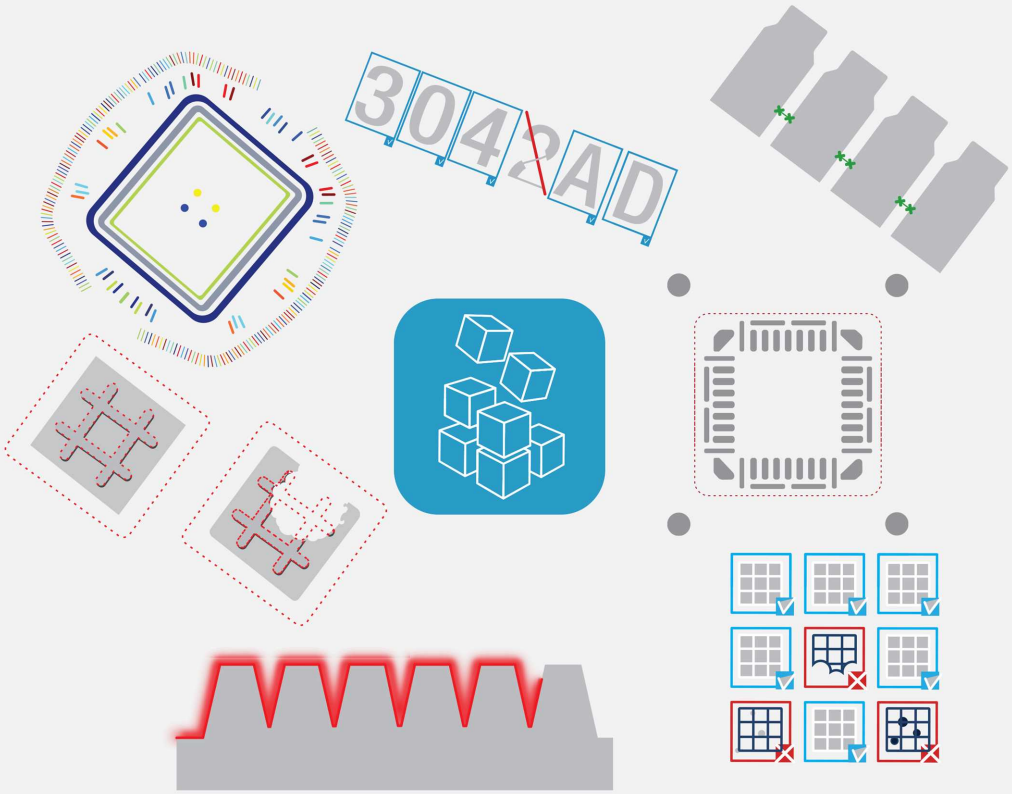
euresys
Empowering Computer Vision

CODE SNIPPETS

Open eVision



This documentation is provided with **Open eVision 24.02.0** (doc build **1198**).
www.euresys.com

Contents

1. Basic Types

1.1. Loading and Saving Images

Functional Guide | Reference: Load, Save, SaveJpeg

```
////////////////////////////////////////////////////////////
// This code snippet shows how to load and save an image. //
////////////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;

// Load an image file
srcImage.Load("mySourceImage.bmp");

// ...

// Save the destination image into a file
dstImage.Save("myDestImage.bmp");

// Save the destination image into a jpeg file
// The default compression quality is 75
dstImage.Save("myDestImage.jpg");

// Save the destination image into a jpeg file
// set the compression quality to 50
dstImage.SaveJpeg("myDestImage50.jpg", 50);
```

1.2. Interfacing Third-Party Images

Functional Guide | Reference: SetImagePtr

```
////////////////////////////////////////////////////////////////
// This code snippet shows how to link an Open eVision image //
// to an externally allocated buffer.                        //
////////////////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;

// Size of the third-party image
int sizeX;
int sizeY;

//Pointer to the third-party image buffer
EBW8* imgPtr;

// ...

// Link the Open eVision image to the third-party image
```

```
// Assuming the corresponding  buffer is aligned on 4 bytes
srcImage.SetImagePtr(sizeX, sizeY, imgPtr);
```

1.3. Retrieving Pixel Values

Functional Guide | Reference: GetImagePtr

```
////////////////////////////////////////////////////////////////
// This code snippet shows the recommended method (fastest)  //
// to access the pixel values in a BW8 image                 //
////////////////////////////////////////////////////////////////

EImageBW8 img;

OEV_UINT8* pixelPtr;
OEV_UINT8* rowPtr;
OEV_UINT8  pixelValue;
OEV_UINT32 rowPitch;
int x, y;

rowPtr = reinterpret_cast <OEV_UINT8*>(img.GetImagePtr());
rowPitch = img.GetRowPitch();

for (y = 0; y < height; y++)
{
    pixelPtr = rowPtr;

    for (x = 0; x < width; x++)
    {
        pixelValue = *pixelPtr;

        // Add your pixel computation code here

        *pixelPtr = pixelValue;
        pixelPtr++;
    }

    rowPtr += rowPitch;
}
```

1.4. Importing Bitmap from the Resources

Functional Guide | Reference: SetImagePtr

```
////////////////////////////////////////////////////////////////
// This code snippet shows how to import a bitmap from       //
// the resources.                                            //
////////////////////////////////////////////////////////////////

// Get the bitmap
HBITMAP hbitmap = (HBITMAP)LoadImage(GetModuleHandle(NULL), MAKEINTRESOURCE(IDB_BITMAP1), IMAGE_BITMAP, 0, 0,
LR_CREATEDIBSECTION);
BITMAP bitmap;
GetObject(hbitmap, sizeof(bitmap), (LPVOID)&bitmap);

int width = bitmap.bmWidth;
```

```
int height = bitmap.bmHeight;
UINT8* buffer = reinterpret_cast<UINT8*>(bitmap.bmBits);

EImageC24 image(width, height);

for (int y = 0; y < height; ++y)
{
  // Copy the entire row
  memcpy(image.GetImagePtr(0, height - 1 - y), buffer, 3 * width);
  buffer += 3 * width;
}
```

1.5. ROI Placement

Functional Guide | Reference: Attach, SetPlacement

```
//////////////////////////////////////////////////////////////
// This code snippet shows how to attach an ROI to an image  //
// and set its placement.                                    //
//////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 parentImage;

// ROI constructor
EROIBW8 myROI;

// ...

// Attach the ROI to the image
myROI.Attach(&parentImage);

//Set the ROI position
myROI.SetPlacement(50, 50, 200, 100);
```

1.6. Vector Management

Functional Guide | Reference: Empty, AddElement

```
//////////////////////////////////////////////////////////////
// This code snippet shows how to create a vector, fill it   //
// and retrieve the value of a given element.                //
//////////////////////////////////////////////////////////////

// EBW8Vector constructor
EBW8Vector ramp;

// Clear the vector
ramp.Empty();

// Fill the vector with increasing values
for(int i= 0; i < 128; i++)
{
       ramp.AddElement((EBW8)i);
}
```

```
// Retrieve the 10th element value
EBW8 value= ramp[9];
```

1.7. Exception Management

Functional Guide | Reference: GetPixel, What

```
//////////////////////////////////////////
// This code snippet shows how to manage  //
// Open eVision exceptions.               //
//////////////////////////////////////////

try
{
	// Image constructor
	EImageC24 srcImage;

	// ...

	// Retrieve the pixel value at coordinates (56, 73)
	EC24 value= srcImage.GetPixel(56, 730);
}

catch(Euresys::Open_eVision_1_1::EException exc)
{
	// Retrieve the exception description
	std::string error = exc.What();
}
```

2. ERegion

See also: Arbitrary-Shaped ROI (ERegion) / example: Inspecting Pads Using Regions

2.1. Basic Usage

```
////////////////////////////////////////////////////////////
// This code snippet shows how to perform a threshold on a //
// circular region in an image.                            //
////////////////////////////////////////////////////////////


               // Image constructors
EImageBW8 srcImage;
EImageBW8 dstImage;

//...

// Create the region
ECircleRegion circleRegion(center, radius);

// Threshold the image
EasyImage::Threshold(&srcImage, circleRegion, &dstImage);
```

2.2. Prepare Once, Use Multiple Times

```
////////////////////////////////////////////////////////////
// This code snippet shows how to perform a threshold on a //
// circular region in multiple image while preparing it    //
// only once.                                              //
////////////////////////////////////////////////////////////



               // Image constructors
EImageBW8 srcImage[10];
EImageBW8 dstImage[10];

//...

// Create the region
ECircleRegion circleRegion(center, radius);

// Prepare the region
circleRegion.Prepare(srcImage[0]);

// Threshold the images
for (int i = 0; i < 10; i++)
  EasyImage::Threshold(&srcImage[i], circleRegion, &dstImage[i]);
```

2.3. Combine Regions

```
////////////////////////////////////////////////////////////
// This code snippet shows how to perform a threshold on a //
// combined region in an image                             //
////////////////////////////////////////////////////////////

// Image constructors
EImageBW8 srcImage;
EImageBW8 dstImage;

//...

// Create first region
ECircleRegion circleRegion(center, radius);

// Create second region
ERectangleRegion rectangleRegion(center, width, height, angle);


// Combine regions
ERegion combinedRegion = ERegion::Union(circleRegion, rectangleRegion);

// Threshold the image
EasyImage::Threshold(&srcImage, combinedRegion, &dstImage);
```

2.4. Tool Chain

```
////////////////////////////////////////////////////////////
// This code snippet shows how to perform a threshold on a //
// region coming for a previous EasyFind process           //
////////////////////////////////////////////////////////////

// Image constructors
EImageBW8 findImage;
EImageBW8 srcImage;
EImageBW8 dstImage;


// EPatternFinder constructor
EPatternFinder finder;

//...

// Use EasyFind
std::vector<EFoundPattern> patterns = finder.Find(&findImage);


// Create region from found pattern
ERegion foundRegion(patterns[0]);


// Threshold the image
EasyImage::Threshold(&srcImage, foundRegion, &dstImage);
```

3. EGrabberBridge

See also: EGrabberBridge - Using Images from Coaxlink

3.1. Using EGrabberBridge

```
//////////////////////////////////////////////////////////////////////
// This code snippet shows how to go from an EGrabber buffer to an //
// EGrabberImageBW8, compatible with Open eVision processing       //
//////////////////////////////////////////////////////////////////////

// Construct the EGrabber objects.
EGenTL genTL;
EGrabber<CallbackOnDemand> grabber(genTL);

// Allocate one buffer
grabber.reallocBuffers(1);

//...

// Start the grabber acquisition of one buffer
grabber.start(1);

// Get the acquired buffer
ScopedBuffer buffer(grabber);

// Convert the ScopedBuffer to an Open eVision data container
EGrabberBridge::EGrabberImageBW8 image(buffer.getInfo());

// Stop the grabber
grabber.stop();

// Use the EGrabberImageBW8 as an Open eVision EImage Object
// Here an inversion of the image is performed
EImageBW8 invertedImage(image.GetWidth(), image.GetHeight());
EasyImage::Oper(EArithmeticLogicOperation_Invert, &image, &invertedImage);
```

3.2. Using EGrabberBridge with Format Conversion

```
//////////////////////////////////////////////////////////////////////
// This code snippet shows how to go from an EGrabber buffer to an //
// EGrabberImageBW8, compatible with Open eVision processing using //
// format conversion//////////////////////////////////////////////////////////////////////////////

// Construct the EGrabber objects.
// The FormatConverter will automatically convert the EGenTL buffer to
// the chosen Open eVision image type
EGenTL genTL;
EGrabber<CallbackOnDemand> grabber(genTL);
```

```
FormatConverter converter(genTL);

// Allocate one buffer
grabber.reallocBuffers(1);

//...

// Start the grabber acquisition of one buffer
grabber.start(1);

// Get the acquired buffer
ScopedBuffer buffer(grabber);

// Convert the ScopedBuffer to an Open eVision data container
EGrabberBridge::EGrabberImageBW8 image(converter, buffer.getInfo());

// Stop the grabber
grabber.stop();
// Use the EGrabberImageBW8 as an Open eVision EImage Object
// Here an inversion of the image is performed
EImageBW8 invertedImage(image.GetWidth(), image.GetHeight());
EasyImage::Oper(EArithmeticLogicOperation_Invert, &image, &invertedImage);
```

3.3. Managing EGrabber Parameters

```
//////////////////////////////////////////////////////////////////////
// This code snippet shows how to go from an EGrabber buffer to an //
// EGrabberImageBW8, compatible with Open eVision
processing//////////////////////////////////////////////////////////////////////////

// Construct the EGrabber objects.
// The FormatConverter is optional and will automatically convert the EGenTL buffer to
// the chosen Open eVision image type
EGenTL genTL;
EGrabber<CallbackOnDemand> grabber(genTL);
FormatConverter converter(genTL);

// Allocate one buffer
grabber.reallocBuffers(1);

// ...


// Manage EGrabber features
// Get/set camera (RemoteModule) features of various types:
// string - integer - float.
// WARNING: The features might be specific to each camera
std::string pixelFormat = grabber.getString<RemoteModule>("PixelFormat");
grabber.setString<RemoteModule>("PixelFormat", "Mono8");

int64_t width = grabber.getInteger<RemoteModule>("Width");
grabber.setInteger<RemoteModule>("Width", 1024);

double exposureTime = grabber.getFloat<RemoteModule>("ExposureTime");
grabber.setFloat<RemoteModule>("ExposureTime", 60.0f);

// ...

// Start the grabber acquisition of one buffer
grabber.start(1);
```

```
// Get the acquired buffer
ScopedBuffer buffer(grabber);

// Convert the ScopedBuffer to an Open eVision data container
EGrabberBridge::EGrabberImageBW8 image(converter, buffer.getInfo());

// ...
```

4. VimbaXBridge

See also: VimbaXBridge - Using Images from VimbaX Sources

4.1. Using VimbaXBridge

```
//////////////////////////////////////////////////////////////////////
// This code snippet shows how to go from an VimbaX frame to an    //
// EVimbaXImageBW8, compatible with Open eVision processing        //
//////////////////////////////////////////////////////////////////////
// Get a EVimbaXImage from a C (VmbFrame_t) or C++ (VmbCPP::FramePtr)
// frame pointer. If vmbTransform has been included, an automatic
// pixel format conversion will be performed if needed.
VimbaXBridge::EVimbaXImageBW8 image(frame);

// Use the EVimbaXImageBW8 as an Open eVision EImage Object
// Here an inversion of the image is performed EImageBW8
EImageBW8 invertedImage(image.GetWidth(), image.GetHeight());
EasyImage::Oper(EArithmeticLogicOperation_Invert, &image, &invertedImage);
```

5. EasyImage

5.1. Thresholding

Single Thresholding

Functional Guide | Reference: SetSize, Threshold

```
//////////////////////////////////////////////////////////////
// This code snippet shows how to perform minimum residue   //
// thresholding, absolute thresholding and relative         //
// thresholding operations.                                 //
//////////////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;

// ...

// Source and destination images must have the same size
dstImage.SetSize(&srcImage);

// Minimum residue thresholding (default method)
EasyImage::Threshold(&srcImage, &dstImage);

// Absolute thresholding (threshold = 110)
EasyImage::Threshold(&srcImage, &dstImage, 110);

// Relative thresholding (70% black, 30% white)
EasyImage::Threshold(&srcImage, &dstImage, EThresholdMode_Relative, 0, 255, 0.7f);
```

