USER GUIDE

Open eVision

Text and Code Reading Tools
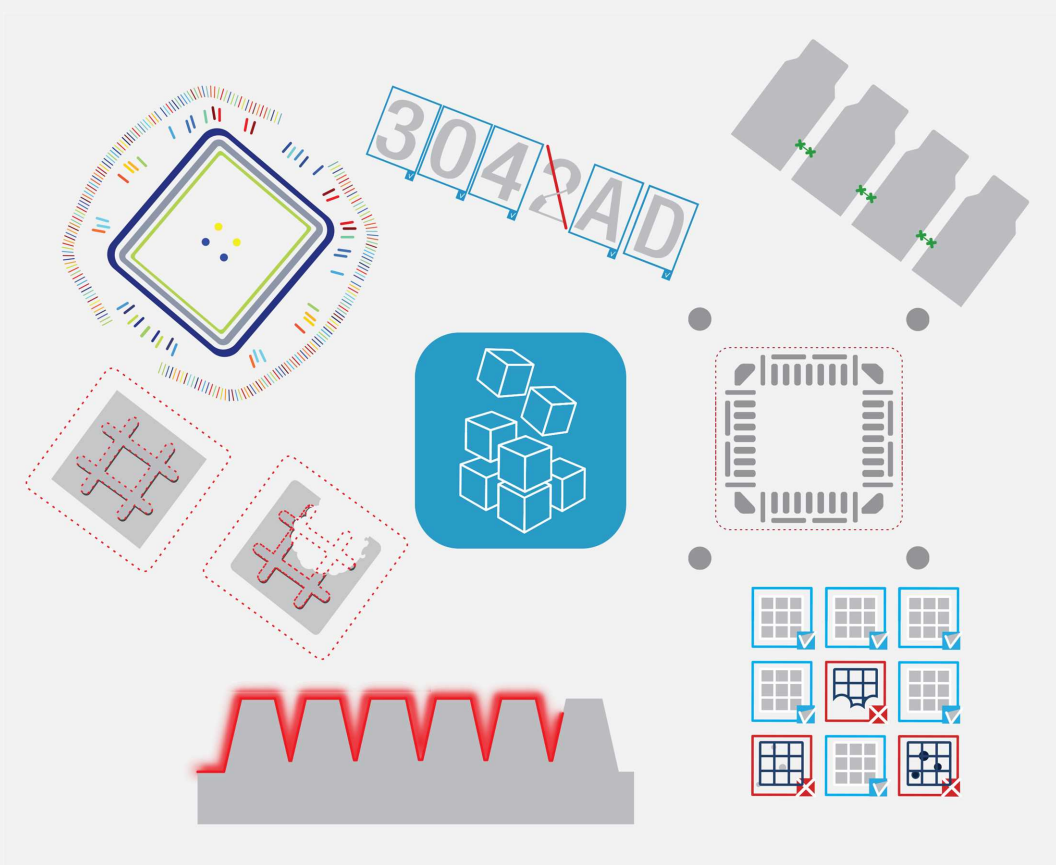
Contents

1. Dealing with Pixel Containers and Files

1.1. Pixel Container Definition

Image objects

The **Open eVision** image objects contain image data that represents rectangular images.

Each image object has a data buffer, accessible via a pointer, where pixel values are stored contiguously, row by row.
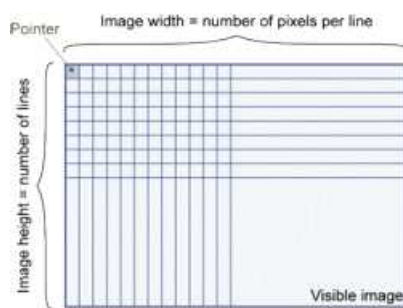
Image main parameters

The rectangular array of pixels of an **Open eVision** image object is characterized by the EBaseROI parameters:

- The Width is the number of pixels per row of the image.

- The Height is the number of rows of the image.

- The Size contains both the Width and the Height of the image.

The maximum size for the width and the height is:

  - ☐ 32,767 (2^{15}-1) in **Open eVision** 32-bit

  - ☐ 2,147,483,647 (2^{31}-1) in **Open eVision** 64-bit

- The Plane contains the number of color components.
  - ☐ For gray-level images: Plane = 1
  - ☐ For color images: Plane = 3

Classes

The image and ROI classes derive from the abstract class EBaseROI and inherit all its properties.
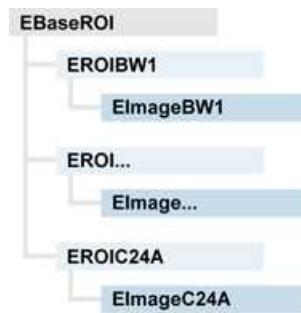
Depth maps

A *depth map* represents a 3D object using a 2D grayscale image in which each pixel represents a 3D point.

