness | Incoherent bar/space thickness sequence - Check enabled symbologies |
| EError_InvalidCheckCharacter | Invalid checksum character - Check enabled symbologies |
| EError_SymbologyNotEnabled | Symbology not enabled - Invoke method **SetSymbologies**() |
| EError_NoEdgesFound | No edges found - Adjust location or improve bar/space contrast |
| EError_InvalidEMailBarcodeReaderFile | The file-type and structure could not be verified |
| EError_NotAnEasyOcvFile | Not an EasyOCV file - Check file source |
| EError_UnsupportedEasyOcvFileVersion | Unsupported EasyOCV file version - Upgrade Open eVision |

| | |
|---|---|
| EError_NotEnoughSampleImages | Not enough sample images - Use **AddToStatistics** |
| EError_NotAnEcheckerFile | Not an EChecker file - Check file source |
| EError_UnsupportedEcheckerFileVersion | Unsupported EChecker file version - Upgrade Open eVision |
| EError_NotEnoughSamplesLearnt | Not enough samples learnt - Use **UpdateStatistics** |
| EError_InvalidNormalizationMode | Invalid normalization mode - Check **SetNormalize** call |
| EError_ImageNotRegistered | Image not registered - Use method **Register** before **Learn** |
| EError_InvalidLearningSequence | Invalid learning sequence - Use AVERAGE followed by ABS_ DEVIATION, or RMS_DEVIATION, then READY |
| EError_E_ERROR_CONTRAST_TOO_LOW | Image contrast is too low |
| EError_MotherAlreadyHasThisDaughter | Mother already has this daughter - Detach daughter first |
| EError_ShapeAlreadyHasDaughters | Shape already has daughters - Detach daughters first |
| EError_NoValidPointFound | No valid point found in the transition computation. |
| EError_NotInListAttachmentMode | Not in list attachment mode - Detach daughters first |
| EError_NotInIndexedAttachmentMode | Not in indexed attachment mode - Detach daughters and call **SetIndexed** first |
| EError_UnsupportedShapeVersion | Unsupported shape version - Upgrade Open eVision |
| EError_RawCalibrationMode | Raw calibration mode - Cannot be used for this operation |
| EError_BadLandmarkLayout | The layout of supplied landmarks makes the calibration impossible - Check landmarks positions |

| | |
|---|---|
| EError_IncompatibleCalibrationModes | Incompatible calibration modes - Check calibration mode categories |
| EError_NotEnoughLandmarks | Not enough landmarks to calibrate - Add landmarks or check calibration mode categories |
| EError_UnexpectedShapeTypeInFile | Unexpected shape type in file - Check target shape against file model root |
| EError_UnsupportedModelFileVersion | Unsupported model file version - Upgrade Open eVision |
| EError_CannotAttachDetachWorldShapes | Cannot Attach or Detach World shape - World shapes never have a mother |
| EError_UnexpectedWorldShapeInFile | Unexpected World Shape in file - Check target shape against file model root |
| EError_UnexpectedFrameShapeInFile | Unexpected Frame Shape in file - Check target shape against file model root |
| EError_UnexpectedPointShapeInFile | Unexpected Point Shape in file - Check target shape against file model root |
| EError_UnexpectedLineShapeInFile | Unexpected Line Shape in file - Check target shape against file model root |
| EError_UnexpectedCircleShapeInFile | Unexpected Circle Shape in file - Check target shape against file model root |
| EError_UnexpectedRectangleShapeInFile | Unexpected Rectangle Shape in file - Check target shape against file model root |
| EError_UnexpectedWedgeShapeInFile | |

| | |
|---|---|
| EError_UnexpectedPointGaugeInFile | Unexpected Wedge Shape in file - Check target shape against file model root<br>Point Gauge in file - Check target shape against file |
| EError_UnexpectedLineGaugeInFile | Unexpected Line Gauge in file - Check target shape against file model root |
| EError_UnexpectedCircleGaugeInFile | Unexpected Circle Gauge in file - Check target shape against file model root |
| EError_UnexpectedRectangleGaugeInFile | Unexpected Rectangle Gauge in file - Check target shape against file model root |
| EError_UnexpectedWedgeGaugeInFile | Unexpected Wedge Gauge in file - Check target shape against file model root |
| EError_UnexpectedBarCodeInFile | Unexpected Bar Code model in file - Check target shape against file model root |
| EError_AnActiveCurvedEdgeIsRequired | At least one curved edge must be active - Activate r and/or R edges |
| EError_BrokenWedgeShapeConstraints | Constraints between the geometric and the tolerance of the wedge are broken |
| EError_InvalidGrid | The detected grid is invalid |
| EError_InvalidSymbolSize | The detected symbol size is invalid |
| EError_InvalidFixedPattern | The fixed pattern of the detected code is invalid |
| EError_CalibrationModelNotDefined | A 3D calibration model is required to perform the conversion |
| EError_InvalidE3DModelFile | The file-type and structure could not be verified |

| | |
|---|---|
| EError_EmptyPointCloud | The point cloud should not be empty |
| EError_WrongOrientationVector | The supplied orientation vector is not correct |
| EError_InvalidE3DCalibrationGeneratorFile | The file-type and structure could not be verified |
| EError_CalibrationModelNotInitialized | A 3D calibration model is not Initialized |
| EError_InvalidE3DConverterFile | The file-type and structure of the E3D converter file could not be verified |
| EError_InvalidE3DCalibrationFile | The file-type and structure of the E3D calibration file could not be verified |
| EError_UnknownCalibrationObjectType | The calibration object type is not set |
| EError_ResultOutOfTolerances | Result out of tolerances |
| EError_MalformedTriangleIndexes | The triangle indexes are not correct in E3DObject |
| EError_FitFailed | The 3D fit operation failed |
| EError_ParamNotSet | Trying to get a parameter value that has not been set |
| EError_UndefinedPixelValue | The pixel has undefined value |
| EError_FindFailed | The 3D find operation failed |
| EError_AlignFailed | The 3D align operation failed |
| EError_WrongCalibrationParameters | Wrong calibration parameters |

| | |
|---|---|
| EError_WrongNormalVector | Wrong normal vector |
| EError_WrongNormalTolerance | Wrong normal angle tolerance |
| EError_CalibrationModelNotFound | A 3D calibration model is not found |
| EError_AxesNotNormal | The axis system is not normed |
| EError_AxesNotRightHanded | The axis system is not righthanded |
| EError_AxesNotOrthogonal | The axis system is not orthogonal |
| EError_MatrixNotRigid | A matrix is not rigid |
| EError_AxisSystemNotRigid | An axis system is not rigid |
| EError_CoordinatesOutOfMap | The coordinates are out of the map |
| EError_InvalidAxisSystem | Axis system is invalid |
| EError_InvalidPointCloud | The point cloud is not valid |
| EError_PointCloudOutOfRange | The point cloud is out of range |
| EError_ DepthMapNotCompatibleWithCalibration | The depth map point is not compatible with the calibration model (wrong association) |
| EError_InvalidE3DObjectFile | The file-type and structure of the E3D object file could not be verified |
| EError_ Map3DConversionModeMustBeInitialized | Conversion mode must be initialized |
| EError_InvalidE3DBoxFile | |

| | |
|---|---|
| EError_DataAugmentationFailed | The file-type and structure of the E3DBox file could not be verified |

tation failed for an unknown reason.

| | |
|---|---|
| EError_InvalidInputSpecification | Invalid input specification. |
| EError_LabelDoesNotExist | The label does not exist in the dataset. |
| EError_ImageIsNotAPath | The image was not given as a path. |
| EError_ImageDoesNotConformToInputFormat | The images do not conform to the input format of the classifier and the automatic reformat is disabled. |
| EError_CannotDisableAutoreshape | The automatic procedure to make every image conform to the input specification cannot be disabled because there already are images in the dataset that do not conform to the input specification. |
| EError_NoEnoughImagesToSplitDataset | There are not enough images in the dataset to perform a split such that each part has at least one image from each label of the original dataset. |
| EError_NotAvailableIn32Bits | This method is not available for the 32 bits binaries of Open eVision. |
| EError_DatasetIncompatibleWithClassifier | The dataset is incompatible with this classifier. A pre-trained classifier comes with some mandatory input specifications. |
| EError_ClassifierCurrentlyTraining | This operation is impossible because the classifier is currently training. |
| EError_EClassifierNotTraining | Cannot wait because the classifier is not training. |
| EError_CannotChangeInputSpecification | A pre-trained classifier comes with some input specifications that can't be changed. |

| | |
|---|---|
| EError_ TrainingAndValidationDatasetIncompatible | The training and validation dataset have incompatible image format. |
| EError_ TrainingAndValidationLabelsIncompatible | The training and validation dataset have incompatible labels. |
| EError_ ClassifierTrainedWithIncompatibleLabels | The classifier was previously trained with incompatible labels. |
| EError_CannotChangeClassifierType | The type of classifier can't be changed once the classifier is trained. |
| EError_UnknownClassifierType | Unknown classifier type. |
| EError_ClassifierNotTrained | The classifier is not trained. |
| EError_NoGPUFound | No GPU was found. |
| EError_InvalidMetrics | The classification metrics are not valid. |
| EError_MetricsIncompatibleWithResult | The classification metrics are incompatible with the given result. |
| EError_InvalidClassificationResult | The classification result is invalid. |
| EError_HeatmapGenerationFailure | Can not generate Heatmap. |
| EError_NotEnoughMemoryForTraining | There is not enough free memory on the CPU or GPU to perform the training. |
| EError_NotEnoughMemoryForPrediction | There is not enough free memory on the CPU or GPU to perform a prediction. |
| EError_NotEnoughMemoryForBatchPrediction | There is not enough free memory on the CPU or GPU to perform a batch prediction. |
| | |

| | |
|---|---|
| EError_LayerOutputDisabled | The layer has its output disabled. |
| EError_NotEnoughMemoryForCache | There is not enough free memory on the CPU for storing the dataset images in the cache. |
| EError_FlexnetHandleInitializationFailed | Licensing handle initialization failed |
| EError_FlexnetLoadingActivationLibraryFailed | Loading of the activation library failed |
| EError_ FlexnetInitializationActivationLibraryFailed | Initialization of activation library failed |
| EError_FlexnetActivationLibraryMismatch | Activation library mismatch |
| EError_FlexnetActivationLibraryUnloaded | Activation library component has been unloaded |
| EError_FlexnetLicensingServiceNotInstalled | The licensing service is not installed |
| EError_FlexnetNotEnoughRights | Not enough rights to talk to service |
| EError_FlexnetLicenseJobCreationFailed | License job creation failed |
| EError_FLEXnetLicensePromptForFileFailed | Unable to disable license finder dialog |
| EError_PixelHandling | Internal error during image processing |
| EError_EmptyMorphologicalKernel | Use of a morphological kernel without any element set |
| EError_MatrixOperation | Internal error during matrix processing |
| EError_NonSquareMatrix | The operation is only valid for square matrices |
| EError_IncompatibleMatrixSizes | The sizes of the matrix are incompatible for the operation |