Double Thresholding

Functional Guide | Reference: DoubleThreshold

```
//////////////////////////////////////////////////////////////
// This code snippet shows how to perform a thresholding    //
// operation based on low and high threshold values.        //
//////////////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;

// ...

// Source and destination images must have the same size
dstImage.SetSize(&srcImage);
```

```
// Double thresholding, low threshold = 50, high threshold = 150,
// pixels below 50 become black, pixels above 150 become white,
// pixels between thresholds become gray
EasyImage::DoubleThreshold(&srcImage, &dstImage, 50, 150, 0, 128, 255);
```

Histogram-Based Single Thresholding

Functional Guide | Reference: Histogram, HistogramThreshold

```
////////////////////////////////////////////////////////////////
// This code snippet shows how to perform a minimum residue   //
// thresholding operation based on an histogram.              //
////////////////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;

// Histogram constructor
EBWHistogramVector histo;

// Variables
unsigned int thresholdValue;
float avgBelowThr, avgAboveThr;

// ...

// Compute the histogram
EasyImage::Histogram(&srcImage, &histo);

// Compute the single threshold (and the average pixel values below and above the threshold)
thresholdValue= EThresholdMode_MinResidue;
EasyImage::HistogramThreshold(&histo, thresholdValue, avgBelowThr, avgAboveThr);

// Source and destination images must have the same size
dstImage.SetSize(&srcImage);

// Perform the single thresholding
EasyImage::Threshold(&srcImage, &dstImage, thresholdValue);
```

Histogram-Based Double Thresholding

Functional Guide | Reference: Histogram, ThreeLevelsMinResidueThreshold, DoubleThreshold

```
////////////////////////////////////////////////////////////////////
// This code snippet shows how to perform a double thresholding   //
// operation. The low and high threshold values are computed      //
// according to the minimum residue method based on an histogram. //
////////////////////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;

// Histogram constructor
EBWHistogramVector histo;

// Variables
```

```
EBW8 lowThr;
EBW8 highThr;
float avgBelowThr, avgBetweenThr, avgAboveThr;

// ...

// Compute the histogram
EasyImage::Histogram(&srcImage, &histo);

// Compute the low and high threshold values automatically
// (and the average pixel values below, between and above the threshold)
EasyImage::ThreeLevelsMinResidueThreshold(&histo, lowThr, highThr, avgBelowThr, avgBetweenThr, avgAboveThr);

// Source and destination images must have the same size
dstImage.SetSize(&srcImage);

// Perform the double thresholding
EasyImage::DoubleThreshold(&srcImage, &dstImage, lowThr.Value, highThr.Value);
```

5.2. Arithmetic and Logic Operations

Functional Guide | Reference: Oper

```
////////////////////////////////////////////////////
// This code snippet shows how to apply miscellaneous //
// arithmetic and logic operations to images.         //
////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcGray0, srcGray1, dstGray;
EImageC24 srcColor, dstColor;

// ...

// All images must have the same size
dstGray.SetSize(&srcGray0);
// ...

// Subtract srcGray1 from srcGray0
EasyImage::Oper(EArithmeticLogicOperation_Subtract, &srcGray0, &srcGray1, &dstGray);

// Multiply srcGray0 by a constant value
EasyImage::Oper(EArithmeticLogicOperation_Multiply, &srcGray0, (EBW8)2, &dstGray);

// Add a constant value to srcColor
EasyImage::Oper(EArithmeticLogicOperation_Add, &srcColor, EC24(128,64,196), &dstColor);

// Erase (blacken) the destination image where the source image is black
EasyImage::Oper(EArithmeticLogicOperation_SetZero, &srcGray0, (EBW8)0, &dstGray);
```

5.3. Convolution

Pre-Defined Kernel Filtering

```
////////////////////////////////////////////////////////
// This code snippet shows how to apply miscellaneous    //
// convolution operations based on pre-defined kernels.  //
////////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;

// ...

// Source and destination images must have the same size
dstImage.SetSize(&srcImage);

// Perform a Uniform filtering (5x5 kernel)
EasyImage::ConvolUniform(&srcImage, &dstImage, 2);

// Perform a Highpass filtering
EasyImage::ConvolHighpass1(&srcImage, &dstImage);

// Perform a Gradient filtering
EasyImage::ConvolGradient(&srcImage, &dstImage);

// Perform a Sobel filtering
EasyImage::ConvolSobel(&srcImage, &dstImage);
```

User-Defined Kernel Filtering

```
////////////////////////////////////////////////////////
// This code snippet shows how to apply a convolution  //
// operation based on a user-defined kernel.           //
////////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;

// ...

// Create and define a user-defined kernel
// (Frei-Chen row gradient, positive only)
EKernel kernel;
kernel.SetKernelData(0.2929f, 0, -0.2929f,
                     0.4142f, 0, -0.4142f,
                     0.2929f, 0, -0.2929f);

// Source and destination images must have the same size
dstImage.SetSize(&srcImage);

// Apply the convolution kernel
EasyImage::ConvolKernel(&srcImage, &dstImage, &kernel);
```

5.4. Non-Linear Filtering

Functional Guide | Reference

Morphological Filtering

Functional Guide | Reference: ErodeBox, DilateBox, OpenDisk

```
//////////////////////////////////////////////////////
// This code snippet shows how to apply miscellaneous  //
// morphological filtering operations.                 //
//////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;

// ...

// Source and destination images must have the same size
dstImage.SetSize(&srcImage);

// Perform an erosion (3x3 square kernel)
EasyImage::ErodeBox(&srcImage, &dstImage, 1);

// Perform a dilation (5x3 rectangular kernel)
EasyImage::DilateBox(&srcImage, &dstImage, 2, 1);

// Perform an Open operation (5x5 circular kernel)
EasyImage::OpenDisk(&srcImage, &dstImage, 2);
```

Hit-and-Miss Transform

Functional Guide | Reference: SetValue, HitAndMiss

```
//////////////////////////////////////////////////////////////
// This code snippet shows how to highlight the left corner //
// of a rhombus by means of a Hit-and-Miss operation.       //
//////////////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;

// ...

// Create and define a Hit-and-Miss kernel
// corresponding to the left corner of a rhombus
EHitAndMissKernel leftCorner(-1, -1, 1, 1);

// Left column of the kernel
leftCorner.SetValue(-1, 0, EHitAndMissValue_Background);

// Middle column of the kernel
leftCorner.SetValue(0, -1, EHitAndMissValue_Background);
leftCorner.SetValue(0, 0, EHitAndMissValue_Foreground);
```

```
leftCorner.SetValue(0, 1, EHitAndMissValue_Background);

// Right column of the kernel
leftCorner.SetValue(1, -1, EHitAndMissValue_Foreground);
leftCorner.SetValue(1, 0, EHitAndMissValue_Foreground);
leftCorner.SetValue(1, 1, EHitAndMissValue_Foreground);

// Source and destination images must have the same size
dstImage.SetSize(&srcImage);

// Apply the Hit-and-Miss kernel
EasyImage::HitAndMiss(&srcImage, &dstImage, leftCorner);
```

5.5. Vector Operations

Functional Guide | Reference

Path Sampling

Functional Guide | Reference: Empty, AddElement, ImageToPath

```
//////////////////////////////////////////////////////////////
// This code snippet shows how to retrieve and store the    //
// pixel values along a given path together with the        //
// corresponding pixel coordinates.                         //
//////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// ...

// Vector constructor
EBW8PathVector path;

// Path definition
path.Empty();
for (int i = 0; i < 100; i++)
{
  EBW8Path p;
  p.X = i;
  p.Y = i;
  p.Pixel = 128;
  path.AddElement(p);
}

// Get the image data along the path
EasyImage::ImageToPath(&srcImage, &path);
```

Profile Sampling

Functional Guide | Reference: ImageToLineSegment, LineSegmentToImage

```
////////////////////////////////////////////////////////////
// This code snippet shows how to set, retrieve and store   //
// the pixel values along a given line segment.             //
////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// ...

// Vector constructor
EBW8Vector profile;

// Get the image data along segment (10,510)-(500,40)
EasyImage::ImageToLineSegment(&srcImage, &profile, 10, 510, 500, 40);

// Set all these points to white (255) in the image
EasyImage::LineSegmentToImage(&srcImage, 255, 10, 510, 500, 40);
```

5.6. Statistics

Image Statistics

```
////////////////////////////////////////////////////////////////////
// This code snippet shows how to compute basic image statistics. //
////////////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// ...

// Count the number of pixels above the threshold (128)
INT32 count;
EasyImage::Area(&srcImage, 128, count);

// Compute the pixels' average and standard deviation values
float stdDev, average;
EasyImage::PixelStdDev(&srcImage, stdDev, average);

// Compute the image gravity center (pixels above threshold)
float x, y;
EasyImage::GravityCenter(&srcImage, 128, x, y);
```

Sliding Windows Statistics

Functional Guide | Reference: LocalAverage, LocalDeviation

```
////////////////////////////////////////////////////////////////////////////
// This code snippet shows how to perform sliding windows statistics. //
////////////////////////////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
```

```
EImageBW8 dstImage0, dstImage1;

// ...

// All images must have the same size
dstImage0.SetSize(&srcImage);
dstImage1.SetSize(&srcImage);

// Local average in a 11x11 window
EasyImage::LocalAverage(&srcImage, &dstImage0, 5, 5);

// Local deviation in a 11x11 window
EasyImage::LocalDeviation(&srcImage, &dstImage1, 5, 5);
```

Histogram-Based Statistics

Functional Guide | Reference: Histogram, AnalyseHistogram

```
//////////////////////////////////////////////////////
// This code snippet shows how to compute statistics   //
// based on an histogram.                              //
//////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// ...

// Histogram constructor
EBWHistogramVector histo;

// Compute the histogram
EasyImage::Histogram(&srcImage, &histo);

// Compute the average gray-level value
float average = EasyImage::AnalyseHistogram(&histo, EHistogramFeature_AveragePixelValue, 0, 255);

// Compute the gray-level standard deviation
float deviation = EasyImage::AnalyseHistogram(&histo, EHistogramFeature_PixelValueStdDev, 0, 255);
```

5.7. Noise Reduction by Integration

Functional Guide | Reference

Temporal Noise Reduction

Functional Guide | Reference: Oper

```
//////////////////////////////////////////////////
// This code snippet shows how to perform noise  //
// reduction by temporal averaging.              //
//////////////////////////////////////////////////

// Images constructor
```

```
EImageBW16 noisyImage, cleanImage;

// 16 bits work image used as an accumulator
EImageBW16 store;

// ...

// All images must have the same size
cleanImage.SetSize(&noisyImage);
store.SetSize(&noisyImage);

// Clear the accumulator image
EasyImage::Oper(EArithmeticLogicOperation_Copy, (EBW16)0, &store);

// Accumulation loop
int n;
for (n=0; n < 10; n++)
{
  // Acquire a new image into noisyImage
  // ...

  // Add this new noisy image into the accumulator
  EasyImage::Oper(EArithmeticLogicOperation_Add, &noisyImage, &store, &store);
}

// Perform noise reduction
EasyImage::Oper(EArithmeticLogicOperation_Divide, &store, (EBW16)n, &cleanImage);
```

Recursive Average

Functional Guide | Reference: Oper, SetRecursiveAverageLUT, RecursiveAverage

```
/////////////////////////////////////////////////////
// This code snippet shows how to perform noise  //
// reduction by recursive averaging.             //
/////////////////////////////////////////////////////

// Images constructor
EImageBW8 noisyImage, cleanImage;

// 16 bits work image used as an accumulator
EImageBW16 store;

// ...

// All images must have the same size
cleanImage.SetSize(&noisyImage);
store.SetSize(&noisyImage);

// Clear the accumulator image
EasyImage::Oper(EArithmeticLogicOperation_Copy, (EBW16)0, &store);

// Prepare the transfer lookup table (reduction factor = 3)
EBW16Vector lut;
EasyImage::SetRecursiveAverageLUT(&lut, 3.f);

// Perform the noise reduction
EasyImage::RecursiveAverage(&noisyImage, &store, &cleanImage, &lut);
```

5.8. Feature Point Detectors

Harris Corner Detector

Functional Guide | Reference: GetPointCount, GetPoint

```
////////////////////////////////////////////////////////////////
// This code snippet shows how to retrieve corners' coordinates //
// by means of the Harris corner detector algorithm.            //
////////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// ...

// Harris corner detector
EHarrisCornerDetector harris;
EHarrisInterestPoints interestPoints;
harris.SetIntegrationScale(2.f);

// Perform the corner detection
harris.Apply(srcImage, interestPoints);

// Retrieve the number of corners
unsigned int index = interestPoints.GetPointCount();

// Retrieve the first corner coordinates
EPoint point = interestPoints.GetPoint(0);
float x = point.GetX();
float y = point.GetY();
```

Canny Edge Detector

Functional Guide | Reference: Apply

```
////////////////////////////////////////////////////
// This code snippet shows how to highlight edges  //
// by means of the Canny edge detector algorithm.  //
////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;

// ...

// Canny edge detector
ECannyEdgeDetector canny;

// Source and destination images must have the same size
dstImage.SetSize(&srcImage);

// Perform the edges detection
canny.Apply(srcImage, dstImage);
```

5.9. Using Flexible Masks

Functional Guide | Reference

Computing Pixels Average

Functional Guide | Reference: PixelAverage

```
////////////////////////////////////////////////////////
// This code snippet shows how to compute statistics  //
// inside a region defined by a flexible mask.        //
////////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageBW8 mask;

// ...

// Compute the average value of the source image pixels
// corresponding to the mask do-care areas only
float average;
EasyImage::PixelAverage(&srcImage, &mask, average);
```

5.10. Warping

Performing a Ring Warping

```
// Create normal and warp images
EImageBW8 img;
EImageBW8 imgWarped;

EImageBW16 WarpX;
EImageBW16 WarpY;

// Load the image
img.Load("");

// Set size of imgWarped to what is wanted
imgWarped.SetSize(, );

// Set size of imgUnwarped to previous original image
imgUnwarped.SetSize(&img);

// Set size of warp images
WarpX.SetSize(&imgWarped);
WarpY.SetSize(&imgWarped);
EasyImage::Oper(EArithmeticLogicOperation_Copy, EBW8(0), &imgWarped);


// Do a regular ring warp
EasyImage::SetCircleWarp(, , , , , , , , &WarpX, &WarpY);
EasyImage::Warp(&img, &imgWarped, &WarpX, &WarpY);
```

Performing an Inverse Warping

NOTE: We use the same notation as the code snippet "Performing a Ring Warping" on page 27.

```
// Create imgUnwarped and inverseWarp images
EImageBW8 imgUnwarped;
EImageBW16 InverseWarpX;
EImageBW16 InverseWarpY;


// Set size of inverse warp images
InverseWarpX.SetSize(&img);
InverseWarpY.SetSize(&img);

EasyImage::Oper(EArithmeticLogicOperation_Copy, EBW8(0), &imgUnwarped);

EasyImage::SetInvCircleWarp(, , , , , , , , &InverseWarpX, &InverseWarpY);
EasyImage::Warp(&imgWarped, &imgUnwarped, &InverseWarpX, &InverseWarpY);
// Use an oper Set zero to add the background into the image
EasyImage::Oper(EArithmeticLogicOperation_SetZero, &imgUnwarped,&img, &imgUnwarped);
```