- The pixel coordinates are the X and Y coordinates of the point.
- The gray value of the pixel is a representation of the Z coordinate of the point.

Point clouds

A *point cloud* is an unstructured set of 3D points representing discrete positions on the surface of an object.

The point clouds are produced by various 3D scanning techniques, such as laser triangulation, time of flight or structured lighting.

📄 For details, see, for example, en.wikipedia.org/wiki/Point_cloud.
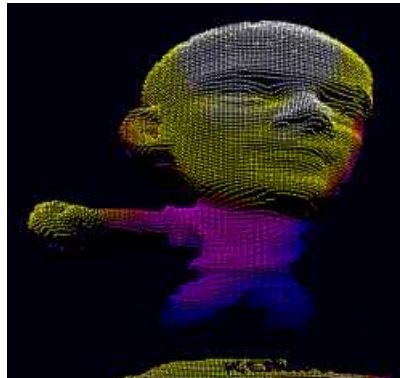
1.2. Pixel Container Types

📄 For the enumeration of the available types, see "EImageType Enum" on page 1.

Images

Open eVision supports the following image types according to their pixel types.

Open eVision	Genicam PNFC	Definition	Class
BW1	Mono1	1-bit black and white image (8 pixels are stored in 1 byte).	EImageBW1
BW8	Mono8	8-bit grayscale image (each pixel is stored in 1 byte).	EImageBW8
BW16	Mono16	16-bit grayscale image (each pixel is stored in 2 bytes).	EImageBW16
BW32	Mono32	32-bit grayscale image (each pixel is stored in 4 bytes).	EImageBW32
C15	RGB5	15-bit color image (each pixel is stored in 2 bytes). Compatible with Microsoft® Windows RGB15 color images.	EImageC15
C16	RGB565	16-bit color image (each pixel is stored in 2 bytes). Compatible with Microsoft® Windows RGB16 color images.	EImageC16
C24	RGB8	24-bit color image (each pixel is stored in 3 bytes). Compatible with Microsoft® Windows RGB24 color images.	EImageC24
C24A	BGRa8	32-bit color image (each pixel is stored in 4 bytes). Compatible with Microsoft® Windows RGB32 color images.	EImageC24A

✅ **TIP**
Easy.GetBestMatchingImageType returns the best matching image type for a given file on disk.

Depth Maps

Open eVision	Genicam PNFC	Definition	Class
EDepth8	Coord3D_C8	8-bit depth map (each pixel is stored in 1 byte as an integer)	EDepthMap8
EDepth16	Coord3D_C16	16-bit depth map (each pixel is stored in 2 bytes as a fixed point)	EDepthMap16
EDepth32f	Coord3D_C32	32-bit depth map (each pixel is stored in 4 bytes as a float)	EDepthMap32f

> **TIP**
>
> 8 and 16-bit depth map values are stored in buffers compatible with the 2D
> **Open eVision** images.

Point Clouds

Open eVision	Genicam PNFC	Definition	Class
Point Cloud	Coord3D_ABC32	Set of points coordinates (each coordinate is stored in 4 bytes as a float)	EPointCloud

1.3. Supported Image File Types

For the enumeration of the available types, see "EImageFileType Enum" on page 1.

Type	Description
BMP	Uncompressed image data format (Windows Bitmap Format).
JPEG	A lossy data compression standard issued by the **Joint Photographic Expert Group** registered as ISO/IEC 10918-1. The compression irretrievably loses quality.
JFIF	JPEG File Interchange Format.
JPEG-2000	A data compression standard issued by the **Joint Photographic Expert Group** registered as ISO/IEC 15444-1 and ISO/IEC 15444-2. **Open eVision** supports only lossy compression format, file format and code stream variants. - The *code stream* describes the image samples. - The *file format* includes meta-information such as the image resolution and color space.
PNG	Lossless data compression method (Portable Network Graphics).
Serialized	The **Euresys** proprietary image file format obtained from the **serialization** of **Open eVision** image objects.
TIFF	The **Tag Image File Format** is currently controlled by **Adobe Systems** and uses the LibTIFF third-party library to process images written for the 5.0 or 6.0 TIFF specification. - The file *save* operations are lossless and save the images without any compression. - The file *load* operations support all the TIFF variants listed in the LibTIFF specification.

1.4. Pixel and File Types Compatibility

For the compatible combinations in the following table, the image integrity is preserved with no data loss (except from JPEG and JPEG2000 with lossy compression).

The other combinations are not supported and an exception occurs if you use them.