| | |
|---|---|
| EError_UnderdeterminedMatrix | Unsupported operation: The matrix has less rows than columns |
| EError_OverdeterminedMatrix | Unsupported operation: The matrix has more rows than columns |
| EError_PointAtInfinity | Unable to apply this operation to points at infinity |
| EError_NotEnoughCalibrationPoints | Not enough points for the calibration process to succeed |
| EError_LineAtInfinity | Unable to apply this operation to lines at infinity |
| EError_UndeterminedGeometricEntity | Undetermined geometric entity in projective geometry |
| EError_NotANumber | Not a number |
| EError_MetadataAlreadyExists | Metadata already exists in the metadata store |
| EError_MetadataDoesNotExist | Metadata does not exist in the metadata store |
| EError_InternalError_000 | Internal error 0 |
| EError_InternalError_001 | Internal error 1 |
| EError_InternalError_002 | Internal error 2 |
| EError_InternalError_003 | Internal error 3 |
| EError_InternalError_004 | Internal error 4 |
| EError_InternalError_005 | Internal error 5 |
| EError_InternalError_006 | Internal error 6 |
| EError_InternalError_007 | Internal error 7 |

| | |
|---|---|
| EError_InternalError_008 | Internal error 8 |
| EError_InternalError_009 | Internal error 9 |
| EError_InternalError_010 | Internal error 10 |
| EError_InternalError_011 | Internal error 11 |
| EError_InternalError_012 | Internal error 12 |
| EError_InternalError_013 | Internal error 13 |
| EError_InternalError_014 | Internal error 14 |
| EError_InternalError_015 | Internal error 15 |
| EError_InternalError_016 | Internal error 16 |
| EError_InternalError_017 | Internal error 17 |
| EError_InternalError_018 | Internal error 18 |
| EError_InternalError_019 | Internal error 19 |
| EError_InternalError_020 | Internal error 20 |
| EError_InternalError_021 | Internal error 21 |
| EError_InternalError_022 | Internal error 22 |
| EError_InternalError_023 | Internal error 23 |
| EError_InternalError_024 | Internal error 24 |

| | |
|---|---|
| EError_InternalError_025 | Internal error 25 |
| EError_InternalError_026 | Internal error 26 |
| EError_InternalError_027 | Internal error 27 |
| EError_InternalError_028 | Internal error 28 |
| EError_InternalError_029 | Internal error 29 |
| EError_InternalError_030 | Internal error 30 |
| EError_InternalError_031 | Internal error 31 |
| EError_InternalError_032 | Internal error 32 |
| EError_InternalError_033 | Internal error 33 |
| EError_InternalError_034 | Internal error 34 |
| EError_InternalError_035 | Internal error 35 |
| EError_InternalError_036 | Internal error 36 |
| EError_InternalError_037 | Internal error 37 |
| EError_InternalError_038 | Internal error 38 |
| EError_InternalError_039 | Internal error 39 |
| EError_InternalError_040 | Internal error 40 |
| EError_InternalError_041 | Internal error 41 |

| | |
|---|---|
| EError_InternalError_042 | Internal error 42 |
| EError_InternalError_043 | Internal error 43 |
| EError_InternalError_044 | Internal error 44 |
| EError_InternalError_045 | Internal error 45 |
| EError_InternalError_046 | Internal error 46 |
| EError_InternalError_047 | Internal error 47 |
| EError_InternalError_048 | Internal error 48 |
| EError_InternalError_049 | Internal error 49 |
| EError_InternalError_050 | Internal error 50 |
| EError_InternalError_051 | Internal error 51 |
| EError_InternalError_052 | Internal error 52 |
| EError_InternalError_053 | Internal error 53 |
| EError_InternalError_054 | Internal error 54 |
| EError_InternalError_055 | Internal error 55 |
| EError_InternalError_056 | Internal error 56 |
| EError_InternalError_057 | Internal error 57 |
| EError_InternalError_058 | Internal error 58 |

| | |
|---|---|
| EError_InternalError_059 | Internal error 59 |
| EError_InternalError_060 | Internal error 60 |
| EError_InternalError_061 | Internal error 61 |
| EError_InternalError_062 | Internal error 62 |
| EError_InternalError_063 | Internal error 63 |
| EError_InternalError_064 | Internal error 64 |
| EError_InternalError_065 | Internal error 65 |
| EError_InternalError_066 | Internal error 66 |
| EError_InternalError_067 | Internal error 67 |
| EError_InternalError_068 | Internal error 68 |
| EError_InternalError_069 | Internal error 69 |
| EError_InternalError_070 | Internal error 70 |
| EError_InternalError_071 | Internal error 71 |
| EError_InternalError_072 | Internal error 72 |
| EError_InternalError_073 | Internal error 73 |
| EError_InternalError_074 | Internal error 74 |
| EError_InternalError_075 | Internal error 75 |

| | |
|---|---|
| EError_InternalError_076 | Internal error 76 |
| EError_InternalError_077 | Internal error 77 |
| EError_InternalError_078 | Internal error 78 |
| EError_InternalError_079 | Internal error 79 |
| EError_InternalError_080 | Internal error 80 |
| EError_InternalError_081 | Internal error 81 |
| EError_InternalError_082 | Internal error 82 |
| EError_InternalError_083 | Internal error 83 |
| EError_InternalError_084 | Internal error 84 |
| EError_InternalError_085 | Internal error 85 |
| EError_InternalError_086 | Internal error 86 |
| EError_InternalError_087 | Internal error 87 |
| EError_InternalError_088 | Internal error 88 |
| EError_InternalError_089 | Internal error 89 |
| EError_InternalError_090 | Internal error 90 |
| EError_InternalError_091 | Internal error 91 |
| EError_InternalError_092 | Internal error 92 |

| | |
|---|---|
| EError_InternalError_093 | Internal error 93 |
| EError_InternalError_094 | Internal error 94 |
| EError_InternalError_095 | Internal error 95 |
| EError_InternalError_096 | Internal error 96 |
| EError_InternalError_097 | Internal error 97 |
| EError_InternalError_098 | Internal error 98 |
| EError_InternalError_099 | Internal error 99 |
| EError_InternalError_100 | Internal error 100 |
| EError_CannotTraceErrors | Cannot trace errors because of a system failure |
| EError_NotImplemented | Feature not implemented |
| EError_NullPointer | The supplied pointer is NULL |
| EError_InvalidTimeout | The current timeout value is 0 |
| EError_InvalidTimeoutReentrancy | Cannot Stop a timeout that has not been started. Cannot Pop a timeout that has not been pushed |
| EError_InvalidTimeoutState | Cannot Start a timeout that has been reached Cannot Pop a timeout that is Active |
| EError_Unknown | Unknown error |

# 6.39. EFamily Enum

Allowed values for the ECC symbol family in EasyMatrixCode.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EFamily_ECC000 | ECC 000, no error recovery capability by convolutional coding. |
| EFamily_ECC050 | ECC 050, 2.8 % error recovery capability by convolutional coding. |
| EFamily_ECC080 | ECC 080, 5.5 % error recovery capability by convolutional coding. |
| EFamily_ECC100 | ECC 100, 12.6 % error recovery capability by convolutional coding. |
| EFamily_ECC140 | ECC 140, 25 % error recovery capability by convolutional coding. |
| EFamily_ECC200 | ECC 200, 20 % error recovery capability. |
| EFamily_Unknown | - |

### Remarks

This enumeration is in the **Euresys::Open eVision** namespace.

# 6.40. EFeature Enum

The various features that can be measured on the coded elements of a selection.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EFeature_ElementIndex | Index of the coded element (cf. ECodedElement::ElementIndex). |
| | |

| | |
|---|---|
| EFeature_LayerIndex | Index of the layer of the coded element (cf. ECodedElement::LayerIndex). |
| EFeature_RunCount | Number of runs (cf. ECodedElement::RunCount). |
| EFeature_Area | Number of pixels (cf. ECodedElement::Area). |
| EFeature_LargestRun | Length of the largest run (cf. ECodedElement::LargestRun). |
| EFeature_ContourX | Starting point abscissa of the contour of the coded element (cf. ECodedElement::ContourX). |
| EFeature_ContourY | Starting point ordinate of the countour of the coded element (cf. ECodedElement::ContourY). |
| EFeature_LeftLimit | Abscissa of the leftmost pixel (cf. ECodedElement::LeftLimit). |
| EFeature_RightLimit | Abscissa of the rightmost pixel (cf. ECodedElement::RightLimit). |
| EFeature_TopLimit | Abscissa of the topmost pixel (cf. ECodedElement::TopLimit). |
| EFeature_BottomLimit | Ordinate of the bottommost pixel (cf. ECodedElement::BottomLimit). |
| EFeature_GravityCenterX | Abscissa of the gravity center (cf. ECodedElement::GravityCenterX). |
| EFeature_GravityCenterY | Ordinate of the gravity center (cf. ECodedElement::GravityCenterY). |
| EFeature_BoundingBoxCenterX | Abscissa of the center of the bounding box (cf. ECodedElement::BoundingBoxCenterX). |
| EFeature_BoundingBoxCenterY | Ordinate of the center of the bounding box (cf. ECodedElement::BoundingBoxCenterY). |

| | |
|---|---|
| EFeature_BoundingBoxWidth | Width of the bounding box (Feret diameter 0° - cf. ECodedElement::BoundingBoxWidth). |
| EFeature_BoundingBoxHeight | Height of the bounding box (Feret diameter 90° - cf. ECodedElement::BoundingBoxHeight). |
| EFeature_FeretBox22CenterX | Abscissa of the center of the Feret box oriented at 22.5° (cf. ECodedElement::FeretBox22CenterX). |
| EFeature_FeretBox22CenterY | Ordinate of the center of the Feret box oriented at 22.5° (cf. ECodedElement::FeretBox22CenterY). |
| EFeature_FeretBox22Width | Width of the Feret box oriented at 22.5° (Feret diameter at 22.5° - cf. ECodedElement::FeretBox22Width). |
| EFeature_FeretBox22Height | Height of the Feret box oriented at 22.5° (Feret diameter at 112.5° - cf. ECodedElement::FeretBox22Height). |
| EFeature_FeretBox45CenterX | Abscissa of the center of the Feret box oriented at 45° (cf. ECodedElement::FeretBox45CenterX). |
| EFeature_FeretBox45CenterY | Ordinate of the center of the Feret box oriented at 45° (cf. ECodedElement::FeretBox45CenterY). |
| EFeature_FeretBox45Width | Width of the Feret box oriented at 45° bounding box (Feret diameter at 45° - cf. ECodedElement::FeretBox45Width). |
| EFeature_FeretBox45Height | Height of the Feret box oriented at 45° (Feret diameter at 135° - cf. ECodedElement::FeretBox45Height). |
| EFeature_FeretBox68CenterX | Abscissa of the center of the Feret box oriented at 67.5° (cf. ECodedElement::FeretBox68CenterX). |
| EFeature_FeretBox68CenterY | Ordinate of the center of the Feret box oriented at 67.5° (cf. ECodedElement::FeretBox68CenterY). |
| | |