5.11. Fourier Transforms

Performing a Direct Fourier Transform

```
////////////////////////////////////////////////////////
// This code snippet shows how to compute            //
// the direct Fourier transform of a grayscale image //
// (from the spatial domain to the frequency domain) //
////////////////////////////////////////////////////////

// Assuming you have the following loaded image
EImageBW8 spatialImage;

// Initialize an empty image that will contain frequencies information
// Be aware of the specified dimensions
EImageBW32f frequencyImage(spatialImage.GetWidth * 2, spatialImage.GetHeight);

// Compute the direct Fourier transform
EFourierTransformer fourier;
fourier.SetFrequentialDomainFormat(EFrequentialDomainFormat_ComplexExtended);
fourier.DirectTransform(spatialImage, frequencyImage);
```

Performing an Inverse Fourier Transform

```
////////////////////////////////////////////////////////
// This code snippet shows how to compute            //
// the inverse Fourier transform to get              //
// the original grayscale image                      //
// (from the frequency domain to the spatial domain) //
////////////////////////////////////////////////////////

```

```
// Assuming you have the following loaded image
EImageBW32f frequencyImage;

// Initialize an empty image that will contain spatial information
// Be aware of the specified dimensions
EImageBW8 spatialImage(frequencyImage.GetWidth / 2, frequencyImage.GetHeight);

// Compute the inverse Fourier transform
EFourierTransformer fourier;
fourier.SetFrequentialDomainFormat(EFrequentialDomainFormat_ComplexExtended);
fourier.InverseTransform(frequencyImage, spatialImage);
```

6. EasyColor

6.1. Colorimetric Systems Conversion

Functional Guide | Reference: ConvertFromRgb, Transform

```
////////////////////////////////////////////////////////
// This code snippet shows how to convert a color image //
// from the RGB to the Lab colorimetric system.         //
////////////////////////////////////////////////////////

// Images constructor
EImageC24 srcImage;
EImageC24 dstImage;

// ...

// Prepare a lookup table for
// the RGB to La*b* conversion
EColorLookup lookup;
lookup.ConvertFromRgb(EColorSystem_Lab);

// Source and destination images must have the same size
dstImage.SetSize(&srcImage);

// Perform the color conversion
EasyColor::Transform(&srcImage, &dstImage, &lookup);
```

6.2. Color Components

Functional Guide | Reference: Compose, ConvertFromRgb, GetComponent

```
////////////////////////////////////////////////////////
// This code snippet shows how to create a color image  //
// from 3 grayscale images and extract the luminance    //
// component from a color image.                        //
////////////////////////////////////////////////////////

// Images constructor
EImageBW8 red, green, blue;
EImageC24 colorImage;
EImageBW8 luminance;

// ...

// Source and destination images must have the same size
colorImage.SetSize(&red);

// Combine the color planes into a color image
EasyColor::Compose(&red, &green, &blue, &colorImage);

// Prepare a lookup table for
```

```
// the RGB to LSH conversion
EColorLookup lookup;
lookup.ConvertFromRgb(EColorSystem_Lsh);

// Source and destination images must have the same size
luminance.SetSize(&colorImage);

// Get the Luminance component
EasyColor::GetComponent(&colorImage, &luminance, 0, &lookup);
```

6.3. White Balance

Functional Guide | Reference: PixelAverage, WhiteBalance, Transform

```
////////////////////////////////////////////////////////////
// This code snippet shows how to perform white balancing. //
////////////////////////////////////////////////////////////

// Images constructor
EImageC24 srcImage, dstImage;
EImageC24 whiteRef;

// ...

// Create a lookup table
EColorLookup lut;

// Measure the calibration values from a white reference image
float r, g, b;
EasyImage::PixelAverage(&whiteRef, r, g, b);

// Prepare the lookup table for
// a white balance operation
lut.WhiteBalance(1.00f, EasyColor::GetCompensateNtscGamma(), r, g, b);

// Source and destination images must have the same size
dstImage.SetSize(&srcImage);

// Perform the white balance operation
lut.Transform(&srcImage, &dstImage);
```

6.4. Pseudo-Coloring

Functional Guide | Reference: SetShading, PseudoColor

```
////////////////////////////////////////////////////////////
// This code snippet shows how to perform pseudo-coloring. //
////////////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageC24 dstImage;

// ...

// Create a pseudo-color lookup table
```

```
EPseudoColorLookup pcLut;

// Define a shade of pure tints, from red to blue
pcLut.SetShading(EC24(255, 0, 0), EC24(0, 0, 255), EColorSystem_Ish);

// Source and destination images must have the same size
dstImage.SetSize(&srcImage);

// Generate the pseudo-colored image
EasyColor::PseudoColor(&srcImage, &dstImage, &pcLut);
```

6.5. Bayer Pattern Decoding

Functional Guide | Reference: BayerToC24

```
//////////////////////////////////////////////////////////////////////
// This code snippet shows how to perform Bayer pattern decoding. //
//////////////////////////////////////////////////////////////////////

// Images constructor
EImageBW8 bayerImage;
EImageC24 dstImage;

// ...

// Source and destination images must have the same size
dstImage.SetSize(&bayerImage);

// Convert to true color with simple interpolation, default parity assumed
EasyColor::BayerToC24(&bayerImage, &dstImage);
```

7. Deep Learning Tools

7.1. Creating a Dataset and Training a Classifier

```
////////////////////////////////////////////////////////////
// This code snippet shows how to create a dataset, train a //
// classifier and get the best performance metrics obtained //
// during the training.                                     //
////////////////////////////////////////////////////////////

// Creating dataset and classifier objects
EClassificationDataset dataset;
EClassificationDataset trainingDataset;
EClassificationDataset validationDataset;
EClassifier classifier;

// Adding images using a glob pattern
dataset.AddImages("*good*.png", "good");
dataset.AddImages("*defective*.png", "defective");

// Enabling data augmentation on the dataset
dataset.SetEnableDataAugmentation(true);

// Rotation of up to 90°
dataset.SetMaxRotationAngle(90);

// Enabling horizontal flips
dataset.SetEnableHorizontalFlip(true);

// Splitting the dataset with 80% of images for the training dataset
// and 20% for the validation dataset
dataset.SplitDataset(trainingDataset, validationDataset, 0.8f);

// Training the classifier for 50 epochs
classifier.Train(trainingDataset, validationDataset, 50);
classifier.WaitForTrainingCompletion();

// Get the best metrics obtained on the validation dataset
EClassificationMetrics bestMetrics = classifier.GetValidationMetrics(classifier.GetBestEpoch());
```

7.2. Loading a Classifier and Classifying a New Image

```
////////////////////////////////////////////////////////////
// This code snippet shows how load a trained classifier and //
// classify a new image.                                    //
////////////////////////////////////////////////////////////

// Image and classifier constructor
EClassifier classifier;
```

```
EImageBW8 srcImage;

// String and probability for the most probable result
std::string label;
float probability;

// Load classifier and image
classifier.Load(...);
srcImage.Load(...);

// Classify image
EClassificationResult result = classifier.Classify(srcImage);

// Get the most probable label
label = result.GetBestLabel();
probability = result.GetBestProbability();
```

7.3. Using Multithreading for Classification

```
//////////////////////////////////////////////////////////////
// This code snippet shows how to parallelize the           //
// classification of new images on the CPU.                 //
// This code snippet is in C++ 11 and requires a recent     //
// compiler.                                                //
//////////////////////////////////////////////////////////////


#include <queue>
#include <thread>
#include <mutex>
#include <atomic>
#include <condition_variable>

#define NUM_THREADS 4

class ImageQueue
{
public:
  ImageQueue() : complete_(false), maxQueueLength_(10) {}

  /// <summary>
  /// Adds an image to the queue.
  /// </summary>
  void addImage(EBaseROI* img)
  {
    {
      std::unique_lock<std::mutex> locker(mutex_);
      cv_.wait(locker, [&]() { return queue_.size() < maxQueueLength_; });
      queue_.push(img);
    }
    cv_.notify_one();
  }

  /// <summary>
  /// Takes an image from the queue. Returns a null pointer if the queue is complete and doesn't contain
anymore images.
  /// </summary>
  EBaseROI* takeImage()
  {
    EBaseROI* img = nullptr;
```

```
    {
      std::unique_lock<std::mutex> locker(mutex_);
      cv_.wait(locker, [&]() { return (queue_.size() > 0) || isComplete(); });

      if (queue_.size() > 0)
      {
        img = queue_.front();
        queue_.pop();
      }
    }

    cv_.notify_one();
    return img;
  }

  /// <summary>
  /// Indicates that the queue will not received any new images.
  /// </summary>
  void setComplete()
  {
    complete_ = true;
  }

  /// <summary>
  /// Checks whether the queue is empty and will not receive any new images.
  /// </summary>
  bool isComplete() const
  {
    return queue_.size() == 0 && complete_;
  }

private:
  std::atomic_bool complete_;
  std::queue<EBaseROI*> queue_;
  std::mutex mutex_;
  std::condition_variable cv_;
  int maxQueueLength_;
};

/// <summary>
/// Processing image loop.
/// </summary>
/// <param name="classifierPath">Path to the classifier used to process images</param>
/// <param name="queue">Pointer to the image queue in which to take images.</param>
void processImage(std::string segmenterPath, ImageQueue* queue)
{
  // Instanciate and load the segmenter.
  EasyDeepLearning::EClassifier classifier;
  classifier.Load(classifierPath);

  // Configure the segmenter.
  classifier.SetEnableGPU(true);
  classifier.SetBatchSize(32);

  // Loop until there are no more image to process
  while (!queue->isComplete())
  {
    // Take an image from the queue.
    EBaseROI* img = queue->takeImage();

    // If the image is null, continue the loop (and hence, check that the queue is complete).
    if (!img)
      continue;
```

```
    EasyDeepLearning::EClassificationResult r = classifier.Classify(*img);

    // Perform operations based on the result.
    …

    // Delete the image once it is processed.
    delete img;
  }
}

// …

// Create an image queue.
std::unique_ptr<ImageQueue> queue = std::make_unique<ImageQueue>();

// Declare the vector that will contain our processing threads.
std::vector<std::thread> threads;

// Create the threads
for (int i = 0; i < NUM_THREADS; i++)
{
  threads.emplace_back(processImage, "classifier.ecl", queue.get());
}

// Image acquisition loop
bool hasImage = true;
while (hasImage)
{
  EBaseROI* img;

  // Load/Create/Acquire new image
  // ...

  queue->addImage(img);
}

// Image acquisition is finished.
queue->setComplete();

// Wait for the thread to finish
for (int i = 0; i < NUM_THREADS; i++)
{
  threads[i].join();
}
```

7.4. Loading an Unsupervised Segmenter and Segmenting an Image

```
////////////////////////////////////////////////////////////////
// This code snippet shows how to load a trained              //
// unsupervised segmenter and how to segment a new image.     //
////////////////////////////////////////////////////////////////

// Image
EImageBW8 image;
image.Load(...) ;

// Segmenter
EUnsupervisedSegmenter segmenter;
```

```
segmenter.Load(...);

// Apply the segmenter on the image
EUnsupervisedSegmenterResult r = segmenter.Apply(image);

// Retrieve the segmentation map
EImageBW8 segmentationMap = r.GetSegmentationMap();
```

8. EasyObject

8.1. Constructing the Blobs

Image Encoder

Functional Guide | Reference: Encode, SetBlackLayerEncoded, SetWhiteLayerEncoded, SetMode, SetAbsoluteThreshold, GetGrayscaleSingleThresholdSegmenter

```
////////////////////////////////////////////////////////////
// This code snippet shows how to build blobs belonging to  //
// the white layer according to the minimum residue method  //
// and how to build blobs belonging to the black layer      //
// according to an absolute threshold.                      //
////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// Image encoder
EImageEncoder encoder;

// Coded image
ECodedImage2 codedImage;

// ...

// Build the blobs belonging to the white layer,
// the segmentation is based on the Minimum Residue method
encoder.Encode(srcImage, codedImage);

// Build the blobs belonging to the black layer,
// the segmentation is based on an absolute threshold (110)
Segmenters::EGrayscaleSingleThresholdSegmenter& segmenter= encoder.GetGrayscaleSingleThresholdSegmenter();
segmenter.SetBlackLayerEncoded(true);
segmenter.SetWhiteLayerEncoded(false);

segmenter.SetMode(EGrayscaleSingleThreshold_Absolute);
segmenter.SetAbsoluteThreshold(110);

encoder.Encode(srcImage, codedImage);
```

Image Segmenter

Functional Guide | Reference: SetSegmentationMethod, GetGrayscaleDoubleThresholdSegmenter, SetHighThreshold, SetLowThreshold

```
////////////////////////////////////////////////////////////
// This code snippet shows how to build blobs according to  //
// a user-defined image segmenter.                          //
```

```
////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// Image encoder
EImageEncoder encoder;

// Coded image
ECodedImage2 codedImage;

// ...

// Set the segmentation method to GrayscaleDoubleThreshold
encoder.SetSegmentationMethod(ESegmentationMethod_GrayscaleDoubleThreshold);

// Retrieve the segmenter object
Segmenters::EGrayscaleDoubleThresholdSegmenter& segmenter= encoder.GetGrayscaleDoubleThresholdSegmenter();

// Set the high and low threshold values
segmenter.SetHighThreshold(150);
segmenter.SetLowThreshold(50);

// Specify the layers to be encoded (neutral layer only)
segmenter.SetBlackLayerEncoded(false);
segmenter.SetNeutralLayerEncoded(true);
segmenter.SetWhiteLayerEncoded(false);

// Encode the image
encoder.Encode(srcImage, codedImage);
```

Holes Extraction

Functional Guide | Reference: GetHoleCount, GetHole, GetObjCount, GetObj

```
////////////////////////////////////////////////////////////
// This code snippet shows how to retrieve blobs' holes. //
////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// Image encoder
EImageEncoder encoder;

// Coded image
ECodedImage2 codedImage;

// ...

// Encode the image
encoder.Encode(srcImage, codedImage);

// Retrieve holes for all the blobs
for (unsigned int blobIndex = 0; blobIndex < codedImage.GetObjCount(); blobIndex++)
{
       EObject& blob = codedImage.GetObj(blobIndex);

       // Browse the holes of the current object
       for (unsigned int holeIndex = 0; holeIndex < blob.GetHoleCount(); holeIndex++)
       {
```

```
                // Retrieve a given hole
                EHole& hole = blob.GetHole(holeIndex);
        }
}
```