Type	BMP	JPEG	JPEG2000	PNG	TIFF	Serialized
BW1	✓	–	–	✓	✓	✓
BW8	✓	✓	✓	✓	✓	✓
BW16	–	–	✓	✓	✓ [2]	✓
BW32	–	–	–	–	✓ [2]	✓
C15	✓	✓ [1]	✓ [1]	✓ [1]	✓ [1]	✓
C16	✓	✓ [1]	✓ [1]	✓ [1]	✓ [1]	✓
C24	✓	✓	✓	✓	✓ [1]	✓
C24A	✓	–	–	✓	–	✓
Depth8	✓	✓	✓	✓	✓	✓
Depth16	–	–	✓	✓	✓ [2]	✓
Depth32f	–	–	–	–	–	✓

- ✓ [1] : C15 and C16 formats are automatically converted into C24 during the save operation.

- ✓ [2] : BW16 and BW32 are not supported by Baseline TIFF readers.

1.5. Color Types

Open eVision supports the following color systems:

EISH	Intensity, Saturation, Hue
ELAB	CIE Lightness, a*, b*
ELCH	Lightness, Chroma, Hue
ELSH	Lightness, Saturation, Hue
ELUV	CIE Lightness, u*, v*
ERGB	NTSC/PAL/SMPTE Red, Green, Blue
EVSH	Value, Saturation, Hue
EXYZ	CIE XYZ
EYIQ	CCIR Luma, Inphase, Quadrature
EYSH	CCIR Luma, Saturation, Hue
EYUV	CCIR Luma, U Chroma, V Chroma

2. Conventions

2.1. Conventions for Strings

Since **Open eVision** 23.08, the only character encoding used in the **Open eVision** libraries and tools is UTF-8.

☐ All methods taking `std.string` as argument expect an UTF-8 encoded `std.string`.

☐ All methods returning a `std.string` always return it as UTF-8 encoded.

[Backward compatibility on Windows](#)

On **Windows** (but not on **Linux**), there is also a sanitization process to preserve backward compatibility with older releases that didn't use the UTF-8 encoding.

● The content of each input string is checked to ensure it is UTF-8 encoded.

   If it is not the case:

☐ The string is assumed to be encoded using the current Windows Language for Non-Unicode Programs parameter.

☐ It is converted to UTF-8.

● The output strings of all libraries and tools are always UTF-8.

> ✓ **TIP**
> Despite the presence of this backward compatibility layer it is recommended to use exclusively UTF-8 to interact with **Open eVision** on all platforms to ensure the best performance and compatibility.

2.2. Image Coordinate Systems

The conventions below apply to all **Open eVision** functions and results.

☐ Pixel coordinates are usually given as integer numbers.

☐ Some results can use subpixel precision with real (floating point) numbers.

☐ Some exceptions apply and are documented per library.

Integer coordinates

- The origin (0,0) of the coordinate system is the upper left pixel of the image.
- The lower right pixel is (width-1, height-1).

Buffer/Pixel Coordinates (integer)
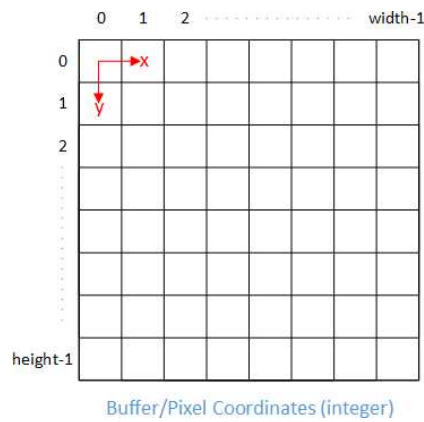
Real coordinates

- With floating point (x,y) coordinates, the origin is the upper left corner of the upper left pixel.
- The first pixel area ranges in [0,1[for X and Y axis.
- Coordinates greater or equal than the width or the height are outside the image.

Image Coordinates (float)
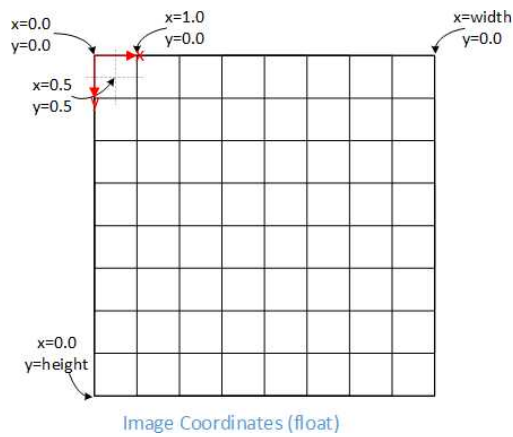
2.3. Image and Depth Map Buffer

The pixels of an image and of an depth map are stored contiguously into a buffer, from left to right and from top to bottom, in the Windows bitmap format (top-down DIB -device-independent bitmap-).