| | |
|---|---|
| EFeature_FeretBox68Width | Width of the Feret box oriented at 67.5° (Feret diameter at 67.5° - cf. ECodedElement::FeretBox68Width). |
| EFeature_FeretBox68Height | Height of the Feret box oriented at 67.5° (Feret diameter at 157.5° - cf. ECodedElement::FeretBox68Height). |
| EFeature_ MinimumEnclosingRectangleCenterX | Abscissa of the Minimum Enclosing Rectangle center (cf. ECodedElement::MinimumEnclosingRectangleCenterX). |
| EFeature_MinimumEnclosingRectangleCenterY | Ordinate of the Minimum Enclosing Rectangle center (cf. ECodedElement::MinimumEnclosingRectangleCenterY). |
| EFeature_MinimumEnclosingRectangleWidth | Width of the Minimum Enclosing Rectangle (cf. ECodedElement::MinimumEnclosingRectangleWidth). |
| EFeature_MinimumEnclosingRectangleHeight | Height of the Minimum Enclosing Rectangle (cf. ECodedElement::MinimumEnclosingRectangleHeight). |
| EFeature_MinimumEnclosingRectangleAngle | Direction of the Minimum Enclosing Rectangle (cf. ECodedElement::MinimumEnclosingRectangleAngle). |
| EFeature_SigmaX | Centered moment of inertia around X (average squared X-deviation - cf. ECodedElement::SigmaX). |
| EFeature_SigmaY | Centered moment of inertia around Y (average squared Y-deviation - cf. ECodedElement::SigmaY). |
| EFeature_SigmaXX | Reduced, centered moment of inertia (around the principal inertia axis - cf. ECodedElement::SigmaXX). |
| EFeature_SigmaXY | Centered cross moment of inertia (average X-deviation * Y-deviation - cf. ECodedElement::SigmaXY). |
| EFeature_SigmaYY | Reduced, centered moment of inertia (around the secondary inertia axis - cf. ECodedElement::SigmaYY). |
| | |

| | |
|---|---|
| EFeature_EllipseWidth<br><br>EFeature_EllipseHeight | Long axis of the ellipse of inertia (cf. ECodedElement::EllipseWidth).<br><br>Short axis of the ellipse of inertia (cf. ECodedElement::EllipseHeight). |
| EFeature_EllipseAngle | Angle of the principal axis of the ellipse of inertia (cf. ECodedElement::EllipseAngle). |
| EFeature_Eccentricity | Eccentricity of the ellipse of inertia (cf. ECodedElement::Eccentricity). |
| EFeature_FeretBoxCenterX | Abscissa of the center of the Feret box oriented at a fixed angle (cf. ECodedElement::ComputeFeretBox). The angle of interest is set through EObjectSelection::FeretAngle. |
| EFeature_FeretBoxCenterY | Ordinate of the center of the Feret box oriented at a fixed angle (cf. ECodedElement::ComputeFeretBox). The angle of interest is set through EObjectSelection::FeretAngle. |
| EFeature_FeretBoxWidth | Width of the Feret box oriented at a fixed angle (cf. ECodedElement::ComputeFeretBox). The angle of interest is set through EObjectSelection::FeretAngle. |
| EFeature_FeretBoxHeight | Height of the Feret box oriented at a fixed angle (cf. ECodedElement::ComputeFeretBox). The angle of interest is set through EObjectSelection::FeretAngle. |
| EFeature_PixelMin | Minimum gray level of the pixels of the attached image over the coded element (cf. ECodedElement::ComputePixelMin). The attached image is set through EObjectSelection::AttachedImage. |
| EFeature_PixelMax | Maximum gray level of the pixels of the attached image over the coded element (cf. ECodedElement::ComputePixelMax). The attached image is set through EObjectSelection::AttachedImage. |
| EFeature_WeightedGravityCenterX | |

| | |
|---|---|
| EFeature_WeightedGravityCenterY<br><br>he gravity center of the pixels of the attached image<br>ed element (cf.<br>ent::ComputeWeightedGravityCenter). The attached<br>hrough EObjectSelection::AttachedImage. | Abscissa of the gravity center of the pixels of the attached image over the coded element (cf. ECodedElement::ComputeWeightedGravityCenter). The attached image is set through EObjectSelection::AttachedImage. |
| EFeature_PixelGrayAverage | Average gray-level value of the attached image over the coded element (cf. ECodedElement::ComputePixelGrayAverage). The attached image is set through EObjectSelection::AttachedImage. |
| EFeature_PixelGrayVariance | Variance of the gray-level value of the attached image over the coded element (cf. ECodedElement::ComputePixelGrayVariance). The attached image is set through EObjectSelection::AttachedImage. |
| EFeature_PixelGrayDeviation | Standard deviation of the gray-level value of the attached image over the coded element (cf. ECodedElement::ComputePixelGrayDeviation). The attached image is set through EObjectSelection::AttachedImage. |

# 6.41. EFillUndefinedPixelsDirection Enum

Direction in which the undefined pixels are filled in a depthmap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

| | |
|---|---|
| EFillUndefinedPixelsDirection_Vertical | Undefined pixels are filled using a vertical scan. |
| EFillUndefinedPixelsDirection_Horizontal | Undefined pixels are filled using an horizontal scan. |
| EFillUndefinedPixelsDirection_Combined | Undefined pixels are filled using both a vertical and an horizontal scan. |
| | |

| | |
|---|---|
| EFillUndefinedPixelsDirection_Local | Specialized method for filling undefined pixels. Undefined pixels are filled using their 4 neighboring pixels: If at least 2 out of 4 neighbors have a 'defined' value, the pixel will be filled with their average value. Else, the pixel will remain 'undefined'. |

# 42. EFillUndefinedPixelsMethod Enum

Method to fill the undefined pixels in a depthmap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

| | |
|---|---|
| EFillUndefinedPixelsMethod_KeepMinimum | Undefined pixels are filled using the minimum of their neighbours. |
| EFillUndefinedPixelsMethod_KeepMaximum | Undefined pixels are filled using the maximum of their neighbours. |
| EFillUndefinedPixelsMethod_Average | Undefined pixels are filled using the average of their neighbours. |
| EFillUndefinedPixelsMethod_Ramp | Undefined pixels are filled using a ramp between their neighbours. |

# 6.43. EFilteringMode Enum

Allowed values for the filtering mode of EasyMatch.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EFilteringMode_Uniform | Filtering with a uniform 2x2 kernel. This is the preferred mode for natural images. Default mode. |
| | |

| | |
|---|---|
| EFilteringMode_LowPass | Filtering with a low-pass 3x3 kernel. This is the preferred mode for images featuring sharp gray-level transitions. |

## 6.44. EFindContrastMode Enum

Allowed values for the contrast mode of EasyMatch.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EFindContrastMode_Normal | Accepts instances with normal contrast (default mode). |
| EFindContrastMode_Inverse | Accepts instances with reversed contrast. |
| EFindContrastMode_Any | Accepts instances with normal and/or reversed contrast. |
| EFindContrastMode_PointByPointNormal | - |
| EFindContrastMode_PointByPointInverse | - |
| EFindContrastMode_PointByPointAny | - |
| EFindContrastMode_Unknown | - |

## 6.45. EFlipping Enum

Allowed values for the symbol flipping type in EasyMatrixCode.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EFlipping_Yes | Image is flipped. |
| EFlipping_No | Image is not flipped. |
| EFlipping_Unknown | To be determined at **Read** or **Learn** time. |

# 6.46. EFramePosition Enum

This enumeration contains the possible values for the placement of the overlay frame edges that are drawn to highlight the position of an ROI.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EFramePosition_On | The frame is centered on the ROI edges. |
| EFramePosition_Inside | The outer edges of the frame remain totally inside the ROI. |
| EFramePosition_Outside | The inner edges of the frame remain totally outside the ROI. |

# 6.47. EGrayscaleSingleThreshold Enum

The modes that are available to segment a grayscale image using a single threshold.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EGrayscaleSingleThreshold_Absolute | Thresholds the image against a fixed, absolute gray level. The threshold value is fixed through EGrayscaleSingleThresholdSegmenter::AbsoluteThreshold. |
| EGrayscaleSingleThreshold_Relative | |

| | |
|---|---|
| EGrayscaleSingleThreshold_MinResidue<br><br>he image using an automatically-computed value<br>quadratic difference between the source and<br>mage is minimized. | Thresholds the image against a relative gray level: The actual threshold is selected so that a given fraction of the pixels of the image lie below it. The fraction is fixed through EGrayscaleSingleThresholdSegmenter::RelativeThreshold. |
| EGrayscaleSingleThreshold_MaxEntropy | Thresholds the image using an automatically-computed value such that the entropy (i.e. the amount of information) of the resulting thresholded image is maximized. |
| EGrayscaleSingleThreshold_IsoData | Thresholds the image using an automatically-computed value halfway between the average dark gray value (i.e. gray levels below the threshold) and average light gray values (i.e. gray levels above the threshold). |

# 6.48. EHarrisThresholdingMode Enum

The thresholding modes for the Harris corner detector.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EHarrisThresholdingMode_Relative | Relative thresholding mode. |
| EHarrisThresholdingMode_Absolute | Absolute thresholding mode. |

# 6.49. EHistogramFeature Enum

The various parameters that can be extracted from a histogram.