Continuous Mode

Functional Guide | Reference: SetContinuousModeEnabled, FlushContinuousMode

```
////////////////////////////////////////////////
// This code snippet shows how to build blobs  //
// in the continuous mode context.             //
////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// Image encoder
EImageEncoder encoder;

// Coded image
ECodedImage2 codedImage;

// ...

// Enable the continuous mode
encoder.SetContinuousModeEnabled(true);

// Loop to acquire the different chunks
for (int count = 0; count < MAX_COUNT ; count++)
{
        // Store the new chunk into srcImage
        // ...

        // Encode the current chunk
        encoder.Encode(srcImage, codedImage);
}

// Flush the continuous mode
encoder.FlushContinuousMode(codedImage);
```

8.2. Computing Blobs Features

Functional Guide | Reference: GetGravityCenter, GetObj

```
////////////////////////////////////////////////////////////
// This code snippet shows how to retrieve blobs' features. //
////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// Image encoder
EImageEncoder encoder;

// Coded image
ECodedImage2 codedImage;
```

```
// ...

// Encode the source image
encoder.Encode(srcImage, codedImage);

for (unsigned int index = 0; index < codedImage.GetObjCount(); index++)
{
       // Retrieve the selected blob gravity center
       EObject& blob = codedImage.GetObj(index);
       float centerX = blob.GetGravityCenter().GetX();
       float centerY = blob.GetGravityCenter().GetY();
}
```

8.3. Selecting and Sorting Blobs

Functional Guide | Reference: AddObjects, ElementCount, RemoveUsingUnsignedIntegerFeature, Sort

```
//////////////////////////////////////////////////////////
// This code snippet shows how to build blobs, select  //
// some of them and sort the selected ones.            //
//////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// Image encoder
EImageEncoder encoder;

// Coded image
ECodedImage2 codedImage;

// ...

// Encode the source image
encoder.Encode(srcImage, codedImage);

// Create a blob selection
EObjectSelection selection;
selection.AddObjects(codedImage);

// Remove the Small blobs
selection.RemoveUsingUnsignedIntegerFeature(EFeature_Area, 100, ESingleThresholdMode_Less);

// Retrieve the number of remaining blobs
unsigned int numBlobs= selection.GetElementCount();

// Sort the remaining blobs based on their area
selection.Sort(EFeature_Area, ESortDirection_Ascending);

// Retrieve the selected blobs
for (unsigned int index = 0; index < numBlobs; index++)
{
    float centerX= selection.GetElement(index).GetGravityCenterX();
    float centerY= selection.GetElement(index).GetGravityCenterY();
}
```

8.4. Using Flexible Masks

Constructing Blobs

Functional Guide | Reference: Encode

```
//////////////////////////////////////////////////////
// This code snippet shows how to build blobs inside  //
// a region defined by a flexible mask.               //
//////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageBW8 mask;

// Image encoder
EImageEncoder encoder;

// Coded image
ECodedImage2 codedImage;

// ...

// Encode the source image regions
// corresponding to the mask do care areas
encoder.Encode(srcImage, mask, codedImage);
```

Generating a Flexible Mask from an Encoded Image

Functional Guide | Reference: RenderMask

```
//////////////////////////////////////////////////////
// This code snippet shows how to generate a flexible  //
// mask from an encoded image.                        //
//////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageBW8 mask;

// Image encoder
EImageEncoder encoder;

// Coded image
ECodedImage2 codedImage;

// ...

// Encode the source image
encoder.Encode(srcImage, codedImage);

// The source image and the mask must have the same size
mask.SetSize(&srcImage);

// Create the mask based on the white layer
// of the coded image
```

```
codedImage.RenderMask(mask, 1);
```

Generating a Flexible Mask from a Blob Selection

Functional Guide | Reference: RenderMask

```
//////////////////////////////////////////////////////////
// This code snippet shows how to generate a flexible  //
// mask from a selection of blobs.                     //
//////////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageBW8 mask;

// Image encoder
EImageEncoder encoder;

// Coded image
ECodedImage2 codedImage;

// ...

// Encode the source image
encoder.Encode(srcImage, codedImage);

// The source image and the mask must have the same size
mask.SetSize(&srcImage);

// Create a blob selection
EObjectSelection selection;
selection.AddObjects(codedImage);

// Remove the Small blobs
selection.RemoveUsingUnsignedIntegerFeature(EFeature_Area, 100, ESingleThresholdMode_Less);

// Create the mask based on the blob selection
selection.RenderMask(mask);

// Sort the remaining blobs based on their area
selection.Sort(EFeature_Area, ESortDirection_Descending);

// Create the mask corresponding to the largest blob
selection.GetElement(0).RenderMask(mask);
```

8.5. Using the Object Template Matcher

Functional Guide | Reference: EObjectTemplateMatcher

```
//////////////////////////////////////////////////////////////
// This code snippet shows how to use EObjectTemplateMatcher//
// for alignment and template matching                      //
//////////////////////////////////////////////////////////////

// Encode the template image
EImageEncoder encoder;
ECodedImage2 coded_img;
```

```
EImageBW8 template_img;
encoder.Encode(template_img, coded_img);

// Initialize EObjectTemplateMatcher
EObjectTemplateMatcher object_matcher;
object_matcher.SetEnableAlignment(true); // optional
object_matcher.SetMaximumDistance(60);   // optional

// set the template
object_matcher.BuildTemplate(coded_img);

// Encode the test image
EImageBW8 test_img;
encoder.Encode(test_img, coded_img);

// Build a selection of test objects
EObjectSelection object_select;
object_select.AddObjects(coded_img);
object_select.RemoveUsingUnsignedIntegerFeature(EFeature_Area, 10, ESingleThresholdMode_Less);  // optional
filter


// Perform the alignment and the matching
object_matcher.SortSelection(object_select);

// Display the number of matches
std::cout << object_matcher.GetNumberOfPairedObjects() << " paired objects" << std::endl;

// Retrieve the template indexes for each selection object
std::vector<int> template_indexes = object_matcher.GetTemplateIndexes();
```

9. EasyMatch

9.1. Pattern Learning

Functional Guide | Reference: LearnPattern

```
////////////////////////////////////////////////////
// This code snippet shows how to learn a pattern  //
// defined by a region of interest (ROI).          //
////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// ROI constructor
EROIBW8 pattern;

// EMatcher constructor
EMatcher matcher;

// ...

// Attach the ROI to the source image
// and set its position
pattern.Attach(&srcImage);
pattern.SetPlacement(214, 52, 200, 200);

// Learn the pattern
matcher.LearnPattern(&pattern);
```

9.2. Setting Search Parameters

Functional Guide | Reference: SetMaxPositions, SetMinAngle, SetMaxAngle, SetMinScore, SetInterpolate, Save

```
////////////////////////////////////////////////////////
// This code snippet shows how to tune pattern matching //
// search parameters and save them into a file.         //
////////////////////////////////////////////////////////

// Image constructor
EImageBW8 pattern;

// EMatcher constructor
EMatcher matcher;

// ...

// Learn the pattern
matcher.LearnPattern(&pattern);

// Set the maximum number of occurrences
```

```
matcher.SetMaxPositions(5);

// Set the rotation tolerances
matcher.SetMinAngle(-20.f);
matcher.SetMaxScale(20.f);

// Enable sub-pixel accuracy
matcher.SetInterpolate(true);

// Set the minimum score
matcher.SetMinScore(0.70f);

// Save the matching context into a model file
matcher.Save("myModel.mch");
```

9.3. Pattern Matching and Retrieving Results

Functional Guide | Reference: Load, Match, GetNumPositions, GetPosition

```
//////////////////////////////////////////////////
// This code snippet shows how to perform pattern  //
// matching operations and retrieve the results.   //
//////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// EMatcher constructor
EMatcher matcher;

// ...

// Load a model file
matcher.Load("myModel.mch");

// Perform the matching
matcher.Match(&srcImage);

// Retrieve the number of occurrences
int numOccurrences= matcher.GetNumPositions();

// Retrieve the first occurrence
EMatchPosition myOccurrence= matcher.GetPosition(0);

// Retrieve its score and position
float score= myOccurrence.Score;
float centerX= myOccurrence.CenterX;
float centerY= myOccurrence.CenterY;
```

9.4. Pattern Learning with ERegion

Functional Guide | Reference: LearnPattern

```
//////////////////////////////////////////////////
// This code snippet shows how to learn a pattern    //
// whose region of interest is defined by an ERegion //
```

```
//////////////////////////////////////////////////////

EImageBW8 srcImage;
EROIBW8 pattern;
EMatcher matcher;
// ...
// Attach the ROI to the source image and set its position
pattern.Attach(&srcImage);
pattern.SetPlacement(214, 52, 200, 200);

// pattern is a 200*200 square but here we are only
// interested in the inner circle

// OLD method (warning, advanced learning is not compatible with this)
matcher.SetDontCareThreshold(1);
// must paint the part of pattern we are not interested in in black
matcher.LearnPattern(&pattern);

// NEW method (compatible with advanced learning)
ECircleRegion region(100.f, 100.f, 100.f);
matcher.LearnPattern(&pattern, region);
```

10. EasyFind

10.1. Pattern Learning

Functional Guide | Reference: Learn

```
//////////////////////////////////////////////////////
// This code snippet shows how to learn a pattern  //
// defined by a region of interest (ROI).          //
//////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// ROI constructor
EROIBW8 pattern;

// EPatternFinder constructor
EPatternFinder finder;

// ...

// Attach the ROI to the source image
// and set its position
pattern.Attach(&srcImage);
pattern.SetPlacement(214, 52, 200, 200);

// Learn the pattern
finder.Learn(&pattern);
```

10.2. Setting Search Parameters

Functional Guide | Reference: SetMaxInstances, SetAngleTolerance, SetMinScore, Save

```
//////////////////////////////////////////////////////////
// This code snippet shows how to tune pattern finding  //
// search parameters and save them into a file.         //
//////////////////////////////////////////////////////////

// Image constructor
EImageBW8 pattern;

// EPatternFinder constructor
EPatternFinder finder;

// ...

// Learn the pattern
finder.Learn(&pattern);

// Set the maximum number of occurrences
finder.SetMaxInstances(5);
```

```
// Set the rotation tolerances
finder.SetAngleTolerance(20.f);

// Set the minimum score
finder.SetMinScore(0.70f);

// Save the finding context into a model file
finder.Save("myModel.fnd");
```

10.3. Pattern Finding and Retrieving Results

Functional Guide | Reference: Load, Find, GetScore, GetCenter

```
//////////////////////////////////////////////////////
// This code snippet shows how to perform pattern  //
// finding operations and retrieve the results.    //
//////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// EPatternFinder constructor
EPatternFinder finder;

// EFoundPattern constructor
std::vector<EFoundPattern> foundPattern;

// ...

// Load a model file
finder.Load("myModel.fnd");

// Perform the pattern finding
foundPattern= finder.Find(&srcImage);

// Retrieve the number of instances
int numInstances= (int)foundPattern.size();

// Retrieve the score and the
// position of the first instance
float score= foundPattern[0].GetScore();
float centerX= foundPattern[0].GetCenter().GetX();
float centerY= foundPattern[0].GetCenter().GetY();
```

10.4. Learning Using a DXF File

Functional Guide | Reference: LoadDXF, Find

```
//////////////////////////////////////////////////////
// This code snippet shows how to perform          //
// pattern learning and finding operations         //
// using a DXF file.                               //
//////////////////////////////////////////////////////
// Image constructor
EImageBW8 srcImage;
```

```
// EPatternFinder constructor
EPatternFinder finder;
// EVectorModel constructor
EVectorModel myModel;
// Load the model from a dxf file
myModel.LoadDXF("myModel.dxf");
// Learn the model
finder.Learn(myModel);
// EFoundPattern constructor
std::vector<EFoundPattern> foundPattern;
// ...
// Perform the pattern finding
foundPattern = finder.Find(&srcImage);
```

10.5. Learning Using an EPolygonShape

Functional Guide | Reference: SetPolygon, Find

```
//////////////////////////////////////////////////
// This code snippet shows how to perform       //
// pattern learning and finding operations      //
// using EPolygonShape to define the model.     //
//////////////////////////////////////////////////
// Image constructor
EImageBW8 srcImage;
// EPatternFinder constructor
EPatternFinder finder;
// EVectorModel constructor
EVectorModel myModel;
// Get the root EFrameShape of the model
EFrameShape& shapeMother = myModel.GetRoot();
// EPolygonShape constructor
EPolygonShape polygon;
// Define the vertices of a polygon
std::vector<EPoint> vertices = { {0.f, 0.f}, {1.f, 0.f}, {1.f, 1.f}, {0.f, 1.f} };
// Define the EPolygonShape
polygon.SetPolygon(EPolygon(vertices, true));
// Attach the EPolygonShape to the root EFrameShape
polygon.Attach(&shapeMother);
// Sets the polarity of the EPolygonShape
polygon.SetProperty("polarity", "direct");
// Learn the model
finder.Learn(myModel);
// EFoundPattern constructor
std::vector<EFoundPattern> foundPattern;
// ...
// Perform the pattern finding
foundPattern = finder.Find(&srcImage);
```

11. EasyGauge

11.1. Point Location

Functional Guide | Reference: SetTransitionType, SetTransitionChoice, SetCenterXY, SetTolerance, Measure, GetMeasuredPoint, GetX, GetY

```
//////////////////////////////////////////////////////////////////
// This code snippet shows how to create a point location tool, //
// adjust the transition parameters, set the nominal gauge      //
// position, perform the measurement and retrieve the result.   //
//////////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// EPointGauge constructor
EPointGauge pointGauge;

// Adjust the transition parameters
pointGauge.SetTransitionType(ETransitionType_Wb);
pointGauge.SetTransitionChoice(ETransitionChoice_Closest);

// Set the gauge nominal position
pointGauge.SetCenterXY(256.f, 256.f);

// Set the gauge length to 100 units and the angle to 45°
pointGauge.SetTolerances(100.f, 45.f);

// Measure
pointGauge.Measure(&srcImage);

// Get the measured point coordinates
float measuredX = pointGauge.GetMeasuredPoint().GetX();
float measuredY = pointGauge.GetMeasuredPoint().GetY();

// Save the point gauge measurement context
pointGauge.Save("myPointGauge.gge");
```

11.2. Line Fitting

Functional Guide | Reference: SetTransitionType, SetTransitionChoice, SetTransitionIndex, SetLine, Measure, GetMeasuredLine, GetOrg, GetEnd

```
//////////////////////////////////////////////////////////////////
// This code snippet shows how to create a line measurement tool, //
// adjust the transition parameters, set the nominal gauge        //
// position, perform the measurement and retrieve the result.     //
//////////////////////////////////////////////////////////////////

// Image constructor
```

```
EImageBW8 srcImage;

// ELineGauge constructor
ELineGauge lineGauge;

// Adjust the transition parameters
lineGauge.SetTransitionType(ETransitionType_Bw);
lineGauge.SetTransitionChoice(ETransitionChoice_NthFromEnd);
lineGauge.SetTransitionIndex(2);

// Set the line fitting gauge position,
// length (50 units) and orientation (20°)
EPoint center(256.f, 256.f);
ELine line(center, 50.f, 20.f);
lineGauge.SetLine(line);

// Measure
lineGauge.Measure(&srcImage);

// Get the origin and end point coordinates of the fitted line
EPoint originPoint = lineGauge.GetMeasuredLine().GetOrg();
EPoint endPoint = lineGauge.GetMeasuredLine().GetEnd();

// Save the point gauge measurement context
lineGauge.Save("myLineGauge.gge");
```