The buffer address is a pointer to the address that contains the top left pixel of the image.

- Image buffer pitch
  - □ The alignment must be a multiple of 4 bytes.
  - □ The default pitch in **Open eVision** is 32 bytes for performance reasons.
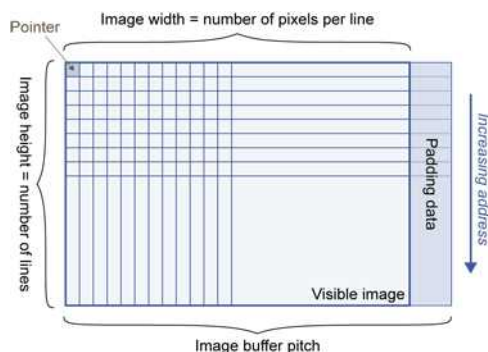
Memory layout

Image format	Layout	Illustration
EImageBW1	Stores 8 pixels in 1 byte	
EImageBW8 EDepthMap8	Store 1 pixel in 1 byte	
EImageBW16	Stores 1 pixel in 2 bytes	
EImageC15	Stores 1 pixel in 2 bytes - Each color component is coded with 5 bits - The 16th bit is unused	

Image format	Layout	Illustration
EImageC16	Stores 1 pixel in 2 bytes - The colors 1 and 3 are coded with 5 bits - The color 2 is coded with 6 bits	
EDepthMap16	Stores 1 pixel in 2 bytes (fixed point format)	
EImageC24	Stores 1 pixel in 3 bytes - Each color component is coded with 8 bits	
EImageC24A	Stores 1 pixel in 4 bytes. - Each color component is coded with 8 bits - The alpha channel is coded with 8 bits	
EDepthMap32f	Stores 1 pixel in 4 bytes (float format)	

3. Basic Operations

3.1. Memory Allocation

You can construct an image using an internal or an external memory allocation.

Internal memory allocation

The image object dynamically allocates and deallocates a buffer:
- ☐ The memory management is transparent.
- ☐ When the image size changes, a reallocation occurs.
- ☐ When an image object is destroyed, the buffer is deallocated.

To declare an image with an internal memory allocation:

1. Construct an image object, for instance `EImageBW8`, either with width and height arguments or using the `SetSize` function.

2. Access a given pixel using one of the multiple available functions.
   For example, use `GetImagePtr` to retrieve a pointer to the first byte of the pixel at the given coordinates.

External memory allocation

Control the buffer allocation or link a third-party image in the memory buffer to an **Open eVision** image.
- ☐ You must specify the image size and the buffer address.
- ☐ When an image object is destroyed, the buffer is unaffected.

📄 For details, see "Image and Depth Map Buffer" on page 13 and "Interfacing Third-Party Images" on page 174.

To declare an image with an external memory allocation:

1. Declare an image object, for instance `EImageBW8`.

2. Create a suitably sized and aligned buffer.

3. Assign the buffer to the image with `SetImagePtr`.

> **NOTE**
> Using the copy constructor of the `EImage` object to copy the externally allocated image does not copy the buffer.
> The copied image points to the same external buffer as the original image.

> **NOTE**
> If your buffer rows are not aligned on 4 bytes, use `InitializeFromUnalignedBuffer` instead of `SetImagePtr`.
> Please note that this allocates the memory internally and copies the external buffer into the internal one instead of using the external one.

3.2. Loading a Pixel Container File

Loading images and depth maps

- Use the method `Load` to load image data into an image object.
  - It has only the argument path that includes the path, filename and file name extension.
  - The file type is determined by the file format.
  - The destination image is automatically resized according to the size of the image on disk.

- `Load` throws an exception when:
  - The file type identification fails.
  - The file type is incompatible with the pixel type of the image object.

NOTE: When loading a BW16 image (with integer values) in a depth map, the fixed point precision set in the depth map (0 by default) is left unchanged and used.

Loading point clouds

Use the following methods to load a point cloud saved in a specific format:
  - `EPointCloud.Load`: **Open eVision** proprietary file format.
  - `EPointCloud.LoadCSV`: CSV file.
  - `EPointCloud.LoadOBJ`: OBJ file.
  - `EPointCloud.LoadPCD`: PCD file (supported in ASCII and binary modes).
  - `EPointCloud.LoadPLY`: PLY file (supported only in ASCII mode).
  - `EPointCloud.LoadXYZ`: XYZ file.