**Namespace:** Euresys::Open_eVision_2_10

| |
|---|

| | |
|---|---|
| EHistogramFeature_MostFrequentPixelValue | Value of the most frequent pixel. |
| EHistogramFeature_ MostFrequentPixelFrequency | Frequency of the most frequent pixel. |
| EHistogramFeature_LeastFrequentPixelValue | Value of the least frequent pixel. |
| EHistogramFeature_ LeastFrequentPixelFrequency | Frequency of the least frequent pixel. |
| EHistogramFeature_SmallestPixelValue | Smallest pixel value. |
| EHistogramFeature_GreatestPixelValue | Largest pixel value. |
| EHistogramFeature_PixelCount | Number of pixels. |
| EHistogramFeature_AveragePixelValue | Mean of the pixel values. |
| EHistogramFeature_PixelValueStdDev | Standard deviation of the pixel values. |

# 6.50. EHitAndMissValue Enum

The allowed values for the elements of a hit-and-miss kernel.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EHitAndMissValue_Background | The element belongs to the background. |
| EHitAndMissValue_DontCare | The element does not belongs to the background, neither to the foreground. It is ignored by the kernel. |
| EHitAndMissValue_Foreground | The element belongs to the foreground. |

# 6.51. EImageFileType Enum

-

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EImageFileType_Bmp | - |
| EImageFileType_Jpeg2000 | - |
| EImageFileType_Jpeg | - |
| EImageFileType_Png | - |
| EImageFileType_Tiff | - |
| EImageFileType_Auto | - |
| EImageFileType_Euresys | - |
| EImageFileType_Unknown | - |

# 6.52. EImageType Enum

Image type.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EImageType_BW1 | Bi-level image. |
| | |

| | |
|---|---|
| EImageType_BW8 | 8 bits per pixel gray-level image. |
| EImageType_BW16 | 16 bits per pixel gray-level image |
| EImageType_BW32 | 32 bits per pixel gray-level image. |
| EImageType_C15 | 15 bits per pixel color image (R:5, G:5, B:5). |
| EImageType_C16 | 16 bits per pixel color image (R:5, G:6, B:5). |
| EImageType_C24 | 24 bits per pixel color image (RGB). |
| EImageType_C24A | 32 bits per pixel color image (RGB + unused alpha channel). |
| EImageType_C48 | 48 bits per pixel color image (RGB). |
| EImageType_Depth8 | 8 bits per pixel gray-level image. |
| EImageType_Depth16 | 16 bits per pixel gray-level image. |
| EImageType_Depth32f | 32 bits per pixel gray-level image. |

### Remarks

For example, an EImageC24 has type value EImageType_C24 and its pixels are typed as EC24.

# 6.53. EKernelRectifier Enum

Possible values for the rectification mode of a kernel. This property allows specifying how negative convolution result values are handled.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EKernelRectifier_DoNotRectify | The offset of the kernel is added to the values resulting from the convolution. Negative values are then set to zero, and the values that exceed the maximum value for the image type are set to this maximum value. |
| EKernelRectifier_KeepNegative | |
| EKernelRectifier_KeepPositive | Positive value is used. The negative values are discarded (set to zero). |
| EKernelRectifier_Absolute | The absolute value is used. |

values are discarded (set to zero) and the magnitude
ue) of the negative values is used.

# 6.54. EKernelRotation Enum

Possible values for rotating a convolution kernel.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EKernelRotation_NoRotation | No rotation of the structuring element. |
| EKernelRotation_Clockwise | Clockwise rotation (one full turn per pass). |
| EKernelRotation_Anticlockwise | Counterclockwise rotation (one full turn per pass). |

# 6.55. EKernelType Enum

The types of convolution kernels that are supported by Open eVision.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EKernelType_WhiteSkelet | White skeleton morphological probe. |
| EKernelType_BlackSkelet | Black skeleton morphological probe. |
| EKernelType_Edge | Edge detection morphological probe. |
| EKernelType_SobelX | X-axis Sobel derivative. |
| EKernelType_SobelY | Y-axis Sobel derivative. |
| EKernelType_PrewittX | X-axis Prewitt derivative. |
| EKernelType_PrewittY | Y-axis Prewitt derivative. |
| EKernelType_Laplacian4 | 4-connected Laplacian. |
| EKernelType_Laplacian8 | 8-connected Laplacian. |
| EKernelType_LowPass1 | Low pass filter. |
| EKernelType_LowPass2 | Low pass filter (average of neighbors). |
| EKernelType_LowPass3 | Low pass filter (average). |
| EKernelType_HighPass1 | High pass filter (value plus 4-connected Laplacian). |
| EKernelType_HighPass2 | High pass filter (value plus 8-connected Laplacian). |
| EKernelType_Sobel | - |
| EKernelType_Prewitt | - |
| EKernelType_Roberts | - |

| | |
|---|---|
| EKernelType_Uniform3x3 | - |
| EKernelType_Gaussian3x3 | - |
| EKernelType_Uniform5x5 | - |
| EKernelType_Gaussian5x5 | - |
| EKernelType_Gaussian7x7 | - |
| EKernelType_Uniform7x7 | - |
| EKernelType_LaplacianX | - |
| EKernelType_LaplacianY | - |
| EKernelType_Gradient | - |
| EKernelType_GradientX | - |
| EKernelType_GradientY | - |
| EKernelType_Uniform | - |
| EKernelType_Gaussian | - |

# 6.56. ELearningMode Enum

Allowed values for the learning mode in EasyOCV.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ELearningMode_Reset | Restart learning. |
| ELearningMode_Template | Train the mother image. |
| ELearningMode_Average | Accumulate an image for average estimation. |
| ELearningMode_RmsDeviation | Accumulate an image for standard deviation estimation. |
| ELearningMode_AbsDeviation | Accumulate an image for robust deviation estimation. |
| ELearningMode_Ready | End learning and compute threshold images. |

# 6.57. ELearnParam Enum

Allowed values for the kind of parameters that can be learnt by EasyMatrixCode.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ELearnParam_LogicalSize | The data matrix code symbol logical sizes the candidate is matched against at read time. |
| ELearnParam_ContrastType | The data matrix code contrast types the candidate is matched against at read time. |
| ELearnParam_Flipping | The data matrix code flipping values the candidate is matched against at read time. |
| ELearnParam_Family | The data matrix code families the candidate is matched against at read time. |
| | |

| | |
|---|---|
| ELearnParam_NumItems | - |

6-

.-

# 58. ELegacyFeature Enum

The various parameters that can be extracted from a histogram. This enumeration pertains to the EasyObject legacy API. Please use ECodedImage2 instead.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ELegacyFeature_NoFeature | - |
| ELegacyFeature_Class | Class number. |
| ELegacyFeature_RunsNumber | Number of runs. |
| ELegacyFeature_Area | Number of pixels. *(Signed Integer)*. |
| ELegacyFeature_LargestRun | Size of the longest run. *(Signed Integer)*. |
| ELegacyFeature_GravityCenterX | Abscissa of the gravity center. *(Float)*. (*) |
| ELegacyFeature_GravityCenterY | Ordinate of the gravity center. *(Float)*. (*) |
| ELegacyFeature_LimitCenterX | Abscissa of the center of the bounding box. *(Float)*. (*) |
| ELegacyFeature_LimitCenterY | Ordinate of the center of the bounding box. *(Float)*. (*) |
| ELegacyFeature_LimitWidth | Width of the bounding box (Feret's diameter 0°). *(Float)*. (*) |
| | |

| | |
|---|---|
| ELegacyFeature_LimitHeight | Height of the bounding box (Feret's diameter 90°). *(Float)*. (*) |
| ELegacyFeature_Limit45CenterX | Abscissa of the center of the 45° bounding box. *(Float)*. (*) |
| ELegacyFeature_Limit45CenterY | Ordinate of the center of the 45° bounding box. *(Float)*. (*) |
| ELegacyFeature_Limit45Width | Width of the 45° bounding box (Feret's diameter 45°). *(Float)*. (*) |
| ELegacyFeature_Limit45Height | Height of the 45° bounding box (Feret's diameter 135°). *(Float)*. (*) |
| ELegacyFeature_ContourX | Starting point abscissa of the object contour. *(Signed Integer)*. |
| ELegacyFeature_ContourY | Starting point ordinate of the object contour. *(Signed Integer)*. |
| ELegacyFeature_PixelMin | Minimum gray level of all pixels. *(Signed Integer)*. |
| ELegacyFeature_PixelMax | Maximum gray level of all pixels. *(Signed Integer)*. |
| ELegacyFeature_SigmaX | Centered moment of inertia around X (average squared X-deviation). *(Float)*. |
| ELegacyFeature_SigmaY | Centered moment of inertia around Y (average squared Y-deviation). *(Float)*. |
| ELegacyFeature_SigmaXY | Centered cross moment of inertia (average X-deviation * Y-deviation). *(Float)*. |
| ELegacyFeature_SigmaXX | Reduced, centered moment of inertia (around the principal inertia axis). *(Float)*. |
| ELegacyFeature_SigmaYY | Reduced, centered moment of inertia (around the secondary inertia axis). *(Float).* |
| ELegacyFeature_EllipseWidth | Long axis of the ellipse of inertia. *(Float)*. (*) |

| | |
|---|---|
| ELegacyFeature_EllipseHeight | Short axis of the ellipse of inertia. *(Float)*. (*) |
| ELegacyFeature_EllipseAngle | Direction of the principal axis of inertia. *(Float)*. (*) |
| ELegacyFeature_CentroidX | Abscissa of the weighted gravity center. *(Float)*. (*) |
| ELegacyFeature_CentroidY | Ordinate of the weighted gravity center. *(Float)*. (*) |
| ELegacyFeature_PixelGrayAverage | Average gray-level value of the object pixels. *(Float)*. |
| ELegacyFeature_PixelGrayVariance | Variance of the gray-level value of the object pixels. *(Float)*. |
| ELegacyFeature_Limit22CenterX | Abscissa of the center of the 22.5° bounding box. *(Float)*. (*) |
| ELegacyFeature_Limit22CenterY | Ordinate of the center of the 22.5° bounding box. *(Float)*. (*) |
| ELegacyFeature_Limit22Width | Width of the 22.5° bounding box (Feret's diameter 22.5°). *(Float)*. (*) |
| ELegacyFeature_Limit22Height | Height the 22.5° bounding box (Feret's diameter 112.5°). *(Float)*. (*) |
| ELegacyFeature_Limit68CenterX | Abscissa of the center of the 67.5° bounding box. *(Float)*. (*) |
| ELegacyFeature_Limit68CenterY | Ordinate of the center of the 67.5° bounding box. *(Float)*. (*) |
| ELegacyFeature_Limit68Width | Width of the 67.5° bounding box (Feret's diameter 67.5°). *(Float)*. (*) |
| ELegacyFeature_Limit68Height | Height of the 67.5° bounding box (Feret's diameter 157.5°). *(Float)*. (*) |
| ELegacyFeature_LimitAngledCenterX | Abscissa of the center of the bounding box having a skew angle defined by the **LimitAngle** property. *(Float)*. |
| | |

| | |
|---|---|
| ELegacyFeature_LimitAngledCenterY | Ordinate of the center of the bounding box having a skew angle defined by the **LimitAngle** property. *(Float)*. |
| ELegacyFeature_LimitAngledWidth | Width of the bounding box having a skew angle defined by the **LimitAngle** property (Feret's diameter [LimitAngle]). *(Float)*. |
| ELegacyFeature_LimitAngledHeight | Height of the bounding box having a skew angle defined by the **LimitAngle** property (Feret's diameter [LimitAngle + 90°]). *(Float)*. |
| ELegacyFeature_FeretCenterX | Abscissa of the Feret's bounding box center. *(Float)*. (*) |
| ELegacyFeature_FeretCenterY | Ordinate of the Feret's bounding box center. *(Float)*. (*) |
| ELegacyFeature_FeretWidth | Width of the Feret's bounding box. *(Float)*. (*) |
| ELegacyFeature_FeretHeight | Height of the Feret's bounding box. *(Float)*. (*) |
| ELegacyFeature_FeretAngle | Direction of the Feret's bounding box. *(Float)*. (*) |
| ELegacyFeature_ObjectNumber | Identification number. |
| ELegacyFeature_GravityCenter | Abscissa of the gravity center. *(Float)*. (*) |
| ELegacyFeature_Limit | Abscissa of the center of the bounding box. *(Float)*. (*) |
| ELegacyFeature_Limit22 | Abscissa of the center of the 22.5° bounding box. *(Float)*. (*) |
| ELegacyFeature_Limit45 | Abscissa of the center of the 45° bounding box. *(Float)*. (*) |
| ELegacyFeature_Limit68 | Abscissa of the center of the 67.5° bounding box. *(Float)*. (*) |
| ELegacyFeature_LimitAngled | Abscissa of the center of the bounding box having a skew angle defined by the **LimitAngle** property. *(Float)*. |
| | |

| | |
|---|---|
| ELegacyFeature_Ellipse | Long axis of the ellipse of inertia. *(Float)*. (*) |
| ELegacyFeature_Centroid | - |
| ELegacyFeature_Feret | - |