11.3. Circle Fitting

Functional Guide | Reference: SetTransitionType, SetTransitionChoice, SetCircle, Measure, GetMeasuredCircle, GetCenter, GetRadius

```
//////////////////////////////////////////////////////////////////////
// This code snippet shows how to create a circle measurement tool, //
// adjust the transition parameters, set the nominal gauge          //
// position, perform the measurement and retrieve the result.       //
//////////////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// ECircleGauge constructor
ECircleGauge circleGauge;

// Adjust the transition parameters
circleGauge.SetTransitionType(ETransitionType_Bw);
circleGauge.SetTransitionChoice(ETransitionChoice_LargestAmplitude);

// Set the Circle fitting gauge position, diameter (50 units),
// starting angle (10°), and amplitude (270°)
EPoint center(256.f, 256.f);
ECircle circle(center, 50.f, 10.f, 270.f);
circleGauge.SetCircle(circle);

// Measure
circleGauge.Measure(&srcImage);

// Get the center point coordinates and the radius of the fitted circle
float centerX = circleGauge.GetMeasuredCircle().GetCenter().GetX();
float centerY = circleGauge.GetMeasuredCircle().GetCenter().GetY();
float radius = circleGauge.GetMeasuredCircle().GetRadius();
```

```
// Save the point gauge measurement context
circleGauge.Save("myCircleGauge.gge");
```

11.4. Rectangle Fitting

Functional Guide | Reference: SetTransitionType, SetTransitionChoice, SetRectangle, Measure, GetMeasuredRectangle, GetSizeX, GetSizeY, GetAngle

```
//////////////////////////////////////////////////////////////////////////
// This code snippet shows how to create a rectangle measurement tool, //
// adjust the transition parameters, set the nominal gauge position,   //
// perform the measurement and retrieve the result.                    //
//////////////////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// ERectangleGauge constructor
ERectangleGauge rectangleGauge;

// Adjust the transition parameters
rectangleGauge.SetTransitionType(ETransitionType_Bw);
rectangleGauge.SetTransitionChoice(ETransitionChoice_LargestAmplitude);

// Set the rectangle fitting gauge position,
// size (50x30 units) and orientation (15°)
EPoint center(256.f, 256.f);
ERectangle rectangle(center, 50.f, 30.f, 15.f);
rectangleGauge.SetRectangle(rectangle);

// Measure
rectangleGauge.Measure(&srcImage);

// Get the size and the rotation angle of the fitted rectangle
float sizeX = rectangleGauge.GetMeasuredRectangle().GetSizeX();
float sizeY = rectangleGauge.GetMeasuredRectangle().GetSizeY();
float angle = rectangleGauge.GetMeasuredRectangle().GetAngle();

// Save the point gauge measurement context
rectangleGauge.Save("myRectangleGauge.gge");
```

11.5. Wedge Fitting

Functional Guide | Reference: SetTransitionType, SetTransitionChoice, SetWedge, Measure, GetMeasuredWedge, GetInnerRadius, GetOuterRadius

```
//////////////////////////////////////////////////////////////////////
// This code snippet shows how to create a wedge measurement tool, //
// adjust the transition parameters, set the nominal gauge         //
// position, perform the measurement and retrieve the result.      //
//////////////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;
```

```
// EWedgeGauge constructor
EWedgeGauge wedgeGauge;

// Adjust the transition parameters
wedgeGauge.SetTransitionType(ETransitionType_Bw);
wedgeGauge.SetTransitionChoice(ETransitionChoice_NthFromBegin);
wedgeGauge.SetTransitionIndex(0);

// Set the wedge fitting gauge position, diameter (50 units),
// breadth (-25 units), starting angle (0°) and amplitude (270°)
EPoint center(256.f, 256.f);
EWedge wedge(center, 50.f, -25.f, 0.f, 270.f);
wedgeGauge.SetWedge(wedge);

// Measure
wedgeGauge.Measure(&srcImage);

// Get the inner and outer radius of the fitted wedge
float innerRadius = wedgeGauge.GetMeasuredWedge().GetInnerRadius();
float outerRadius = wedgeGauge.GetMeasuredWedge().GetOuterRadius();

// Save the point gauge measurement context
wedgeGauge.Save("myWedgeGauge.gge");
```

11.6. Gauge Grouping

Gauge Hierarchy

Functional Guide | Reference: Attach, SetName, Save

```
////////////////////////////////////////////////////////////
// This code snippet shows how to create a gauge hierarchy  //
// and save it into a file.                                 //
////////////////////////////////////////////////////////////

// EWorldShape constructor
EWorldShape worldShape;

// Gauges constructor
ERectangleGauge rectangleGauge;
ECircleGauge circleGauge1, circleGauge2;

// ...

// Attach the rectangle gauge to the EWorldShape
rectangleGauge.Attach(&worldShape);

// Attach the circle gauges to the rectangle gauge
circleGauge1.Attach(&rectangleGauge);
circleGauge2.Attach(&rectangleGauge);

// Set the first circle gauge name
circleGauge1.SetName("myCircleGauge1");

// ...

// Save worldShape together with its daughters
worldShape.Save("myWorldShape.gge", true);
```

Complex Measurement

Functional Guide | Reference: Load, GetNumDaughters, Process, GetDaughter, GetShapeNamed

```
////////////////////////////////////////////////////////////
// This code snippet shows how to trigger the measurement //
// of a whole gauge hierarchy and retrieve the results.   //
////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// EWorldShape constructor
EWorldShape worldShape;

// Load the EWorldShape together with its daughters
worldShape.Load("myWorldShape.gge", true);

// Retrieve the number of worldShape's daughters
int numDaughters= worldShape.GetNumDaughters();

// ...

// Trigger the measurement of all the
// gauges attached to the EWorldShape
worldShape.Process(&srcImage, true);

// Retrieve the measurement result of
// the first daughter (a rectangle gauge)
ERectangleGauge* rectangleGauge= (ERectangleGauge*)worldShape.GetDaughter(0);
float sizeX= rectangleGauge->GetMeasuredRectangle().GetSizeX();

// Retrieve the measurement result of a
// daughter gauge called "myCircleGauge1"
ECircleGauge* circleGauge= (ECircleGauge*)worldShape.GetShapeNamed("myCircleGauge1");
EPoint center= circleGauge->GetMeasuredCircle().GetCenter();
```

11.7. Calibration using EWorldShape

Functional Guide | Reference

Calibration by Guesswork

Functional Guide | Reference: SetSensor, GetXResolution, GetYResolution

```
////////////////////////////////////////////////////////////
// This code snippet shows how to perform a calibration //
// by guesswork.                                        //
////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// EWorldShape constructor
EWorldShape worldShape;
```

```
// ...

// Compute the calibration coefficients
// Field of view: 32x24 mm
worldShape.SetSensor(srcImage.GetWidth(), srcImage.GetHeight(), 32.f, 24.f);

// Retrieve the spatial resolution
float resolutionX= worldShape.GetXResolution();
float resolutionY= worldShape.GetYResolution();
```

Landmark-Based Calibration

Functional Guide | Reference: EmptyLandmarks, AddLandmark, Calibrate

```
////////////////////////////////////////////////////////////
// This code snippet shows how to perform a landmark-based //
// calibration.                                            //
////////////////////////////////////////////////////////////

// EWorldShape constructor
EWorldShape worldShape;

// ...

// Reset the calibration context
worldShape.EmptyLandmarks();

// Loop on the landmarks
for(int index= 0; index < numLandmarks; index++)
{
    // Get the I-th landmark as a pair of EPoint(x, y)
    EPoint sensorPoint, worldPoint;

    // Retrieve and store the relevant data into worldPoint and sensorPoint
    // ...

    // Add the I-th pair
    worldShape.AddLandmark(sensorPoint, worldPoint);
}

// Perform the calibration
worldShape.Calibrate(ECalibrationMode_Skewed);
```

Dot Grid-Based Calibration

Functional Guide | Reference: EmptyLandmarks, AddPoint, RebuildGrid, AutoCalibrate

```
////////////////////////////////////////////////////////////
// This code snippet shows how to perform a dot grid-based //
// calibration.                                            //
////////////////////////////////////////////////////////////

// EWorldShape constructor
EWorldShape worldShape;

// ...
```

```
// Reset the calibration context
worldShape.EmptyLandmarks();

// Loop on the dots
for(int index= 0; index < numDots; index++)
{
    // Get the I-th dot as an EPoint(x, y)
    EPoint dotPoint;

    // Retrieve and store the relevant data into dotPoint
    // ...

    // Add the I-th dot
    worldShape.AddPoint(dotPoint);
}

// Reconstruct the grid topology
// pitch X and Y = 5 units
worldShape.RebuildGrid(5, 5);

// Perform the calibration
// the calibration modes are computed automatically
worldShape.AutoCalibrate(true);
```

Coordinates Transform

Functional Guide | Reference: SensorToWorld, WorldToSensor

```
////////////////////////////////////////////////////////////
// This code snippet shows how to convert coordinates from //
// the Sensor space to the World space and conversely.     //
////////////////////////////////////////////////////////////

// EWorldShape constructor
EWorldShape worldShape;

// EPoint constructor
EPoint sensor;
EPoint world;

// ...

// Perform the calibration
worldShape.Calibrate(ECalibrationMode_Scaled | ECalibrationMode_Skewed);

// Retrieve the world coordinates of a point, knowing its sensor coordinates
world= worldShape.SensorToWorld(sensor);

// Retrieve the sensor coordinates of a point, knowing its world coordinates
sensor= worldShape.WorldToSensor(world);
```

Image Unwarping

Functional Guide | Reference: SetupUnwarp, Unwarp

```
////////////////////////////////////////////////////////
// This code snippet shows how to unwarp an image based //
// of the computed calibration coefficients.           //
```

```
////////////////////////////////////////////////////////

// Images constructor
EImageBW8 srcImage;
EImageBW8 dstImage;

// EWorldShape constructor
EWorldShape worldShape;

// Lookup table constructor
EUnwarpingLut lut;

// ...

// Perform the calibration
worldShape.Calibrate(ECalibrationMode_Tilted | ECalibrationMode_Radial);

// Setup the lookup table for unwarping
worldShape.SetupUnwarp(&lut, &srcImage, true);

// Perform the image unwarping
worldShape.Unwarp(&lut, &srcImage, &dstImage, true);
```

12. EasyOCR

12.1. Learning Characters

Functional Guide | Reference: NewFont, SetTextColor, SetMinCharWidth, SetMaxCharWidth, SetMinCharHeight, SetMaxCharHeight, SetNoiseArea, LearnPatterns, BuildObjects, FindAllChars, Save

```
//////////////////////////////////////////////////////
// This code snippet shows how to learn characters  //
// based on an image featuring a known text and     //
// save the corresponding font file.                //
//////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// EOCR constructor
EOCR ocr;

// Text to be learned (all digits)
// Assuming the image contains this text
const std::string text= "0123456789";

// ...

// Create a new fon
ocr.NewFont(8, 11);

// Adjust the segmentation parameters
ocr.SetTextColor(EOCRColor_BlackOnWhite);
ocr.SetMinCharWidth(15);
ocr.SetMinCharWidth(50);
ocr.SetMinCharHeight(15);
ocr.SetMinCharHeight(75);
ocr.SetNoiseArea(15);

// Segment the characters
ocr.BuildObjects(&srcImage);
ocr.FindAllChars(&srcImage);

// Learn the characters
ocr.LearnPatterns(&srcImage, text, EOCRClass_Digit);

// Save the font into a file
ocr.Save("myFont.ocr");
```

12.2. Recognizing Characters

Functional Guide | Reference: Load, Recognize

```
//////////////////////////////////////////////////////
// This code snippet shows how to load a font file,  //
// perform a default character recognition operation //
// and perform a character recognition operation     //
// using a class filter.                             //
//////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// EOCR constructor
EOCR ocr;

// Load the font file
ocr.Load("myFont.ocr");

// ...

// Recognize the characters
std::string text= ocr.Recognize(&srcImage, 10, EOCRClass_AllClasses);

// Alternatively
// Define the character filter (2 letters and 3 digits)
std::vector<UINT32> charFilter;
charFilter.push_back(EOCRClass_UpperCase);
charFilter.push_back(EOCRClass_UpperCase);
charFilter.push_back(EOCRClass_Digit);
charFilter.push_back(EOCRClass_Digit);
charFilter.push_back(EOCRClass_Digit);

// Recognize the characters with class filtering
text= ocr.Recognize(&srcImage, 10, charFilter);
```

13. EasyOCR2

13.1. Detecting Characters

```
//////////////////////////////////////////////////////
// This code snippet shows how to detect characters  //
// in an image, using a few parameters and a topology //
//////////////////////////////////////////////////////
// Load an Image
EImageBW8 image;
image.Load("image.tif");
// Attach a ROI to the image
EROIBW8 roi;
roi.Attach(&image, 50, 224, 340, 96);
// Create an EOCR2 instance
EOCR2 ocr2;
// Set the expected character sizes
ocr2.SetCharsWidthRange(EIntegerRange(25,25));
ocr2.SetCharsHeight(37);
// Set the text polarity, in this case WhiteOnBlack
ocr2.SetTextPolarity(EasyOCR2TextPolarity_WhiteOnBlack);
// Set the topology
ocr2.SetTopology(".{10}\n.{3} .{4}");
// Detect the text in the image. The output Text structure contains:
// - an individual textbox for each character
// - an individual bitmap image for each character
// - a threshold value to binarize the bitmap image for each character
// All structured in a hierarchy with Lines -> Words -> Characters
EOCR2Text text = ocr2.Detect(roi);
```

The image used in this code snippet

13.2. Learning Characters

```
//////////////////////////////////////////////////////
// This code snippet shows how to learn characters  //
// based on an image featuring a known text and     //
// save the corresponding character database        //
//////////////////////////////////////////////////////
// Load an Image
```

```
EImageBW8 image;
image.Load("image.tif");
// Attach a ROI to the image
EROIBW8 roi;
roi.Attach(&image, 50, 224, 340, 96);

// Create an EOCR2 instance
EOCR2 ocr2;

// Set the required parameters
ocr2.SetCharsWidthRange(EIntegerRange(25,25));
ocr2.SetCharsHeight(37);
ocr2.SetTextPolarity(EasyOCR2TextPolarity_WhiteOnBlack);
ocr2.SetTopology(".{10}\n.{3} .{4}");

// Learn from the reference image:
//   1) Detect the text in the image
EOCR2Text text = ocr2.Detect(roi);
//   2) Set the true values of the text
text.SetText("Bt121KPJ80\n786 9512");
//   3) Add the characters to the character database
ocr2.Learn(text);

// Save the character database
ocr2.SaveCharacterDatabase("myDB.o2d");

// Alternatively, save the model file.
// This will store the character database and the parameter settings
ocr2.Save("myModel.o2m");
```

The image used in this code snippet

13.3. Reading Characters

Reading Using TrueType Fonts

```
//////////////////////////////////////////////////////////
// This code snippet shows how to                       //
// - create a character database from TrueType fonts  //
// - read the text in an image                          //
//////////////////////////////////////////////////////////

// Load an image
EImageBW8 image;
image.Load("image.tif");
```