3.3. Saving a Pixel Container File

Images and depth maps

- Use the method Save of an image or the method SaveImage of a depth map or a ZMap to save image data of the object into a file.
  - □ The argument Path includes the path, file name and file name extension.
  - □ The argument Image File Type can be omitted. In this case, the file name extension is used.

- Save throws an exception when:
  - □ The requested image file format is incompatible with the pixel type of the image object.
  - □ The file name extension is not supported while using the Auto file type selection method.

NOTE: When saving a 16-bit depth map, the fixed point precision is lost and the pixels are considered as 16-bit integers.

> ✓ **TIP**
> The images with a width or a height larger than 65,536 must be saved in **Open eVision** proprietary format.

Image File Type arguments

Argument	Image file type
EImageFileType_Auto	(Default) Automatically determined by the file name extension. See below.
EImageFileType_ Euresys	**Open eVision** Serialization
EImageFileType_Bmp	Windows bitmap - BMP
EImageFileType_Jpeg	JPEG File Interchange Format - JFIF
EImageFileType_ Jpeg2000	JPEG 2000 File format / Code Stream - JPEG2000
EImageFileType_Png	Portable Network Graphics - PNG
EImageFileType_Tiff	Tagged Image File Format - TIFF

If the argument is EImageFileType_Auto or is missing, the assigned image file type is:

File name extension (case-insensitive)	Assigned image file type
BMP	Windows Bitmap format
JPEG or JPG	JPEG File Interchange Format - JFIF
JP2	JPEG 2000 file format
J2K or J2C	JPEG 2000 Code Stream
PNG	Portable Network Graphics
TIFF or TIF	Tagged Image File Format

Saving JPEG and JPEG2000 lossy compressions

SaveJpeg and SaveJpeg2K specify the compression quality when saving compressed images. They have two arguments:
- ☐ Path: a string of characters including the path, filename, and file name extension.
- ☐ Compression quality of the image file, an integer value in range [0: 100].
  SaveJpeg saves image data using JPEG File Interchange Format – JFIF.
  SaveJpeg2K saves image data using JPEG 2000 File format.

JPEG compression values

JPEG compression	Description
JPEG_DEFAULT_QUALITY (-1)	Default quality (*)
100	Superb image quality, lowest compression factor
75	Good image quality (*)
50	Normal image quality
25	Average image quality
10	Bad Image quality

(*) The default quality corresponds to the good image quality (75).

Representative JPEG 2000 compression quality values

JPEG 2000 compression	Description
-1	Default quality (*)
1	Highest image quality, lowest compression factor
16	Good Image Quality (*) (16:1 rate)
512	Lowest image quality, highest compression factor

(*) The default quality corresponds to the good image quality (16:1 rate).

Saving point clouds

Use the following methods to save a point cloud in a specific format:
- ☐ EPointCloud::Save: **Open eVision** proprietary file format.
- ☐ EPointCloud::SaveCSV: CSV file.
- ☐ EPointCloud::SaveOBJ: OBJ file.
- ☐ EPointCloud::SavePCD: PCD file.
- ☐ EPointCloud::SavePLY: PLY file.
- ☐ EPointCloud::SaveXYZ: XYZ file.

> ✓ TIP
> The PCD format is supported in ASCII and binary modes.

3.4. Drawing in Open eVision

Introduction

- Whenever relevant, the **Open eVision** tools provide methods `Draw` to render their contents and/or configuration. This is, for instance, the contents of an `EImage` or the frame of an `EROI`.

- A given tool can have multiple methods `Draw`, usually one for each feature available.

- The **Open eVision** methods `Draw` take an object `DrawAdapter` as their main parameter, and additional parameters for zoom and pan:

  `Tool::Draw(EDrawAdapter* adapter, float zoomX, float zoomY, float panX, float panY);`

  □ `zoomX` and `zommY` are expressed in percentage, 1 is the default value and means no zoom.

  □ It can be different in the horizontal and vertical directions (which can be useful in the case of non-square pixels for instance).

  □ If you don't provide a vertical zoom, or set it to 0, it will be set identical to the horizontal one.

  □ `panX` and `panY` are expressed in pixels, but in image coordinates. It means that the value you pass to `panX` and `panY` are multiplied by the corresponding zoom before being applied.

Example: How to draw an image and a ROI frame on a window under Windows:

```
EImageBW8 image;
EROIBW8 roi;
EWindowsDrawAdapter adapter(windowHdc);
image.Draw(adapter);
roi.DrawFrame(adapter);
```

Graphical interactions

- You can configure some of the **Open eVision** tools graphically and use the provided methods to put your configuration in place.

- Graphical Interaction-enabled tools provide special parameters to some of their methods `Draw` to draw handles on the tool representation.