# 6.59. ELocalSearchMode Enum

Allowed values for the local search mode of EasyFind.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ELocalSearchMode_Basic | Default local search neighborhood.<br>Sets EPatternFinder::AngleSearchExtent, EPatternFinder::ScaleSearchExtent, EPatternFinder::XSearchExtent and EPatternFinder::YSearchExtent to 3. |
| ELocalSearchMode_ExtendedTranslation | Local search neighborhood extended on the translation degrees of freedom.<br>Sets EPatternFinder::AngleSearchExtent and EPatternFinder::ScaleSearchExtent to 3. Sets EPatternFinder::XSearchExtent and EPatternFinder::YSearchExtent to 5. |
| ELocalSearchMode_ExtendedAll | Local search neighborhood extended on all degrees of freedom.<br>Sets EPatternFinder::AngleSearchExtent, EPatternFinder::ScaleSearchExtent, EPatternFinder::XSearchExtent and EPatternFinder::YSearchExtent to 5. |
| ELocalSearchMode_ExtendedMore | |

| | |
|---|---|
| ELocalSearchMode_Reserved<br><br>internal use. | Local search neighborhood even more extended on all degrees of freedom.<br>Sets EPatternFinder::AngleSearchExtent and EPatternFinder::ScaleSearchExtent to 7. Sets EPatternFinder::XSearchExtent and EPatternFinder::YSearchExtent to 9. |
| ELocalSearchMode_Custom | Custom local search neighborhood.<br>Set EPatternFinder::AngleSearchExtent, EPatternFinder::ScaleSearchExtent, EPatternFinder::XSearchExtent and EPatternFinder::YSearchExtent to custom values. |

# 6.60. ELocationMode Enum

Allowed values for the kind of pre-processing to be applied to the sample image in EasyOCV.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ELocationMode_Raw | The raw gray-level image is used to locate the characters. |
| ELocationMode_Binarized | The gray-level image is thresholded before location, thus enhancing contrast between the characters and the foreground. |
| ELocationMode_Gradient | The gradient (edge-detection) of the image is operated on. |
| ELocationMode_Laplacian | The Laplacian of the image is operated on. |

## Remarks

Experience reveals that the best location reliability is achieved by the ELocationMode_Binarized and ELocationMode_Gradient modes. Additionally, the ELocationMode_Gradient mode is not sensitive to the choice of a threshold level. Use of the ELocationMode_Laplacian mode is not recommended.

# 6.61. ELogicalSize Enum

Allowed values for the logical size of Data Matrix codes in EasyMatrixCode.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ELogicalSize__9x9 | ECC 000-140 squares. |
| ELogicalSize__11x11 | ECC 000-140 squares. |
| ELogicalSize__13x13 | ECC 000-140 squares. |
| ELogicalSize__15x15 | ECC 000-140 squares. |
| ELogicalSize__17x17 | ECC 000-140 squares. |
| ELogicalSize__19x19 | ECC 000-140 squares. |
| ELogicalSize__21x21 | ECC 000-140 squares. |
| ELogicalSize__23x23 | ECC 000-140 squares. |
| ELogicalSize__25x25 | ECC 000-140 squares. |
| ELogicalSize__27x27 | ECC 000-140 squares. |
| ELogicalSize__29x29 | ECC 000-140 squares. |
| ELogicalSize__31x31 | ECC 000-140 squares. |
| ELogicalSize__33x33 | ECC 000-140 squares. |

| | |
|---|---|
| ELogicalSize__35x35 | ECC 000-140 squares. |
| ELogicalSize__37x37 | ECC 000-140 squares. |
| ELogicalSize__39x39 | ECC 000-140 squares. |
| ELogicalSize__41x41 | ECC 000-140 squares. |
| ELogicalSize__43x43 | ECC 000-140 squares. |
| ELogicalSize__45x45 | ECC 000-140 squares. |
| ELogicalSize__47x47 | ECC 000-140 squares. |
| ELogicalSize__49x49 | ECC 000-140 squares. |
| ELogicalSize__10x10 | ECC 200 squares. |
| ELogicalSize__12x12 | ECC 200 squares. |
| ELogicalSize__14x14 | ECC 200 squares. |
| ELogicalSize__16x16 | ECC 200 squares. |
| ELogicalSize__18x18 | ECC 200 squares. |
| ELogicalSize__20x20 | ECC 200 squares. |
| ELogicalSize__22x22 | ECC 200 squares. |
| ELogicalSize__24x24 | ECC 200 squares. |
| ELogicalSize__26x26 | ECC 200 squares. |

| | |
|---|---|
| ELogicalSize__32x32 | ECC 200 squares. |
| ELogicalSize__36x36 | ECC 200 squares. |
| ELogicalSize__40x40 | ECC 200 squares. |
| ELogicalSize__44x44 | ECC 200 squares. |
| ELogicalSize__48x48 | ECC 200 squares. |
| ELogicalSize__52x52 | ECC 200 squares. |
| ELogicalSize__64x64 | ECC 200 squares. |
| ELogicalSize__72x72 | ECC 200 squares. |
| ELogicalSize__80x80 | ECC 200 squares. |
| ELogicalSize__88x88 | ECC 200 squares. |
| ELogicalSize__96x96 | ECC 200 squares. |
| ELogicalSize__104x104 | ECC 200 squares. |
| ELogicalSize__120x120 | ECC 200 squares. |
| ELogicalSize__132x132 | ECC 200 squares. |
| ELogicalSize__144x144 | ECC 200 squares. |
| ELogicalSize__8x18 | ECC 200 rectangles |
| ELogicalSize__8x32 | ECC 200 rectangles |

| | |
|---|---|
| ELogicalSize__12x26 | ECC 200 rectangles |
| ELogicalSize__12x36 | ECC 200 rectangles |
| ELogicalSize__16x36 | ECC 200 rectangles |
| ELogicalSize__16x48 | ECC 200 rectangles |
| ELogicalSize_Unknown | To be determined at **Read** or **Learn** time. |

### Remarks

This enumeration is in the **Euresys::Open eVision** namespace.

# 6.62. EMailBarcodeOrientation Enum

The orientations supported by EMailBarcode

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EMailBarcodeOrientation_Unknown | Unknown orientation. |
| EMailBarcodeOrientation_NoRotation | Non rotated barcode. Horizontal and read from left to right. |
| EMailBarcodeOrientation_Rotated180 | Upside-down barcode. Horizontal and read from right to left. |
| EMailBarcodeOrientation_Rotated90Right | Barcode rotated 90° to the right. Vertical and read from top to bottom. |
| EMailBarcodeOrientation_Rotated90Left | Barcode rotated 90° to the left. Vertical and read from bottom to top. |
| EMailBarcodeOrientation_Horizontal | Barcode is horizontal. |
| | |

| | |
|---|---|
| EMailBarcodeOrientation_Vertical | Barcode is vertical. |
| EMailBarcodeOrientation_Any | Barcode has any one of the supported orientations. |

# 6.63. EMailBarcodeSymbologies Enum

The symbologies supported by EMailBarcode

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EMailBarcodeSymbologies_Unknown | Unknown symbology. |
| EMailBarcodeSymbologies_JapanPost | Japan Post symbology. |
| EMailBarcodeSymbologies_Postnet | US POSTNET symbology. |
| EMailBarcodeSymbologies_Planet | US PLANET symbology. |
| EMailBarcodeSymbologies_IntelligentMail | US Intelligent Mail symbology. |
| EMailBarcodeSymbologies_USMail | US symbologies. |

# 6.64. EMapConversionMode Enum

Conversion modes to use in EConverter.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

| | |
|---|---|
| EMapConversionMode_MaxDynamic | Scale the value to fill the destination dynamic. |

| | |
|---|---|
| EMapConversionMode_Shift | Convert value by bit shifting. |

# 6.65. EMapConversionMode Enum

Conversion modes to use in EConverter.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

| | |
|---|---|
| EMapConversionMode_MaxDynamic | Scale the value to fill the destination dynamic. |
| EMapConversionMode_Shift | Convert value by bit shifting. |

# 6.66. EMatchContrastMode Enum

Allowed values for the contrast mode of EasyMatch.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EMatchContrastMode_Normal | Normal contrast. Pattern occurrences will be found with the same contrast as at learn time. Default mode. |
| EMatchContrastMode_Inverse | Inverse contrast. Pattern occurrences will be found with reversed contrast. |
| EMatchContrastMode_Any | Normal or inverse contrast. Pattern occurrences can be found both with normal and inverse contrast. |

# 6.67. EMatchingMode Enum

Allowed values for the matching mode of EasyOCR.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EMatchingMode_Rms | Root-mean-square error method is used. |
| EMatchingMode_Standard | Gray-level correlation method is used. |
| EMatchingMode_Normalized | Normalized gray-level correlation method is used. |

# 6.68. EMatrixCodeContrastMode Enum

-

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EMatrixCodeContrastMode_BlackOnWhite | Dark cells on a light background. |
| EMatrixCodeContrastMode_WhiteOnBlack | Light cells on a dark background. |

# 6.69. EMaximumAnalysisMode Enum

This enumeration contains the possible values for the analysis mode of the ELaserLineExtractor object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

| |
|---|

| | |
|---|---|
| EMaximumAnalysisMode_Peaks | Peak analysis mode. |
| EMaximumAnalysisMode_Max | Maximum analysis mode. |
| EMaximumAnalysisMode_COG | Center of gravity analysis mode. |

# 6.70. ENoiseRemovalMethod Enum

Type of filter used in method EFilters::RemoveNoise.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

| | |
|---|---|
| ENoiseRemovalMethod_ AbsoluteDifferenceFromMean | Removes points for which the deviation from the average height in the filter window is larger than the specified threshold. The threshold is the absolute value of the maximum difference. |
| ENoiseRemovalMethod_ RelativeDifferenceFromMean | Removes points for which the deviation from the average height in the filter window is larger than the specified threshold multiplied by the standard deviation (in the same filter window). The threshold is a factor. |
| ENoiseRemovalMethod_ HighStandardDeviation | Removes points for which the standard deviation calculated in the filter window is larger than a specified threshold. The threshold is the maximum standard deviation. |

# 6.71. ENormalizationMode Enum

Allowed values for the gray-level normalization mode in EasyOCV.