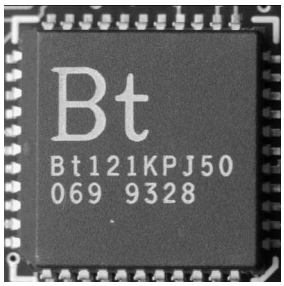
```
// Attach an ROI
EROIBW8 roi;
roi.Attach(&image, 50, 224, 340, 96);

// Create an EOCR2 instance
EOCR2 ocr2;

// Set the required parameters
ocr2.SetCharsWidthRange(EIntegerRange(25,25));
ocr2.SetCharsHeight(37);
ocr2.SetTopology("[LN]{10}\nN{3} N{4}");
ocr2.SetTextPolarity(EasyOCR2TextPolarity_WhiteOnBlack);

// Add TrueType character to the character database
ocr2.AddCharactersToDatabase("C:\\Windows\\Fonts\\calibrib.ttf");
ocr2.AddCharactersToDatabase("C:\\Windows\\Fonts\\yugothb.ttc");

// Read text from the image
std::string result = ocr2.Read(roi);
```



The image used in this code snippet

Reading Using EOCR2 Character Database

```
//////////////////////////////////////////////////////
// This code snippet shows how to                   //
// - load a pre-made character database             //
// - read the text in an image                      //
//////////////////////////////////////////////////////

// Load an image
EImageBW8 image;
image.Load("image.tif");

// Attach an ROI
EROIBW8 roi;
roi.Attach(&image, 50, 224, 340, 96);

// Create an EOCR2 instance
EOCR2 ocr2;

// Set the required parameters
ocr2.SetCharsWidthRange(EIntegerRange(25,25));
ocr2.SetCharsHeight(37);
ocr2.SetTopology("[LN]{10}\nN{3} N{4}");
ocr2.SetTextPolarity(EasyOCR2TextPolarity_WhiteOnBlack);

// Add a pre-made character database to the EOCR2 instance
ocr2.AddCharactersToDatabase("myDB.o2d");
```

```
// Read text from the image
std::string result = ocr2.Read(roi);
```

Reading Using EOCR2 Model File

```
//////////////////////////////////////////////////////
// This code snippet shows how to                   //
// - load a pre-made model file                     //
// - read the text in an image                      //
//////////////////////////////////////////////////////

// Load an image
EImageBW8 image;
image.Load("image.tif");

// Attach an ROI
EROIBW8 roi;
roi.Attach(&image, 50, 224, 340, 96);

// Create an EOCR2 instance
EOCR2 ocr2;

// Load a pre-made model file, this will:
// - (re)set all parameters
// - add the character database in the model file to the EOCR2 instance
ocr2.Load("myModel.o2m");

// Read text from the image
std::string result = ocr2.Read(roi);
```

13.4. View Bitmap

```
//////////////////////////////////////////////
// This code snippet shows how to inspect the //
// characters in a character database         //
//////////////////////////////////////////////
// Create an EOCR2 instance
EOCR2 ocr2;

// Load the character database
ocr2.AddCharactersToDatabase("database.o2d");
// Extract the character database
EOCR2CharacterDatabase db = ocr2.GetCharacterDatabase();

// Select the character that we are interested in (e.g. the third one)
EOCR2DatabaseCharacter chr = db.GetCharacter(2);
// Extract the bitmap for that character
EImageBW8 img = chr.GetBitmap();
```

14. EasyBarCode

14.1. Reading a Bar Code

Functional Guide | Reference: Read

```
////////////////////////////////////////////////////
// This code snippet shows how to read a bar code  //
////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// Bar code reader constructor
EBarCode reader;

// String for the decoded bar code
std::string result;

// ...

// Read the source image
result = reader.Read(&srcImage);
```

14.2. Reading a Bar Code Following a Given Symbology

Functional Guide | Reference: SetAdditionalSymbologies, Detect, Decode

```
////////////////////////////////////////////////////////////////
// This code snippet shows how to enable a given symbology,  //
// enable the checksum verification, perform the bar code    //
// detection and retrieve the decoded string.                //
////////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// Bar code reader constructor
EBarCode reader;

// String for the decoded bar code
std::string result;

// ...

// Disable all standard symbologies
reader.SetStandardSymbologies(0);

// Enable the Code32 symbology only
```

```
reader.SetAdditionalSymbologies(ESymbologies_Code32);

// Enable checksum verification
reader.SetVerifyChecksum(true);

// Detect all possible meanings of the bar code
reader.Detect(&srcImage);

// Retrieve the number of symbologies for
// which the decoding process was successful
int numDecoded = reader.GetNumDecodedSymbologies();

if(numDecoded > 0)
{
    // Decode the bar code according to the Code32 symbology
    result = reader.Decode(ESymbologies_Code32);
}
```

14.3. Reading a Bar Code of Known Location

Functional Guide | Reference: SetCenterXY, SetReadingSize

```
//////////////////////////////////////////////////////////
// This code snippet shows how to specify the bar code  //
// position and perform the bar code reading.           //
//////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// Bar code reader constructor
EBarCode reader;

// String for the decoded bar code
std::string result;

// ...

// Disable automatic bar code detection
reader.SetKnownLocation(TRUE);

// Set the bar code position
reader.SetCenterXY(450.0f, 400.0f);
reader.SetSize(250.0f, 110.0f);
reader.SetReadingSize(1.15f, 0.5f);

// Read the bar code at the specified location
result = reader.Read(&srcImage);
```

14.4. Reading a Mail Bar Code

Functional Guide | Reference: Read

```
//////////////////////////////////////////////////////////
// This code snippet shows how to read Mail Barcodes  //
// and retrieve the decoded data.                     //
```

```
//////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;
// Mail bar code reader constructor
EMailBarcodeReader reader;


// Select expected symbologies and orientations (optional)
reader.SetExpectedSymbologies(...);
reader.SetExpectedOrientations(...);
// ...
// Read
std::vector<EMailBarcode> codes = reader.Read(srcImage);
// Retrieve the data included in found mail barcodes
for (unsigned int index= 0; index < codes.size(); index++)
{
    std::string text = codes[index].GetText();
    std::vector<EStringPair> components = codes[index]. GetComponentStrings();
}
```

15. EasyBarCode2

15.1. Reading a Bar Code

Functional Guide | Reference: Read, SetMaxNumCodes, GetDecodedString

```
////////////////////////////////////////////////
//This code snippet shows how to read bar codes //
////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// Bar code reader constructor
EasyBarCode2::EBarCodeReader reader;

// Set the max number of bar codes to find
reader.SetMaxNumCodes(...);

// Read the barcodes in the source image
std::vector<EasyBarCode2::EBarCode> results = reader.Read(srcImage);

// Get decoded string
std::string decodedString = results[0].GetDecodedString();
```

15.2. Reading a Bar Code of a Specific Symbology

Functional Guide | Reference: Read, DisableAllSymbologies, EnableSymbology, GetDecodedString

```
////////////////////////////////////////////////
// This code snippet shows how to read bar codes //
// of a specific symbology                       //
////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// Bar code reader constructor
EasyBarCode2::EBarCodeReader reader;


// Set the max number of bar codes to find
reader.SetMaxNumCodes(...);


// Set symbology to use
reader.DisableAllSymbologies();
reader.EnableSymbology(...);


// Read the barcodes in the source image
std::vector<EasyBarCode2::EBarCode> results = reader.Read(srcImage);
```

```
// Get decoded string
std::string decodedString = results[0].GetDecodedString();
```

15.3. Reading a Grid of Bar Codes

Functional Guide | Reference: Read, EBarCodeGrid

```
//////////////////////////////////////////////////////////////
// This code snippet shows how to read bar codes that are    //
// disposed in a 5 by 3 grid with 10% overlap between grid   //
// cells                                                     //
//////////////////////////////////////////////////////////////

EImageBW8 srcImage;
ERectangleRegion gridRegion;
EasyBarCode2::EBarCodeReader reader;

// Reading succeeds
EasyBarCode2::EBarCodeGrid grid = reader.Read(srcImage, gridRegion, 3, 5, 0.1f);

std::vector<EasyBarCode2::EBarCode> codesMiddleCell = grid.GetResults(1, 2);
```

15.4. Learning a Bar Code

Functional Guide | Reference: Read, Learn

```
//////////////////////////////////////////
// This code snippet shows how to learn //
// from a given set of images           //
//////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// Set of images that are close to srcImage
std::vector<EImageBW8> learningImages;

// Bar code reader constructor
EasyBarCode2::EBarCodeReader reader;

// Reading fails
// std::vector<EasyBarCode2::EBarCode> barcodesFail = reader.Read(srcImage);

// Learns the learningImages
reader.Learn(learningImages);

// Reading succeeds
std::vector<EasyBarCode2::EBarCode> barcodes = reader.Read(srcImage);
```

15.5. Grading a Bar Code

Functional Guide | Reference: Read, SetComputeGrading, GetGradingParameters

```
//////////////////////////////////////////
// This code snippet shows how to compute //
// the ISO15416 grading of a barcode      //
//////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// Bar code reader constructor
EasyBarCode2::EBarCodeReader reader;

// Enables grading computation
reader.SetComputeGrading(true);

// Read the image and retrieve the computed grade
std::vector<EasyBarCode2::EBarCode> barcodes = reader.Read(srcImage);
if (!barcodes.empty())
{
  EasyBarCode2::EBarCodeGradingParameters grades = barcodes[0].GetGradingParameters();

  // global grade on a range from 0 to 40
  grades.GlobalGrade;

  // global grade on a range from F to A
  EasyBarCode2::EBarCodeGradingParameters::ConvertToAlphabeticGrade(grades.GlobalGrade);
}
```

16. EasyMatrixCode

16.1. Reading a Data Matrix Code

Functional Guide | Reference: Read, GetDecodedString

```
////////////////////////////////////////////////////////////
// This code snippet shows how to read a data matrix code  //
// and retrieve the decoded string.                        //
////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// Matrix code reader constructor
EMatrixCodeReader reader;

// Matrix code constructor
EMatrixCode mxCode;

// String for the decoded information
std::string result;

// ...

// Read the source image
mxCode = reader.Read(srcImage);

// Retrieve the decoded string
result = mxCode.GetDecodedString();
```

16.2. Learning a Data Matrix Code

Functional Guide | Reference: SetLearnMaskElement, Learn, Read, GetDecodedString

```
////////////////////////////////////////////////////////////
// This code snippet shows how to learn a given data matrix //
// code type (except its flipping status), perform the      //
// reading and retrieve the decoded string.                 //
////////////////////////////////////////////////////////////

// Images constructor
EImageBW8 model;
EImageBW8 srcImage;

// Matrix code reader constructor
EMatrixCodeReader reader;

// Matrix code constructor
EMatrixCode mxCode;

// String for the decoded information
```

```
std::string result;

// ...

// Tell the reader not to take the flipping into account when learning
reader.SetLearnMaskElement(ELearnParam_Flipping, false);

// Learn the model
reader.Learn(model);

// Read the source image
mxCode = reader.Read(srcImage);

// Retrieve the decoded string
result = mxCode.GetDecodedString();
```

16.3. Tuning the Search Parameters

Functional Guide | Reference: GetSearchParams, ClearLogicalSize, AddLogicalSize, ClearFamily, AddFamily, Read, GetDecodedString

```
////////////////////////////////////////////////////////////////
// This code snippet shows how to explicitly specify the data //
// matrix code logical size and family, perform the reading   //
// and retrieve the decoded string.                           //
////////////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// Matrix code reader constructor
EMatrixCodeReader reader;

// Matrix code constructor
EMatrixCode mxCode;

// String for the decoded information
std::string result;

// ...

// Remove the default logical sizes
reader.GetSearchParams().ClearLogicalSize();

// Add the 15x15 and 17x17 logical sizes
reader.GetSearchParams().AddLogicalSize(ELogicalSize__15x15);
reader.GetSearchParams().AddLogicalSize(ELogicalSize__17x17);

// Remove the default families
reader.GetSearchParams().ClearFamily();

// Add the ECC050 family
reader.GetSearchParams().AddFamily(EFamily_ECC050);

// Read the source image
mxCode = reader.Read(srcImage);

// Retrieve the decoded string
result = mxCode.GetDecodedString();
```

16.4. Grading a Data Matrix Code

Functional Guide | Reference: Read, GetComputeGrading, GetAxialNonUniformityGrade, GetContrastGrade, GetPrintGrowthGrade, GetUnusedErrorCorrectionGrade

```
//////////////////////////////////////////////////////////
// This code snippet shows how to read a data matrix code //
// and retrieve its print quality grading.                //
//////////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// Matrix code reader constructor
EMatrixCodeReader reader;

// Matrix code constructor
EMatrixCode mxCode;

// ...

// Enable grading computation
reader.SetComputeGrading(TRUE);

// Read the source image
mxCode = reader.Read(srcImage);

// Retrieve the print quality grading
int axialNonUniformityGrade= mxCode.GetAxialNonUniformityGrade();
int contrastGrade= mxCode.GetContrastGrade();
int printGrowthGrade= mxCode.GetPrintGrowthGrade();
int unusedErrorCorrectionGrade= mxCode.GetUnusedErrorCorrectionGrade();
```

17. EasyMatrixCode2

17.1. Reading Data Matrix Codes

Functional Guide | Reference: Read, SetMaxNumCodes, GetDecodedString

```
////////////////////////////////////////////////////////////
// This code snippet shows how to read data matrix codes //
// and retrieve the decoded string.                       //
////////////////////////////////////////////////////////////
namespace EMC2 = Euresys::Open_eVision_x_x::EasyMatrixCode2;
// Load an image
EImageBW8 image;
image.Load("image.bmp");
// Prepare a matrix code reader
EMC2::EMatrixCodeReader reader;
// Let the reader know that there are no more than 3 codes in the image
reader.SetMaxNumCodes(3);
// Read the source image
reader.Read(image);
// Retrieve the detected codes
std::vector<EMC2::EMatrixCode> codes = reader.GetReadResults();
// Retrieve the decoded string for the first code
std::string result = codes[0].GetDecodedString();
```

17.2. Learning a Data Matrix Code

Functional Guide | Reference: Read, Learn, GetDecodedString

```
////////////////////////////////////////////////////////////////
// This code snippet shows how to learn from a given image, //
// perform the reading and retrieve the decoded string.     //
////////////////////////////////////////////////////////////////

namespace EMC2 = Euresys::Open_eVision_x_x::EasyMatrixCode2;

// Load an image
EImageBW8 image;
image.Load("image.bmp");

// Prepare a matrix code reader
EMC2::EMatrixCodeReader reader;

// Learn from this image
reader.Learn(image);

// Read the codes in this image
reader.Read(image);


// Retrieve the detected codes
std::vector<EMC2::EMatrixCode> codes = reader.GetReadResults();
```

```
// Retrieve the decoded string of the first code
std::string result = codes[0].GetDecodedString();
```