- To capture the user interactions with those handles, these tools also provide two specialized methods:

  □ `HitTest` detects if a handle is under the mouse when providing it with the current cursor coordinates. You typically use this test during a mouse button down event.

  □ `Drag` moves the detected handle to the given coordinates. This in turn modifies the tool configuration to match the new handle position. `Drag` is typically associated with the mouse button up event.

  NOTE: `HitTest` and `Drag` use the same zoom and pan parameters as `Draw`. You must set them the same way (with the same values) to achieve the desired result.

Draw adapters

- The draw adapters are objects that, in addition to representing the context in which to draw, provide methods to draw the selected primitives in that context.

- They are initialized by providing the targeted context to the constructor.

- Some of the drawing methods provided by the draw adapters are (but are not limited to):
  - `EDrawAdapter::Line` / Lines draws one or more lines on the context
  - `EDrawAdapter::Rectangle` / `FilledRectangle` draws a rectangle, filled or not, on the context
  - `EDrawAdapter::Ellipse` / `FilledEllipse` draws an ellipse, filled or not, in the context
  - `EDrawAdapter::Text` / `BackedText` renders a text in the context, with or without background
  - `EDrawAdapter::Image` renders an image in the context

- For more information about the drawing primitives provided by the draw adapters, please refer to the reference documentation.

- To set the color of the primitives, provide a pen and/or a brush and use the methods `EDrawAdapter::SetPen` and `EDrawAdapter::SetBrush`.
  - If you do not provide a pen and/or a brush, the default colors are used.

- The set the font of the text, provide a font with the method `EDrawAdapter::SetFont`.

Standard draw adapters

Open eVision provides a set of off-the-shelf draw adapters that you can use in different situations:

- `EWindowsDrawAdapter` allows to draw on **Windows** systems. To draw on a window, provide the window's HDC to its constructor, or, to draw in an `EImage` buffer, provide that `EImage`.
  - It relies on GDI and GDI+ to provide its services.
  - This is the preferred way to draw on **Windows**.

- `QtDrawAdapter` allows you to draw using **Qt** on a QPainter context. To draw on a QPainter context, provide the QPainter to the constructor, or, to draw on an `EImage` buffer, provide that `EImage`.
  - You can use the `QtDrawAdapter` both on **Windows** and **Linux**.
  - This is the preferred way to draw on **Linux**.

  NOTE: `QtDrawAdapter` is using an external resource (namely **Qt**) and as such is provided as source code in its own header rather than in the global **Open eVision** header. For more information about external and custom draw adapters, see below.

- `EGenericDrawAdapter` is a draw adapter that can only render on an `EImage`, but it can do it in a consistent manner on all supported OSes.
  - It is available on both **Windows** and **Linux**.

Drawing in an EImage

- As said above, you can draw in an `EImage` (usually an `EImageBW8` or `EImageC24`) by initializing a draw adapter with that image and using either the **Open eVision** methods `Draw` or the draw adapter drawing primitives:

```
EImageBW8 image;
EMatrixCode code;
EWindowsDrawAdapter adapter(image);
code.DrawPosition(adapter);
```

Custom draw adapters

- If you require a draw adapter to render in a specific, unsupported type of context (for ex. a DirectDraw surface, an OpenGL context...), you can build your own draw adapter by deriving from the interface `EExternalDrawAdapter` provided by **Open eVision** and implementing all the required methods.

- Once this work is done, you will be able to use your new, custom draw adapter in the same way as the off-the-shelf ones, taking advantage of **Open eVision** methods `Draw`.

- The provided `QtDrawAdapter` is a draw adapter built using that mechanism, you can use it as a reference on how to build a custom draw adapter. The sources of the `QtDrawAdapter` are bundled with the Qt Samples.

Enhanced Image Display

When the enhanced image display mode is enabled, a high-quality interpolation method is used to display the resized images.