**Namespace:** Euresys::Open_eVision_2_10

| |
|---|

| | |
|---|---|
| ENormalizationMode_NoNormalization | No gray-level normalization (contrast changes will be detected). |
| ENormalizationMode_Moments | Contrast normalization based on linear change. |
| ENormalizationMode_Threshold | Contrast normalization based on non-linear change. |

# 6.72. EObjectBasedCalibrationPrecisionVsSpeedTradeOff Enum

The precision vs speed tradeoff.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

| | |
|---|---|
| EObjectBasedCalibrationPrecisionVsSpeedTradeOff_Fast | Fast mode. |
| EObjectBasedCalibrationPrecisionVsSpeedTradeOff_Balanced | Balanced mode. |
| EObjectBasedCalibrationPrecisionVsSpeedTradeOff_Precise | Precise mode. |

# 6.73. EObjectBasedCalibrationType Enum

The type of the calibration object.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

| | |
|---|---|
| EObjectBasedCalibrationType_DoublePyramid | Double Pyramid calibration model. |
| EObjectBasedCalibrationType_ TruncatedDoublePyramid | Double Truncated Pyramid calibration model. |
| EObjectBasedCalibrationType_NotDefined | Not Defined. |

# 6.74. EOCR2DetectionMethod Enum

This enumeration contains the possible methods for text detection in EOCR2.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EOCR2DetectionMethod_FixedWidth | This method is suited for detecting texts with fixed-with fonts and dotted characters. |
| EOCR2DetectionMethod_Proportional | This method is suited for detecting texts with proportional fonts. |

# 6.75. EOCR2SegmentationMethod Enum

This enumeration contains the possible methods for image segmentation in EOCR2.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EOCR2SegmentationMethod_Local | This method is more complex and suited for segmenting images with text on a nonuniform background, it uses a local threshold value that can be different for each character of the text. |
| | |

| | |
|---|---|
| EOCR2SegmentationMethod_Global | This method is fast and suited for segmenting images with clear text on a uniform background, it uses a global threshold value. |

6-

.-

# 76. EOCRClass Enum

Allowed values for the class of pattern in EasyOCR. These classes have no pre-defined meaning, the user is free to give them any meaning they like. For instance, class 0 could contains only digits, class 1 only the forward slash character, class 3 uppercase letters, etc. Any choice is allowed, as long as the correct classes are specified during the learning process. During recognition/reading, the user can specify the expected class for each character. This means that if a forward slash is expected at that position, they can say it should be class 1 (following the example from above). This will improve the recognition rate and speed because the algorithm only has to choose between characters within the specified class.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EOCRClass__0 | Character belongs to class 0. |
| EOCRClass__1 | Character belongs to class 1. |
| EOCRClass__2 | Character belongs to class 2. |
| EOCRClass__3 | Character belongs to class 3. |
| EOCRClass__4 | Character belongs to class 4. |
| EOCRClass__5 | Character belongs to class 5. |
| EOCRClass__6 | Character belongs to class 6. |
| EOCRClass__7 | Character belongs to class 7. |
| | |

| | |
|---|---|
| EOCRClass__8 | Character belongs to class 8. |
| EOCRClass__9 | Character belongs to class 9. |
| EOCRClass__10 | Character belongs to class 10. |
| EOCRClass__11 | Character belongs to class 11. |
| EOCRClass__12 | Character belongs to class 12. |
| EOCRClass__13 | Character belongs to class 13. |
| EOCRClass__14 | Character belongs to class 14. |
| EOCRClass__15 | Character belongs to class 15. |
| EOCRClass__16 | Character belongs to class 16. |
| EOCRClass__17 | Character belongs to class 17. |
| EOCRClass__18 | Character belongs to class 18. |
| EOCRClass__19 | Character belongs to class 19. |
| EOCRClass__20 | Character belongs to class 20. |
| EOCRClass__21 | Character belongs to class 21. |
| EOCRClass__22 | Character belongs to class 22. |
| EOCRClass__23 | Character belongs to class 23. |
| EOCRClass__24 | Character belongs to class 24. |

| | |
|---|---|
| EOCRClass__25 | Character belongs to class 25. |
| EOCRClass__26 | Character belongs to class 26. |
| EOCRClass__27 | Character belongs to class 27. |
| EOCRClass__28 | Character belongs to class 28. |
| EOCRClass__29 | Character belongs to class 29. |
| EOCRClass__30 | Character belongs to class 30. |
| EOCRClass_Digit | Character belongs to class 0. Equivalent to EOCRClass__0. |
| EOCRClass_UpperCase | Character belongs to class 1. Equivalent to EOCRClass__1. |
| EOCRClass_LowerCase | Character belongs to class 2. Equivalent to EOCRClass__2. |
| EOCRClass_Special | Character belongs to class 3. Equivalent to EOCRClass__3. |
| EOCRClass_Extended | Character belongs to class 4. Equivalent to EOCRClass__4. |
| EOCRClass_AllClasses | Character belongs to all classes, from 0 to 31 included. |

# 6.77. EOCRColor Enum

Allowed values for the text color in EasyOCR.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EOCRColor_BlackOnWhite | The characters appear darker than the background. |

| | |
|---|---|
| EOCRColor_WhiteOnBlack | The characters appear lighter than the background. |
| EOCRColor_DarkOnLight | The characters appear darker than the background. No thresholding takes place when the characters are learnt and/or recognized. |
| EOCRColor_LightOnDark | The characters appear lighter than the background. No thresholding takes place when the characters are learnt and/or recognized. |

# 6.78. EPatternType Enum

Allowed values for the type of patterns in EasyFind.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EPatternType_ConsistentEdges | Defines the **ConsistentEdges** pattern type. |
| EPatternType_ContrastingRegions | Defines the **ContrastingRegions** pattern type. |
| EPatternType_ThinStructure | Defines the **ThinStructure** pattern type. |
| EPatternType_Unknown | - |

# 6.79. EPickingMode Enum

-

**Namespace:** Euresys::Open_eVision_2_10

| |
|---|

| | |
|---|---|
| EPickingMode_All | - |
| EPickingMode_Begin | - |
| EPickingMode_End | - |
| EPickingMode_Central | - |
| EPickingMode_Score | - |

# 6.80. EPlaneCropperType Enum

Type of crop to use in EPlaneCropper.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

| | |
|---|---|
| EPlaneCropperType_KeepAbove | Only the points above the plane are selected. |
| EPlaneCropperType_KeepBelow | Only the points below the plane are selected. |
| EPlaneCropperType_KeepClose | Only the points closer to the plane than a specified distance ("maxDistance") are selected. |
| EPlaneCropperType_KeepFar | Only the points further away from the plane than a specified distance ("maxDistance") are selected. |

# 6.81. EPlotItem Enum

Defines how the profile is drawed across a gauge.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EPlotItem_Transitions | Displays the profile along a point location gauge. |
| EPlotItem_Peak | Displays the corresponding derivative curve. |
| EPlotItem_Thresholds | Displays the threshold and minimum amplitude levels. |
| EPlotItem_Points | Displays the valid transitions. |

# 6.82. EQRCodeCodingMode Enum

This enumeration contains the possible values for the coding mode of a QR code.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EQRCodeCodingMode_Basic | The QR code does not use a specific coding mode. |
| EQRCodeCodingMode_Fnc1_Gs1 | The QR code uses the FNC1/GS1 coding mode (FNC1 in first position). |
| EQRCodeCodingMode_Fnc1_Aim | The QR code uses the FNC1/AIM coding mode (FNC1 in second position). |

# 6.83. EQRCodeEncoding Enum

This enumeration contains the possible values for the encoding used for parts of the bit stream of a QR code.

**Namespace:** Euresys::Open_eVision_2_10

| |
|---|

| | |
|---|---|
| EQRCodeEncoding_Numeric | The stream part is coded numerically. |
| EQRCodeEncoding_Alphanumeric | The stream part is coded alphanumerically. |
| EQRCodeEncoding_Byte | The stream part is coded as bytes. |
| EQRCodeEncoding_Kanji | The stream part is coded in Kanji. |

# 6.84. EQRCodeLevel Enum

This enumeration contains the possible values for the level of error correction of a QR code.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EQRCodeLevel_L | The QR code is level L (about 7% of error correction). |
| EQRCodeLevel_M | The QR code is level M (about 15% of error correction). |
| EQRCodeLevel_Q | The QR code is level Q (about 25% of error correction). |
| EQRCodeLevel_H | The QR code is level H (about 30% of error correction). |

# 6.85. EQRCodeModel Enum

This enumeration contains the possible values for a QR code model.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EQRCodeModel_Model1 | The QR code is a model 1. |

| | |
|---|---|
| EQRCodeModel_Model2 | The QR code is a model 2 or 2005. |
| EQRCodeModel_MicroQR | The QR code is a Micro QR. |

# 6.86. EQRCodePerspectiveMode Enum

This enumeration contains the possible values for the scanning precision of a QR Code reader object.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EQRCodePerspectiveMode_Basic | The QR Code reader handles light perspective deformations. |
| EQRCodePerspectiveMode_Improved | The QR Code reader handles heavy perspective deformations. |

## Remarks

This setting has deprecated as per Open eVision release 2.0, setting the perspectiveMode will have no effect. The EQRCodePerspectiveMode_Basic option has been superceded by EQRDetectionMethod_GradientLegacy, the EQRCodePerspectiveMode_Improved option has been superceded by EQRDetectionMethod_PerspectiveLegacy.

# 6.87. EQRCodeScanPrecision Enum

This enumeration contains the possible values for the scanning precision of a QR code reader object.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EQRCodeScanPrecision_Automatic | The QR code reader determines the scan precision automatically. |
| EQRCodeScanPrecision_Fine | The QR code reader scans finely the search field to find the QR codes. This value is recommended for small images or large images with small QR codes. |
| | |

| | |
|---|---|
| EQRCodeScanPrecision_Coarse | The QR code reader scans coarsely the search field to find the QR codes. This value is recommended for large images with medium to large QR codes. |

# 88. EQRDetectionMethod Enum

This enumeration contains the possible detection methods for QR codes, combinations of the methods are allowed.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EQRDetectionMethod_AdaptiveThreshold | This method detects finder patterns based on adaptive thresholding of the image. |
| EQRDetectionMethod_Gradient | This method detects finder patterns based on gradients in the image. |
| EQRDetectionMethod_PerspectiveLegacy | This selects the gradient-based detection algorithms with improved perspective mode developed for eVision 1.2.2. |
| EQRDetectionMethod_GradientLegacy | This selects the gradient-based detection algorithms with basic perspective mode developed for eVision 1.2.2. |

### Remarks

The variables Topology, CharHeight, CharWidth should be set before performing this operation.