17.3. Reading a Grid of Matrix Codes

Functional Guide | Reference: Read, EMatrixCodeGrid

```
//////////////////////////////////////////////////////////////
// This code snippet shows how to read matrix codes that are //
// disposed in a 5 by 3 grid with 10% overlap between grid   //
// cells                                                     //
//////////////////////////////////////////////////////////////

EImageBW8 srcImage;
ERectangleRegion gridRegion;
EMC2::EMatrixCodeReader reader;

// Reading succeeds
EMC2::EMatrixCodeGrid grid = reader.Read(srcImage, gridRegion, 3, 5, 0.1f);

std::vector<EMC2::EMatrixCode> codesMiddleCell = grid.GetResults(1, 2);
```

17.4. Grading a Data Matrix Code

Functional Guide | Reference: Read, SetComputeGrading

```
//////////////////////////////////////////////////////////////
// This code snippet shows how to read a data matrix code //
// and retrieve its print quality grades.                 //
//////////////////////////////////////////////////////////////
namespace EMC2 = Euresys::Open_eVision_x_x::EasyMatrixCode2;
// Load an image
EImageBW8 image;
image.Load("image.bmp");
// Prepare a matrix code reader
EMC2::EMatrixCodeReader reader;
// Tell the reader to compute grades for the read codes
reader.SetComputeGrading(true);
// Read the codes in this image
reader.Read(image);
// Retrieve the detected codes
std::vector<EMC2::EMatrixCode> codes = reader.GetReadResults();
// Retrieve the SemiT10 grades of the first code
EMatrixCodeSemiT10GradingParameters semiT10Grades = codes[0].GetSemiT10GradingParameters();
// Retrieve specific grade values
float cellDefects = semiT10Grades.CellDefects;
float symbolContrast = semiT10Grades.SymbolContrast;
float unusedErrorCorrection = semiT10Grades.UnusedErrorCorrection;
```

17.5. Asynchronous Processing

```
////////////////////////////////////////////////////////////////////////////
// This code snippet shows how to read data matrix codes asynchronously //
// from three separate images.                                          //
// The code in this snippet is valid for C++11 and newer.               //
////////////////////////////////////////////////////////////////////////////

#include <thread>
#include <atomic>
namespace EMC2 = Euresys::Open_eVision_x_x::EasyMatrixCode2;
// create a subroutine that reads the codes from an image
void Read(EImageBW8& image, EMC2::EMatrixCodeReader& reader, std::vector<EMC2::EMatrixCode>& codes,
std::atomic<bool>& finished)
{
  // read the codes in this image
  reader.Read(image);

  // extract the results
  codes = reader.GetReadResults();

  // notify that the reader has finished
  finished = true;
}
int main()
{
  // Prepare three images
  EImageBW8 img1, img2, img3;

  // Prepare three matrix code readers
  EMC2::EMatrixCodeReader reader1, reader2, reader3;

  // Prepare three vectors of matrix code instances
  std::vector<EMC2::EMatrixCode> codes1, codes2, codes3;

  // Prepare three booleans to track the thread progress
  std::atomic<bool> finished1, finished2, finished3;

  // Load the three images
  img1.Load("image1.bmp");
  img2.Load("image2.bmp");
  img3.Load("image3.bmp");

  // Set the progress trackers to false
  finished1 = false;
  finished2 = false;
  finished3 = false;

  // Launch three threads to read the codes in each image
  // the threads will run in the background.
  std::thread thr1 = std::thread([&]{ Read(img1, reader1, codes1, finished1); });
  std::thread thr2 = std::thread([&]{ Read(img2, reader2, codes2, finished2); });
  std::thread thr3 = std::thread([&]{ Read(img3, reader3, codes3, finished3); });

  // Wait until one of the threads has finished
  while (!(finished1 || finished2 || finished3))
    std::this_thread::sleep_for(std::chrono::milliseconds(5));

  // Here, we manually stop all code readers, they will stop processing
  // even if they have not yet found the codes in the image
  reader1.StopProcess();
  reader2.StopProcess();
```

```
  reader3.StopProcess();

  // wait for the threads to completely finish before continuing
  thr1.join();
  thr2.join();
  thr3.join();

  return 0;
}
```

18. EasyQRCode

18.1. Reading QR Codes

Functional Guide | Reference: Read

```
////////////////////////////////////////////////////
// This code snippet shows how to read a QR code  //
// and retrieve the decoded data.                 //
////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// QR code reader constructor
EQRCodeReader reader;

// ...

// Read
std::vector<EQRCode> qrCodes = reader.Read(srcImage);
```

18.2. Reading a Grid of QR Codes

Functional Guide | Reference: Read, EQRCodeGrid

```
////////////////////////////////////////////////////////////////
// This code snippet shows how to read QR codes that are     //
// disposed in a 5 by 3 grid with 10% overlap between grid   //
// cells                                                     //
////////////////////////////////////////////////////////////////

EImageBW8 srcImage;
ERectangleRegion gridRegion;
EQRCodeReader reader;

// Reading succeeds
EQRCodeGrid grid = reader.Read(srcImage, gridRegion, 3, 5, 0.1f);

std::vector<EQRCode> codesMiddleCell = grid.GetResults(1, 2);
```

18.3. Grading a QR Code

Functional Guide | Reference: Read, SetComputeGrading

```
////////////////////////////////////////////////////
// This code snippet shows how to read a QR code  //
// and retrieve its print quality grades          //
```

```
//////////////////////////////////////////////////

// Load an image
EImageBW8 image;
image.Load("image.bmp");

// Prepare a qr code reader
EQRCodeReader reader;

// Tell the reader to compute grades for the codes read
reader.SetComputeGrading(true);

// Read the codes in this image
std::vector<EQRCode> codes = reader.Read(image);

// Retrieve the detected codes
// Retrieve the ISO15415 grades of the first code
EQRCodeIso15415GradingParameters grades = codes[0].GetIso15415GradingParameters();

// Retrieve the scan grade of the code
float scanGrade = grades.ScanGrade;
```

18.4. Retrieving Information of a QR Code

Functional Guide | Reference: Read, GetVersion, GetModel, GetGeometry

```
//////////////////////////////////////////////////
// This code snippet shows how to read a QR code  //
// and retrieve the associated information.       //
//////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// QR code reader constructor
EQRCodeReader reader;

// ...

// Read
std::vector<EQRCode> qrCodes = reader.Read(srcImage);

// Retrieve version, model and position information
// of the first QR code found, if one was found
if (qrCodes.size() > 0)
{
    int version = qrCodes[0].GetVersion();
    EQRCodeModel model = qrCodes[0].GetModel();
    EQRCodeGeometry geometry = qrCodes[0].GetGeometry();
}
```

18.5. Tuning the Search Parameters

Functional Guide | Reference: Read, GetDecodedString, SetSearchedModels, SetMaximumVersion, SetMinimumIsotropy

```
////////////////////////////////////////////////////
// This code snippet shows how to read a QR code  //
// and retrieve the decoded data after setting a  //
// number of search parameters.                   //
////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// QR code reader constructor
EQRCodeReader reader;

// ...

// Set the search parameters
reader.SetMaximumVersion(7);
reader.SetMinimumIsotropy(0.9f);

// Set the searched models
std::vector<EQRCodeModel> models;
models.push_back(EQRCodeModel_Model2);
reader.SetSearchedModels(models);

// Read
std::vector<EQRCode> qrCodes = reader.Read(srcImage);

// Retrieve the decoded string in best guess mode of the first QR code found
string decodedString = qrCodes[0].GetDecodedString(EByteInterpretationMode_Auto);
```

18.6. Retrieving the Decoded String (Simple)

Functional Guide | Reference: Read, GetDecodedString

```
////////////////////////////////////////////////////
// This code snippet shows how to read a QR code  //
// and retrieve the decoded string.               //
////////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// QR code reader constructor
EQRCodeReader reader;

// ...

// Read
std::vector<EQRCode> qrCodes = reader.Read(srcImage);

// Retrieve the data of the first QR code found in best guess mode
string decodedString = qrCodes[0].GetDecodedString(EByteInterpretationMode_Auto);
```

18.7. Retrieving the Decoded String (Safe)

Functional Guide | Reference: Read, GetDecodedString

```
//////////////////////////////////////////////////
// This code snippet shows how to read a QR code  //
// and retrieve the decoded string                //
//////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// QR code reader constructor
EQRCodeReader reader;

// ...

// Read
std::vector<EQRCode> qrCodes = reader.Read(srcImage);

// Retrieve the data of the first QR code found
string decodedString = "";
try
{
  // The QR Code can be fully decoded without user input
  decodedString = qrCodes[0].GetDecodedString();
}
catch(EException exc)
{
  // Handle the exception
  ...
  // The QR Code cannot be fully decoded without user input
  // use hexadecimal byte interpretation
  decodedString = qrCodes[0].GetDecodedString(EByteInterpretationMode_Hexadecimal);
}
```

18.8. Retrieving the Decoded Data (Advanced)

Functional Guide | Reference: Read, GetDecodedStream, GetDecodedData, GetCodingMode, GetDecodedStreamParts

```
//////////////////////////////////////////////////
// This code snippet shows how to read a QR code  //
// and retrieve its coding mode,                  //
// the raw bit stream and the data part by part   //
//////////////////////////////////////////////////

// Image constructor
EImageBW8 srcImage;

// QR code reader constructor
EQRCodeReader reader;
// ...

// Read
std::vector<EQRCode> qrCodes = reader.Read(srcImage);

// Retrieve the data stream of the first QR code found
EQRCodeDecodedStream stream = qrCodes[0].GetDecodedStream();

// Retrieve the coding mode and the raw bit stream of the first QR code found
EQRCodeCodingMode codingMode = stream.GetCodingMode();
vector<UINT8> bitstream = stream.GetRawBitstream();
```

```
// Retrieve the encoding and the decoded data of each part of the first QR code found
vector<EQRCodeDecodedStreamPart> parts = stream.GetDecodedStreamParts();
for(unsigned int i = 0 ; i < parts.size(); ++i)
{
  // Retrieve encoding
  EQRCodeEncoding encoding = parts[i].GetEncoding();

  // Retrieve the decoded data
  vector<UINT8> decodedData = parts[i].GetDecodedData();

  // Interpret the decoded data based on the retrieved encoding
  ...
}
```

19. Easy3D

19.1. Using EFilters to Remove the Noise on a ZMap Based on the Standard Deviation

```
////////////////////////////////////////////////////////////
// The code below removes pixels with a standard deviation //
// higher than a defined threshold in a ZMap.              //
////////////////////////////////////////////////////////////

// Load the ZMap data
EZMap16 zmap;
zmap.Load("");

// Compute the filtered ZMap. The new ZMap is called filteredZmap
// The size of the kernel is 7x7, the threshold is 30.0
EZMap16 filteredZmap;
filteredZmap.SetSize(zmap);

EFilters::RemoveNoise(zmap, filteredZmap, ENoiseRemovalMethod_HighStandardDeviation, 3, 30.0, 0.0);
```

19.2. Using EFilters to Remove the Noise on a ZMap Based on the Derivation from Neighborhood

```
////////////////////////////////////////////////////////////////////
// The code below first applies a low pass filter to a ZMap.      //
// It then removes from the result the pixels showing a deviation //
// from the neighborhood larger than the defined threshold.       //
////////////////////////////////////////////////////////////////////

// Load the ZMap data
EZMap16 zmap;
zmap.Load("");

// Compute the filtered ZMap. The new ZMap is called averagedZMap
// The size of the kernel is 7x7, the threshold is 30.0
EZMap16 averagedZMap;
averagedZMap.SetSize(zmap);

EStatistics::ComputeAverageMap(zmap, averagedZMap, 3, 0.2f);


// Compute the filtered ZMap. From averagedZMap, compute filteredZMap
// The size of the kernel is 31x31, the threshold is 20.0
EZMap16 filteredZMap;
filteredZMap.SetSize(zmap);

EFilters::RemoveNoise(averagedZMap, filteredZMap, ENoiseRemovalMethod_AbsoluteDifferenceFromMean, 15, 20.0,
0.2f);
```

19.3. Reducing the Size of a Cloud with Random Decimation

```
//////////////////////////////////////////////////////////////////////
// The code reduce the size of a cloud by removing points randomly //
//////////////////////////////////////////////////////////////////////

EPointCloud pc;
pc.Load("");

// Explicitly decimate the point cloud to keep 5000 points
ERandomDecimator decimator(5000);
EPointCloud pcDecimated;

decimator.Decimate(pc, pcDecimated);
```

19.4. Reducing the Size of a Cloud with Grid Decimation

```
//////////////////////////////////////////////////////////////
// The code reduce the size of a cloud by removing points //
// to keep at most one per cell of a regular grid         //
//////////////////////////////////////////////////////////////

EPointCloud pc;
pc.Load("");

// Explicitly decimate the point cloud to keep
// one point in every cube of 10*10*10
EGridDecimator decimator(10.f);
EPointCloud pcDecimated;

decimator.Decimate(pc, pcDecimated);
```

19.5. Using Photometric Stereo

```
//////////////////////////////////////////////////
// The code shows how to use Photometric Stereo //
// from calibration to retrieve the results     //
//////////////////////////////////////////////////

EPhotometricStereoImager photometricStereo;

std::vector<EImageBW8> calibrationImages;
// Load calibration images (Todo)
std::vector<EROIBW8> calibrationROIs;
// Set the calibration ROIs (Todo)

// Calibrate
float score = photometricStereo.CalibrateFromSphere(calibrationROIs);
```

```
std::vector<EImageBW8> objectImages;
// Load object images in the same order than the calibration images/angles (Todo)

std::vector<EROIBW8> objectROIs;
// Set the object ROIs (Todo)


// Compute
photometricStereo.Compute(objectROIs);

// Retrieve the results
EImageC24 normals = photometricStereo.GetNormals();
EImageBW8 albedos = photometricStereo.GetAlbedos(Easy3D::EPhotometricStereoContrast_HighContrast);
EImageBW8 gradientsX = photometricStereo.GetGradientsX();
EImageBW8 gradientsY = photometricStereo.GetGradientsY();
EImageBW8 gaussianCurvatures = photometricStereo.ComputeGaussianCurvatures(Easy3D::EPhotometricStereoContrast_
HighContrast);
EImageBW8 meanCurvatures = photometricStereo.ComputeMeanCurvatures(Easy3D::EPhotometricStereoContrast_
HighContrast);
EZMap8 heightMap = photometricStereo.ComputeHeightMap();
```

19.6. Using Flat Images to Improve Photometric Stereo

```
////////////////////////////////////////////
// The code shows how to use flat images to //
// improve photometric stereo's results     //
////////////////////////////////////////////

EPhotometricStereoImager photometricStereo;
// calibrate imager or sets its angles (Todo)

// Load flat images in the same order than the calibration images/angles (Todo)

std::vector<EROIBW8> flatROIs;
// Set the flat images ROIs (Todo)


// Configure flat images, this could optionally be done with a dark image as well
photometricStereo.ConfigureNonUniformLightingCorrection(flatROIs);

std::vector<EROIBW8> objectROIs;
// Set the object ROIs (Todo)


// Perform one or more computations, each will use the flat images (Todo)
photometricStereo.Compute(objectROIs);

// Optional: non uniform lighting correction could be disabled or (re-)enabled
// using SetEnableNonUniformLightingCorrection
```