- Set `Easy::SetEnableEnhancedImageDisplay(bool)` to `TRUE`, to enable the enhanced image display.
- By default, this option is disabled.
- Enhanced image display has a significant impact on display speed, the drawing can be 4x to 10x slower.
- The drawing of images with `EBW8Vector` or `EC24Vector` used as Look Up Table doesn't support enhanced image display
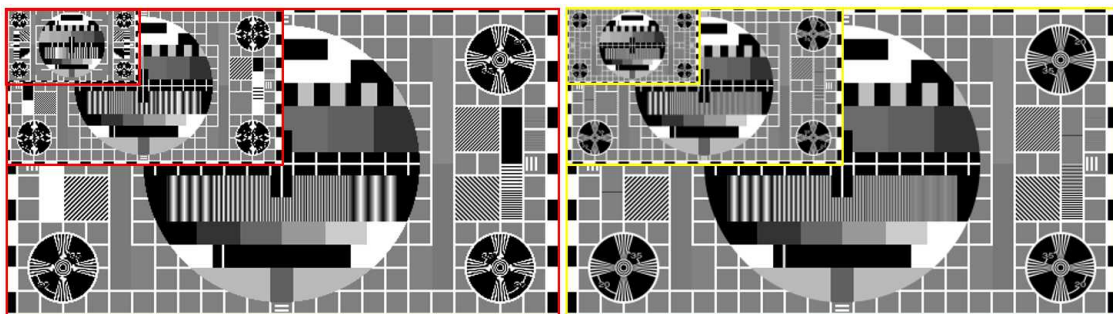
EnhancedImageDisplay disabled (left) and enabled (right)

- **Open eVision Studio** exposes this option in `View` > `Option` dialog:
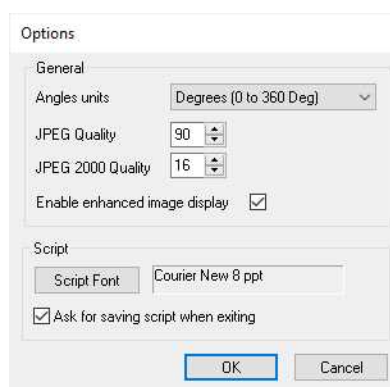
3.5. 3D Rendering of 2D Images

These images are viewed by rotating them around the X-axis, then the Y-axis.

Gray 3D rendering

Easy::Render3D prepares a 3-dimensional rendering where gray-level values are altitudes. Magnification factors in the three directions (X = width, Y = height and Z = depth) can be given. The rendered image appears as independent dots whose size can be adjusted to make the surface more or less opaque.
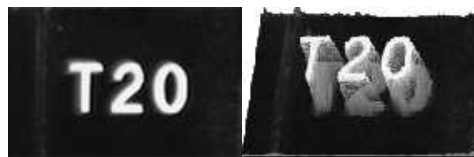
3D rendering

Color histogram 3D rendering

Easy::RenderColorHistogram prepares a 3-dimensional rendering of a color image histogram. The pixels are drawn in the RGB space (not XY-plane) to show clustering and dispersion of RGB values.
This function can process pixels in other color systems (using EasyColor to convert), but the raw RGB image is required to display the pixels in their usual colors.

Magnification factors in all three directions (X = red, Y = green and Z = blue) can be given.
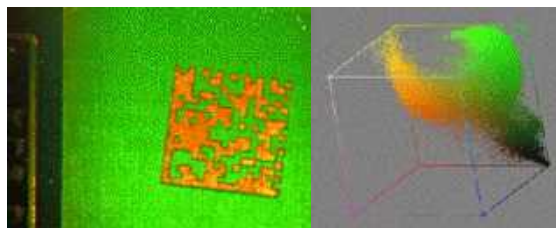
Color histogram rendering

3.6. Vector Types and Main Properties

A vector is a one-dimensional array of pixels (taken from an image profile or contour).

EVector is the base class for all vectors. It contains all non-type-specific methods, mainly for counting elements and serialization.
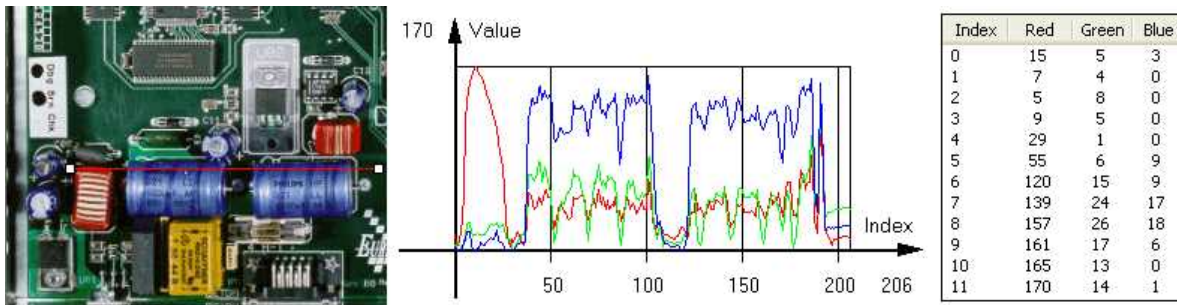
Profile in a C24 image, RGB values plot along profile and RGB values array (EC24Vector)

A vector manages an array of elements. Memory allocation is transparent, so vectors can be resized dynamically. Whenever a function uses a vector, the vector type, size and structure are automatically adjusted to suit the function needs.