# 6.89. EQRDetectionTradeOff Enum

This enumeration contains several settings for the tradeoff between detection speed and reliability of the easyQRCode methods. Setting this parameter will overwrite the current settings for EQRDetectionMethod and EQRCodeScanPrecision.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EQRDetectionTradeOff_FavorSpeed | |
| | This setting gives the fastest detection speed, but may reduce the detection accuracy. Sets EQRDetectionMethod_AdaptiveThreshold and EQRCodeScanPrecision_Coarse. |
| EQRDetectionTradeOff_Balanced | This setting gives the a balance between detection speed and reliability. Sets EQRDetectionMethod_AdaptiveThreshold \| EQRDetectionMethod_Gradient and EQRCodeScanPrecision_ Automatic. |
| EQRDetectionTradeOff_FavorReliability | This setting gives the best detection reliability, at the cost of detection speed. Sets EQRDetectionMethod_AdaptiveThreshold \| EQRDetectionMethod_Gradient and EQRCodeScanPrecision_Fine. |
| EQRDetectionTradeOff_Custom | |
| | This setting is returned when the current settings EQRDetectionMethod and EQRCodeScanPrecision do not match any of the EQRDetectionTradeOff presets. This choice should NOT be used to set a desired trade-off setting. |

# 6.90. EQualityIndicator Enum

Allowed values for the quality indicators in EasyOCV.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EQualityIndicator_Location | Global scores based on the edge pixels of the characters. |
| EQualityIndicator_Area | Foreground and background pixel counts. |
| EQualityIndicator_Sum | |

| | |
|---|---|
| EQualityIndicator_Correlation | Sum of the normalized foreground and background gray-level values. |
| EQualityIndicator_Contrast | Global contrast of the inspected ROI. |

orrelation.

# 6.91. EReadMode Enum

This enumeration contains the possible operation modes for the EMatrixCodeReader::Read method

**Namespace:** Euresys::Open_eVision_2_10::EasyMatrixCode2

| | |
|---|---|
| EReadMode_Speed | The EMatrixCodeReader::Read method will halt as soon as one of the following is true:<br>(1) it has found the required number of codes given by EMatrixCodeReader::MaxNumCodes.<br>(2) it reaches the timelimit given by EMatrixCodeReader::TimeOut.<br>(3) it has completely finished its process.<br>This mode will result in the shortest processing times. |
| EReadMode_Quality | The EMatrixCodeReader::Read method will keep trying to improve its detection until one of the following is true:<br>(1) it reaches the timelimit given by EMatrixCodeReader::TimeOut.<br>(2) it has completely finished its process.<br>This mode will results in the best grading results. |

# 6.92. ERectangleMode Enum

The modes that specify how the selection of coded elements with a rectangle behaves.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ERectangleMode_EntirelyInside | |
| | Takes into consideration only the coded elements that entirely lie inside the given rectangle, not touching its borders |
| ERectangleMode_EntirelyOutside | |
| | Takes into consideration only the coded elements that entirely lie outside the given rectangle, not touching its borders. |
| ERectangleMode_InsideOrOnBorder | |
| | Takes into consideration only the coded elements that entirely lie inside the given rectangle, or that touch its borders. |
| ERectangleMode_OutsideOrOnBorder | |
| | Takes into consideration only the coded elements that entirely lie outside the given rectangle, or that touch its borders. |
| ERectangleMode_OnBorder | Takes into consideration only the coded elements that touch the borders of the rectangle. |

# 6.93. EReductionMode Enum

The reduction mode to be used when learning a Consistent Edges model.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EReductionMode_Auto | Use the best-guess algorithm for selecting the reduction strength when learning a model. |
| | |

| | |
|---|---|
| EReductionMode_Manual | Use a user-set value for the reduction strength when learning a model (cf. the property EPatternFinder::ReductionStrength). |
| EReductionMode_Unknown | - |

# 6.94. EReferenceNoise Enum

Enumeration for specifying how a reference image is affected by noise in EasyImage.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EReferenceNoise_NoReference | The reference image is free from noise (synthetic image or noise source cancelled). |
| EReferenceNoise_SameAsImage | The reference image is contaminated by the same noise source as the source image. |

# 6.95. ERgbStandard Enum

Allowed values for the RGB standard in EasyColor.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ERgbStandard_Ntsc | NTSC primaries with the following CIE XYZ coordinates: Red: (0.607, 0.299, 0.000), Green: (0.174, 0.587, 0.066), Blue: (0.201, 0.114, 1.117). |
| ERgbStandard_Pal | PAL primaries with the following CIE XYZ coordinates: Red: (0.4303, 0.2219, 0.0202), Green: (0.3416, 0.7068, 0.1296), Blue: (0.1784, 0.0713, 0.9393). |
| | |

| | |
|---|---|
| ERgbStandard_Smpte | SMPTE primaries with the following CIE XYZ coordinates: Red: (0.393, 0.212, 0.019), Green: (0.365, 0.701, 0.112), Blue: (0.192, 0.087, 0.958). |

**Remarks**

The definition of the RGB primaries is not unique. In principle, there is one RGB system for each set of phosphors used in color monitors. Anyway, the CCIR has defined standard combinations for use in digital TV broadcast. Before performing a conversion, function EasyColor::RgbStandard can be used to specify the standard used.

# 6.96. ERoiHit Enum

Describes the ROI that was hit by the mouse cursor.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ERoiHit_NoHit | No ROI. |
| ERoiHit_Learn_0 | First learning ROI. |
| ERoiHit_Learn_1 | Second learning ROI. |
| ERoiHit_Match_0 | First matching ROI. |
| ERoiHit_Match_1 | Second matching ROI. |
| ERoiHit_Inspect | Inspection ROI. |

# 6.97. ESegmentationMethod Enum

The segmentation methods that are available to the image encoder.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ESegmentationMethod_Custom | - |
| ESegmentationMethod_BinaryImage | Segmentation of binary images (cf. EBinaryImageSegmenter). |
| ESegmentationMethod_ColorRangeThreshold | Segments a color image by specifying a cube in the RGB space (cf. EColorRangeThresholdSegmenter). |
| ESegmentationMethod_ColorSingleThreshold | Segments a color image by specifying a single threshold (cf. EColorSingleThresholdSegmenter). |
| ESegmentationMethod_ GrayscaleDoubleThreshold | Segments a grayscale image by specifying a double threshold (cf. EGrayscaleDoubleThresholdSegmenter). |
| ESegmentationMethod_ GrayscaleSingleThreshold | Segments a grayscale image by specifying a single threshold (cf. EGrayscaleSingleThresholdSegmenter). |
| ESegmentationMethod_ImageRange | Segments an image by specifying a pixel-by-pixel double threshold (cf. EGrayscaleDoubleThresholdSegmenter). |
| ESegmentationMethod_ReferenceImage | Segments an image by specifying a pixel-by-pixel single threshold (cf. EGrayscaleSingleThresholdSegmenter). |
| ESegmentationMethod_LabeledImage | Segments an image by mapping the value of the pixels directly to a layer index (cf. ELabeledImageSegmenter). |

### Remarks

The parameters of the segmentation methods are configured through the getters finishing by "Segmenter" that are available in EImageEncoder.

# 6.98. ESegmentationMode Enum

Allowed values for the segmentation mode in EasyOCR.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ESegmentationMode_KeepObjects | After segmentation, keep the blobs as they were found. |
| ESegmentationMode_RepasteObjects | After segmentation, group together the blobs believed to belong to the same character. |

# 6.99. ESelectByPosition Enum

Allowed values for the selection mode of ECodedImage.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ESelectByPosition_InsertIn | Insert the objects completely inside the given area. |
| ESelectByPosition_InsertTouch | Insert all the objects with a non-empty intersection with the given area. |
| ESelectByPosition_InsertOut | Insert the objects completely outside the given area. |
| ESelectByPosition_RemoveIn | Remove the objects completely inside the given area. |
| ESelectByPosition_RemoveTouch | Remove all the objects with a non-empty intersection with the given area. |
| ESelectByPosition_RemoveOut | Remove the objects completely outside the given area. |
| ESelectByPosition_RemoveBorder | Remove the objects outside the given area (including the objects touching the given area boundary). |

## Remarks

When specifying the position by means of an ROI, the minimum width and height of the ROI object must be at least 3 pixels. This enumeration pertains to the EasyObject legacy API and should not be used for new developments. See ECodedImage2 for the new API.

# 6.100. ESelectionFlag Enum

Specifies to which subset of a selection an operation should be applied in EasyObject and EasyOCV.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ESelectionFlag_Any | The operation applies to both selected and unselected items. |
| ESelectionFlag_True | The operation applies to selected items only. |
| ESelectionFlag_False | The operation applies to unselected items only. |

# 6.101. ESelectOption Enum

Allowed values for the selection mode of ECodedImage. This enumeration pertains to the EasyObject legacy API and should not be used for new developments. See ECodedImage2 for the new API.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ESelectOption_InsertAll | Add all objects. |
| ESelectOption_InsertGreaterOrEqual | Add all objects with feature value above the upper threshold. |
| ESelectOption_InsertLesserOrEqual | Add all objects with feature value below the lower threshold. |
| ESelectOption_InsertRange | Add all objects with feature value between or equal to the lower and upper thresholds. |
| ESelectOption_RemoveAll | Delete all objects. |
| ESelectOption_RemoveGreaterOrEqual | Delete all objects with feature value above the lower threshold. |

| | |
|---|---|
| ESelectOption_RemoveLesserOrEqual | Delete all objects with feature value below the upper threshold. |
| ESelectOption_RemoveRange | Delete all objects with feature value between or equal to the lower and upper thresholds. |
| ESelectOption_InsertOutOfRange | Add all objects with feature value above the upper and below the lower threshold. |
| ESelectOption_RemoveOutOfRange | Delete all objects with feature value above the upper and below the lower threshold. |

# 6.102. ESerializerFileWriterMode Enum

Creation mode of the file.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ESerializerFileWriterMode_Create | Creates the archive file on the hard disk. If the file already exists, the ESerializer::CreateFileWriter method returns **NULL**. |
| ESerializerFileWriterMode_Overwrite | Overwrites the previously created archive if it already exists, or creates it otherwise. |
| ESerializerFileWriterMode_Append | Appends the data at the end of the previously created archive, or creates the archive if it doesn't exist. |