19.7. Performing Plane Leveling on Point Clouds

```
////////////////////////////////////////////////
// The code shows how to perform plane leveling //
// on point clouds                              //
////////////////////////////////////////////////

// find the reference plane on the point cloud
E3DPlane ref_plane;
EPointCloud point_cloud;

// define the ground plane as the plane Z=0
E3DPlane ground_plane(E3DPlane::ZPlane());

// get the transformation that moves
// the reference plane to the ground plane
E3DTransformMatrix transformation;
transformation = ref_plane.GetTransformationTo(ground_plane);

// apply the transformation to the point cloud
EAffineTransformer transformer;
transformer.ApplyMatrix(transformation, point_cloud);
```

19.8. Using an ERegion to Crop a ZMap

```
////////////////////////////////////////////////////
// The code shows how to perform cropping on a zmap //
////////////////////////////////////////////////////

EZMap8 zmap;
zmap.Load("");

// prepare an ERegion
std::vector<EPoint> points = { EPoint(90, 76), EPoint(432, 87),
EPoint(466, 91), EPoint(502, 122), EPoint(513, 169),
EPoint(485, 218), EPoint(436, 231), EPoint(86, 215) };
EPolygonRegion region(points);  // could be any type of ERegion

EZMap8 zmapCropped(zmap.GetWidth(), zmap.GetHeight());
EUtils::Copy(zmap.GetUndefinedValue(), zmapCropped);
EUtils::Copy(zmap, region, zmapCropped);
```

19.9. Add an Attribute to an EPointCloud with Initial Data

```
////////////////////////////////////////////////////
// The code shows how to add an attribute           //
// to a cloud when you already have the data        //
////////////////////////////////////////////////////

EPointCloud cloud;
std::vector<EC24A> data;
```

```
// case 1: data is Color, Normal, Intensity, Texture, Index, Confidence or Distance
cloud.FillAttributeBuffer(E3DAttribute_Color, data);

// case 2: data is something else
int attributeOffset = cloud.AddCustomAttributeBuffer(data);
```

19.10. Add an Attribute to an EPointCloud without Initial Data

```
//////////////////////////////////////////////////////
// The code shows how to add an attribute           //
// to a cloud when you don't have the data          //
//////////////////////////////////////////////////////

EPointCloud cloud;
EC24A defaultValue;

// case 1: data is Color, Normal, Intensity, Texture, Index, Confidence or Distance
cloud.AllocateAttributeBuffer(E3DAttribute_Color, defaultValue);

// case 2: data is something else
int attributeOffset = cloud.AllocateCustomAttributeBuffer(defaultValue);
```

19.11. Retrieve an Attribute from an EPointCloud

```
//////////////////////////////////////////////////////////////
// The code shows how to retrieve an attribute from a cloud //
//////////////////////////////////////////////////////////////

EPointCloud cloud;
const EC24A* data;

// case 1: data is Color, Normal, Intensity, Texture, Index, Confidence or Distance
data = static_cast<const EC24A*>(cloud.GetAttributeBuffer(E3DAttribute_Color));

// case 2: data is something else
int attributeOffset = 17;  // value retrieved when we initialized the attribute
data = static_cast<const EC24A*>(cloud.GetAttributeBuffer(attributeOffset));
```

20. Easy3DObject

20.1. Extracting 3D Objects with a Selection Criterion

```
// EZmap constructor
EZMap8 zMap;

// Extractor constructor
E3DObjectExtractor extractor;

// Setting a selection criterion
extractor.SetWidthRange(EFloatRange(10, 500));


// Extracts the objects from the EZMap
int regionNB = extractor.Extract(zMap);

// Retrieve the extracted objects
std::vector<E3DObject> objects = extractor.GetObjects();
```

20.2. Inspecting a Feature from the List of E3DObjects

```
                // Get the list of E3DObjects
std::vector<E3DObject> objects = extractor.GetObjects();

// Get the volume of the first object
float volume = objects[0].GetVolume();

// Get the ERectangleRegion of the last (the largest) object
ERectangleRegion region = objects.back().GetRectangleRegion();
```

20.3. Drawing a 2D Feature from the List of E3DObjects

```
                // Get the list of E3DObjects
std::vector<E3DObject> objects = extractor.GetObjects();

// Get a render context
HDC drawHDC;

// Draw the ERegion of each object
```

```
int nObjects = (int)objects.size();
for (int i = 0; i < nObjects; i++)
  objects[i].Draw(drawHDC, E3DObjectFeature_ERegion, ERGBColor(0, 255, 0));
```

20.4. Drawing 3D Features from a List of E3DObjects

```
                // Get the list of E3DObjects
std::vector<E3DObject> objects = extractor.GetObjects();

// Register the list of E3DObjects to the 3D viewer
E3DViewer viewer3D(0,0,640,480);
viewer3D.Register3DObjects(objects);


// Define and use a render style for the ReferenceTopPosition feature
ERenderStyle renderStyle;
renderStyle.pointRGB = EC24A(100, 0, 0);
viewer3D.SetFeatureStyleForAll3DObjects(renderStyle, E3DObjectFeature_ReferenceTopPosition);

// Set a different rendering color for the first object
ERenderStyle selectedRenderStyle;
selectedRenderStyle.pointRGB = EC24A(255, 255, 0);
viewer3D.SetFeatureStyleFor3DObject(0, selectedRenderStyle, E3DObjectFeature_ReferenceTopPosition);

// Enable the display of the TopZPosition feature
viewer3D.ShowFeatureForAll3DObjects(E3DObjectFeature_ReferenceTopPosition);
```

21. Easy3DMatch

21.1. E3DAligner Minimal Code

```
////////////////////////////////////////////////////
// This code snippet shows how to compute the     //
// alignment between a sample and a cad reference.//
////////////////////////////////////////////////////

// load the reference mesh and define the pose
Easy3D::E3DAligner aligner;
Easy3D::EMesh cad;
cad.Load("…");
float azimuthReference = 0.f, elevationReference = 90.f;
aligner.SetReference(cad, azimuthReference, elevationReference);

// load the sample
Easy3D::EPointCloud sample;
sample.Load("…");
float azimuthSample = 0.f, elevationSample = 90.f;

// perform alignment
Easy3D::E3DAlignment alignment = aligner.Align(sample, azimuthSample, elevationSample);
```

21.2. E3DAligner Reprojection Plane

```
////////////////////////////////////////////////////
// This code snippet shows how to set the         //
// reprojection plane when performing alignment.  //
////////////////////////////////////////////////////

// load the reference mesh and define the pose
Easy3D::E3DAligner aligner;
Easy3D::EMesh cad;
cad.Load("…");
Easy3D::E3DPlane refPlane(Easy3D::E3DPoint(0, 0, 1), 0);
aligner.SetReference(cad, refPlane);

// define the reprojection plane
bool userKnowsPlaneAZEl = false; // depending on the user
if (userKnowsPlaneEquation)
{
  Easy3D::E3DPlane reprojectionPlane(Easy3D::E3DPoint(0, 0, -1), -15);
  aligner.SetScanReprojectionPlane(reprojectionPlane);
}
else
{
  Easy3D::EPointCloud cloud;
  cloud.Load("…");
  bool objectAbovePlane = true; // is the object above the plane on the cloud
  aligner.SetFlatScan(cloud, objectAbovePlane);
```

```
}

// load the sample
Easy3D::EPointCloud sample;
sample.Load("…");
float azimuthSample = 0.f, elevationSample = 90.f;

// perform alignment
Easy3D::E3DAlignment alignment = aligner.Align(sample, azimuthSample, elevationSample);
```

21.3. E3DAlignment Align Sample

```
////////////////////////////////////////////////////
// This code snippet shows how to apply the       //
// transformation of the E3DAlignment to the      //
// sample to overlap it on the reference          //
////////////////////////////////////////////////////

// perform alignment (see previous examples)
Easy3D::E3DAlignment alignment;
EPointCloud sample;

// align sample on reference
Easy3D::EPointCloud alignedSample;
Easy3D::EAffineTransformer::ApplyMatrix(alignment.GetPose(), sample, alignedSample);
```

21.4. E3DComparer Minimal Sample

```
////////////////////////////////////////////////////
// This code snippet shows how to compare a sample //
// with a golden scan reference.                   //
////////////////////////////////////////////////////

// load the reference golden scan and set reference
Easy3D::E3DComparer comparer;
Easy3D::EPointCloud cloud;
cloud.Load("…");
comparer.SetPointCloudReference(cloud);

// set thresholds
float distanceThresh = .2f, areaThresh = 1.f;
comparer.SetAnomalyThresholds(distanceThresh, areaThresh);

// prepare data structures (optional)
comparer.PrepareReference();

// load the sample and perform comparison
Easy3D::EPointCloud sample;
sample.Load("…");
comparer.Compare(sample);

// compute anomalies
std::vector<Easy3D::E3DAnomaly> anomalies = comparer.ComputesAnomalies();


// TODO: if (anomalies.size() != 0u): an anomaly was detected: inspect the sample manually? throw it away?
```

```
// get cloud to inspect it manually
Easy3D::EPointCloud visualisationCloud;
comparer.GetComparisonPointCloud(visualisationCloud);
```

21.5. E3DComparer Advanced Sample

```
////////////////////////////////////////////////////
// This code snippet shows how to set the options //
// when comparing two elements with E3DComparer.  //
////////////////////////////////////////////////////

// load the reference golden scan and set reference
Easy3D::E3DComparer comparer;
Easy3D::EPointCloud cloud;
cloud.Load("…");
comparer.SetPointCloudReference(cloud);

// set thresholds
float distanceThresh = .2f, areaThresh = 1.f;
float hystDistanceThresh = 1.5f, hystAreaThresh = .5f;
comparer.SetAnomalyThresholds(distanceThresh, areaThresh);
comparer.SetAnomalyHysteresis(hystDistanceThresh, hystAreaThresh);  // defined relatively to base thresholds

// set ROIs
std::vector<Easy3D::E3DBox> rois = { Easy3D::E3DBox(15, 15, 15) };
comparer.SetROI(rois);
std::vector<Easy3D::E3DBox> dontCare = { Easy3D::E3DBox(5, 5, 5) };
comparer.SetDontCare(dontCare);
std::vector<Easy3D::E3DBox> noExtraMaterial = { Easy3D::E3DBox(Easy3D::E3DPoint(10, 15, 20), 0, 0, 0, 5, 5, 5)
};
comparer.SetNoExtraMaterial(noExtraMaterial);


// prepare data structures (optional)
comparer.PrepareReference();

// load the sample and perform comparison
Easy3D::EPointCloud sample;
sample.Load("…");
comparer.Compare(sample);

// compute anomalies
std::vector<Easy3D::E3DAnomaly> anomalies = comparer.ComputesAnomalies();

// TODO: if (anomalies.size() != 0u): an anomaly was detected: inspect the sample manually? throw it away?

// get cloud to inspect it manually
Easy3D::EPointCloud visualisationCloud;
comparer.GetComparisonPointCloud(visualisationCloud);
```

21.6. E3DMatcher Minimal Sample

```
////////////////////////////////////////////////////
// This code snippet shows how to match a sample  //
// with a golden scan reference.                  //
```

```
////////////////////////////////////////////////////

// load the reference golden scan and set reference
Easy3D::E3DMatcher matcher;
Easy3D::EPointCloud reference;
float azimuthReference = 0.f, elevationReference = 90.f;
reference.Load("…");
matcher.SetReference(reference, azimuthReference, elevationReference);

// set thresholds
float distanceThresh = .2f, areaThresh = 1.f;
matcher.SetAnomalyThresholds(distanceThresh, areaThresh);

// prepare data structures (optional)
matcher.PrepareReference();

// load the sample and perform comparison
Easy3D::EPointCloud sample;
float azimuthSample = 0.f, elevationSample = -90.f;
sample.Load("…");
Easy3D::E3DMatch match = matcher.Match(sample, azimuthSample, elevationSample);
std::vector<Easy3D::E3DAnomaly> anomalies = match.GetAnomalies();

// TODO: if (anomalies.size() != 0u): an anomaly was detected: inspect the sample manually? throw it away?

// get cloud to inspect it manually
Easy3D::EPointCloud visualisationCloud;
matcher.GetComparisonPointCloud(visualisationCloud);
```

21.7. E3DMatcher Advanced Sample

```
////////////////////////////////////////////////////
// This code snippet shows how to set the options //
// when matching two elements with E3DMatcher.    //
////////////////////////////////////////////////////

// load the reference golden scan and set reference
Easy3D::E3DMatcher matcher;
Easy3D::EPointCloud reference;
float azimuthReference = 0.f, elevationReference = 90.f;
reference.Load("…");
matcher.SetReference(reference, azimuthReference, elevationReference);

// use advanced comparison mode
matcher.SetComparisonDistanceMode(EComparisonDistanceMode_Advanced);

// ignore shadows
matcher.SetEnableMissingPointAsAnomaly(false);

// set thresholds
float distanceThresh = .2f, areaThresh = 1.f;
float hystDistanceThresh = 1.5f, hystAreaThresh = .5f;
matcher.SetAnomalyThresholds(distanceThresh, areaThresh);
matcher.SetAnomalyHysteresis(hystDistanceThresh, hystAreaThresh);  // defined relatively to base thresholds

// retrieve reference poses (reference must have been set)
std::vector<Easy3D::EZMap8> referencePoseProjections;
matcher.RetrieveReferencePosesProjections(referencePoseProjections);

// set ROI on the left half of the object
```

```
ERectangleRegion roiRegion(0.f, 0.f, float(referencePoseProjections[0].GetWidth()) / 2.f, float
(referencePoseProjections[0].GetHeight()));
matcher.SetComparisonROI(&roiRegion);

// set No Extra material on the whole object
ERectangleRegion noExtraMatRegion(0.f, 0.f, float(referencePoseProjections[0].GetWidth()) / 2.f, float
(referencePoseProjections[0].GetHeight()));
matcher.SetComparisonNoExtraMaterial(&noExtraMatRegion);


// prepare data structures (optional)
matcher.PrepareReference();

// load the sample and perform comparison
Easy3D::EPointCloud sample;
float azimuthSample = 0.f, elevationSample = -90.f;
sample.Load("…");
Easy3D::E3DMatch match = matcher.Match(sample, azimuthSample, elevationSample);
std::vector<Easy3D::E3DAnomaly> anomalies = match.GetAnomalies();

// TODO: if (anomalies.size() != 0u):an anomaly was detected: inspect the sample manually? throw it away?

// get cloud to inspect it manually
Easy3D::EPointCloud visualisationCloud;
matcher.GetComparisonPointCloud(visualisationCloud);
```

21.8. EPointCloudMerger Sample

```
////////////////////////////////////////////////////////////
// This code snippet shows how to perform sensor fusion. //
////////////////////////////////////////////////////////////

// Calibration
Easy3D::EPointCloudMerger merger;
std::vector<EPointCloud> calibrationClouds;  // TODO: load or grab
float calibrationObjectSize = 100.f;  // size of an edge of the cube in the calibrationClouds
float calibrationScore = merger.Calibrate(calibrationClouds, calibrationObjectSize, true);

// Merging
std::vector<EPointCloud> clouds;  // TODO: load or grab, must be in same order as CalibrationClouds
EPointCloud mergedCloud;

merger.Merge(clouds, mergedCloud);
```