# 6.103. EShapeBehavior Enum

Allowed values for conditions on the behavior of a shape.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EShapeBehavior_Visible | Identifies a visible shape. |
| EShapeBehavior_Selected | Identifies a selected shape. |
| EShapeBehavior_Selectable | Identifies a selectable shape. |
| EShapeBehavior_Dragable | Identifies a dragable shape. |
| EShapeBehavior_Rotatable | Identifies a rotatable shape. |
| EShapeBehavior_Resizable | Identifies a resizable shape. |
| EShapeBehavior_Labeled | Identifies a labeled shape. |
| EShapeBehavior_Active | Identifies an active shape. |
| EShapeBehavior_Passed | Identifies a non defective shape. |

# 6.104. EShapeType Enum

Gauge type.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EShapeType_NoShape | - |
| EShapeType_FrameShape | Defines a frame shape. |
| EShapeType_WorldShape | - |
| | |

| | |
|---|---|
| EShapeType_PointGauge | Defines a point location gauge. |
| EShapeType_LineGauge | Defines a line fitting gauge. |
| EShapeType_CircleGauge | Defines a circle fitting gauge. |
| EShapeType_RectangleGauge | Defines a rectangle fitting gauge. |
| EShapeType_WedgeGauge | Defines a wedge fitting gauge. |

# 6.105. EShiftingMode Enum

Allowed values for the shifting mode of EasyOCR.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EShiftingMode_Chars | Each character is moved individually. |
| EShiftingMode_Text | The all set of characters is moved together. |

# 6.106. ESingleThresholdMode Enum

The single threshold mode for the selection of coded elements with respect to a given feature.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ESingleThresholdMode_Less | The value of the feature must be strictly less than the threshold. |
| ESingleThresholdMode_LessEqual | The value of the feature must be less or equal to the threshold. |

| | |
|---|---|
| ESingleThresholdMode_Equal | The value of the feature must be equal to the threshold. |
| ESingleThresholdMode_GreaterEqual | The value of the feature must be greater or equal to the threshold. |
| ESingleThresholdMode_Greater | The value of the feature must be strictly greater than the threshold. |
| ESingleThresholdMode_Different | The value of the feature must be different to the threshold. |

# 6.107. ESortDirection Enum

The sorting mode for selections of coded elements based on their features.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ESortDirection_Ascending | Sorts the coded elements with respect to a given feature, in ascending order. |
| ESortDirection_Descending | Sorts the coded elements with respect to a given feature, in descending order. |

# 6.108. ESortOption Enum

Allowed values for the sort mode of ECodedImage. This enumeration pertains to the EasyObject legacy API and should not be used for new developments. See ECodedImage2 for the new API.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ESortOption_Ascending | Sort by increasing feature values. |
| | |

| | |
|---|---|
| ESortOption_Descending | Sort by decreasing feature values. |

6-

.-

# 109. EStockMeasurementUnit Enum

Allowed values for the type of Measurement Unit.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EStockMeasurementUnit_None | Defines the **None** value type. |
| EStockMeasurementUnit_um | Defines the **micron meter** value type. |
| EStockMeasurementUnit_mm | Defines the **milimeter** value type. |
| EStockMeasurementUnit_cm | Defines the **centimeter** value type. |
| EStockMeasurementUnit_dm | Defines the **decimeter** value type. |
| EStockMeasurementUnit_m | Defines the **meter** value type. |
| EStockMeasurementUnit_dam | Defines the **decameter** value type. |
| EStockMeasurementUnit_hm | Defines the **Hectometer** value type. |
| EStockMeasurementUnit_km | Defines the **Kilometer** value type. |
| EStockMeasurementUnit_mil | Defines the **milliliter** value type. |
| EStockMeasurementUnit_inch | Defines the **inch** value type. |

| | |
|---|---|
| EStockMeasurementUnit_foot | Defines the **foot** value type. |
| EStockMeasurementUnit_yard | Defines the **yard** value type. |
| EStockMeasurementUnit_mile | Defines the **mile** value type. |

# 6.110. ESymbologies Enum

The symbologies supported by EasyBarCode.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ESymbologies_Standard_Symbologies | Reserved for internal use |
| ESymbologies_Additional_Symbologies | Reserved for internal use |
| ESymbologies_Codabar | Codabar symbology. |
| ESymbologies_Code128 | Code 128 symbology. |
| ESymbologies_Code25Interleaved | Code 25 Interleaved symbology. |
| ESymbologies_Code39 | Code 39 symbology. |
| ESymbologies_Ean128 | EAN 128 symbology. |
| ESymbologies_Ean13 | EAN 13 symbology. |
| ESymbologies_Msi | MSI symbology. |
| ESymbologies_UpcA | UPC A symbology. |

| | |
|---|---|
| ESymbologies_UpcE | UPC E symbology. |
| ESymbologies_BinaryCode | Binary Code symbology. |
| ESymbologies_AdsAnker | Code ABC Anker symbology. |
| ESymbologies_Bc412 | Code BC 412 symbology. |
| ESymbologies_Code11 | Code 11 symbology. |
| ESymbologies_Code13 | Code 13 symbology. |
| ESymbologies_Code25Datalogic | Code 25 DataLogic symbology. |
| ESymbologies_Code25Matrix | Code 25 Matrix symbology. |
| ESymbologies_Code25Iata | Code 25 IATA symbology. |
| ESymbologies_Code25Industry | Code 25 Industry symbology. |
| ESymbologies_Code25Compressed | Code 25 Compressed symbology. |
| ESymbologies_Code25Inverted | Code 25 Inverted symbology. |
| ESymbologies_Code32 | Code 32 symbology. |
| ESymbologies_Code39Extended | Code 39 Extended symbology. |
| ESymbologies_Code39Reduced | Code 39 Reduced symbology. |
| ESymbologies_Code93 | Code 93 symbology. |
| ESymbologies_Code93Extended | Code 93 Extended symbology. |

| | |
|---|---|
| ESymbologies_CodeBcdMatrix | Code BCD Matrix symbology. |
| ESymbologies_CodeCip | Code CIP symbology. |
| ESymbologies_CodeStk | Code STK symbology. |
| ESymbologies_Ean8 | EAN 8 symbology. |
| ESymbologies_IbmDeltaDistanceA | IBM Delta Distance A symbology. |
| ESymbologies_Plessey | Plessey symbology. |
| ESymbologies_Telepen | Telepen symbology. |
| ESymbologies_Rss14 | RSS-14 symbology. |
| ESymbologies_Rss14Limited | RSS-14 Limited symbology. |
| ESymbologies_Rss14Expanded | RSS-14 Expanded symbology. |
| ESymbologies_Standard | Gathers all the symbologies belonging to the standard group. |
| ESymbologies_Additional | Gathers all the symbologies belonging to the additional group. |
| ESymbologies_Unknown | - |

## Remarks

Due to the large number of supported symbologies, they have been splitted into two groups. The most commonly used symbologies have been gathered under the name Standard symbologies. The remaining symbologies belong to the Additional symbologies group.

# 6.111. EThinStructureMode Enum

Allowed values for the type of thin structures in EasyFind.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EThinStructureMode_Auto | Lets EasyFind choose automatically the best contrast of thin elements. |
| EThinStructureMode_Dark | Favors thin elements darker than regions. |
| EThinStructureMode_Bright | Favors thin elements brighter than regions. |

# 6.112. EThresholdMode Enum

The various modes for thresholding that are supported by Open eVision.

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| EThresholdMode_Absolute | Reserved value. For absolute thresholding, use the threshold value itself, cast to EKernelRectifier. |
| EThresholdMode_Relative | Relative threshold; determines the required threshold level so that a given fraction of the image pixels lie below it. |
| EThresholdMode_MinResidue | Selects a threshold value such that the quadratic difference between the source and thresholded image is minimized. |
| EThresholdMode_MaxEntropy | |

| | |
|---|---|
| | Selects a threshold value such that the entropy (i.e. the amount of information) of the resulting thresholded image is maximized. |
| EThresholdMode_Isodata | Selects a threshold value that lies halfway between the average dark gray value (i.e. gray levels below the threshold) and average light gray values (i.e. gray levels above the threshold). |

# 6.113. ETransitionChoice Enum

The transition selection method applied by the gauge measurement

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ETransitionChoice_NthFromBegin | N-th transition from the beginning (counting from 0). |
| ETransitionChoice_NthFromEnd | N-th transition from the end (counting from 0). |
| ETransitionChoice_LargestAmplitude | Transition whose peak has the largest amplitude value. |
| ETransitionChoice_LargestArea | Transition whose peak has the largest area value. |
| ETransitionChoice_Closest | Transition closest to the center. |
| ETransitionChoice_All | All transitions. |

# 6.114. ETransitionType Enum

The type of transition to be retained by the gauge measurement

**Namespace:** Euresys::Open_eVision_2_10

| | |
|---|---|
| ETransitionType_Bw | Black to white. |
| ETransitionType_Wb | White to black. |
| ETransitionType_BwOrWb | Black to white or white to black. |
| ETransitionType_Bwb | Black to white to black. |
| ETransitionType_Wbw | White to black to white. |

# 6.115. EZMapOrientationVectorMode Enum

The EZMap orientation, it's the direction of the X (width) axis of the ZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

| | |
|---|---|
| EZMapOrientationVectorMode_Auto | Chooses automatically the orientation of the ZMap. |
| EZMapOrientationVectorMode_XAxis | The X (width) axis of the ZMap is aligned with the world X axis. |
| EZMapOrientationVectorMode_YAxis | The X (width) axis of the ZMap is aligned with the world Y axis. |
| EZMapOrientationVectorMode_ZAxis | The X (width) axis of the ZMap is aligned with the world Z axis. |
| EZMapOrientationVectorMode_Explicit | The orientation vector is arbitrary and given by the method SetOrientationVector. |

# 6.116. EZMapReferencePlaneMode Enum

The 3D reference plane used to build the EZMap.

**Namespace:** Euresys::Open_eVision_2_10::Easy3D

| | |
|---|---|
| EZMapReferencePlaneMode_XPlane | The reference plane is orthogonal to the X axis (YZ domain). |
| EZMapReferencePlaneMode_YPlane | The reference plane is orthogonal to the Y axis (XZ domain). |
| EZMapReferencePlaneMode_ZPlane | The reference plane is orthogonal to the Z axis (XY domain). That's the default reference plane. |
| EZMapReferencePlaneMode_Explicit | The reference plane is arbitrary and given by the method SetReferencePlane. |

# 6.117. Features Enum

Open eVision Features.

**Namespace:** Euresys::Open_eVision_2_10::LicenseFeatures

| | |
|---|---|
| EasyGauge | - |
| EasyColor | - |
| EasyImage | - |
| EasyObject | - |
| EasyBarCode | - |

| | |
|---|---|
| EasyMatch | - |
| eVisionStudio | - |
| EasyFind | - |
| EasyMatrixCode | - |
| EasyOCR | - |
| EasyOCV | - |
| EasyQRCode | - |
| EasyOCR2 | - |
| Easy3D | - |
| EasyDeepLearning | - |
| Easy3DObject | - |
| Easy3DMatch | - |
| Easy3DLaserLine | - |