The use of vectors is quite straightforward:

- To create a vector of the appropriate type:
  - □ Use its constructor and preallocate elements if required.

- To fill a vector with values:
  - □ Call the EVector::Empty member to empty it.
  - □ Call the EC24Vector::AddElement member to add elements one by one.
  - □ Use the indexing to access any element.

- To access a vector element, either for reading or writing:
  - □ Use the brackets operator EC24Vector::operator[].

- To determine the current number of elements:
  - □ Use the EVector::NumElements member.

- To draw the vector:
  - □ A pixel vector is a plot of the element values as a function of the element index, so its graphical appearance depends on its type. You can draw a vector in a window. For legibility, the drawing should appear on a neutral background.
  - □ Drawing is done in the device context associated to the desired window. By default, curves are drawn in blue and annotations in black. You can define: graphicContext, width, height, originX, originY, color0, color1 and color2.
  - □ The EC24Vector has three curves drawn instead of one, each corresponding to a color component. By default the red, blue and green pens are used.

Vector types

- **EBW8Vector**: a sequence of gray-level pixel values, often extracted from an image profile
  (used by EasyImage::Lut, EasyImage::SetupEqualize, EasyImage::ImageToLineSegment,
  EasyImage::LineSegmentToImage, EasyImage::ProfileDerivative...).
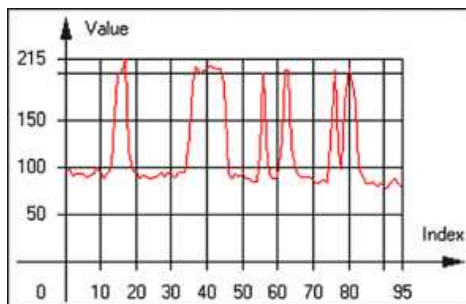
Graphical representation of an EBW8Vector (see Draw method)

- **EBW16Vector**: a sequence of gray-level pixel values, using an extended range (16 bits), mainly
  for intermediate computations.
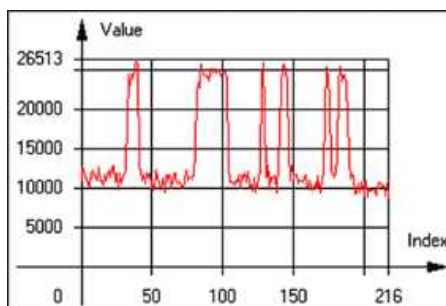
Graphical representation of an EBW16Vector

- **EBW32Vector**: a sequence of gray-level pixel values, using an extended range (32 bits), mainly
  for intermediate computations
  (used in EasyImage::ProjectOnARow, EasyImage::ProjectOnAColumn, ...).
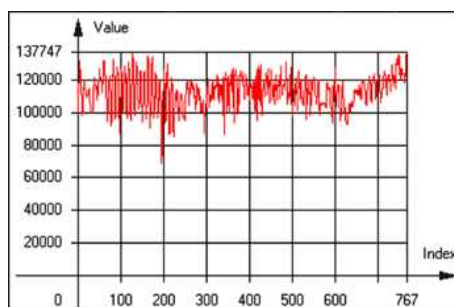
Graphical representation of an EBW32Vector

- **EC24Vector**: a sequence of color pixel values, often extracted from an image profile
  (used by `EasyImage::ImageToLineSegment`, `EasyImage::LineSegmentToImage`,
  `EasyImage::ProfileDerivative`, ...).
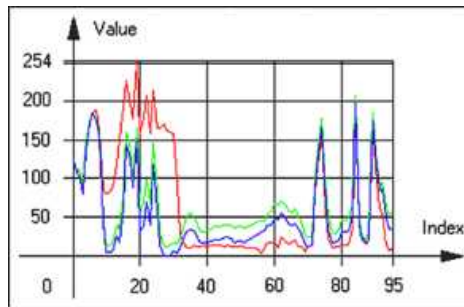
Graphical representation of an `EC24Vector`

- **EBW8PathVector**: a sequence of gray-level pixel values, extracted from an image profile or
  contour, with corresponding pixel coordinates
  (used by `EasyImage::ImageToPath`, `EasyImage::PathToImage`, ...).
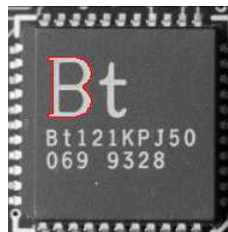
Graphical representation of an `EBW8PathVector` (see `Draw` method)

- **EBW16PathVector**: a sequence of gray-level pixel values, extracted from an image profile or
  contour, with corresponding pixel coordinates
  (used by `EasyImage::ImageToPath`, `EasyImage::PathToImage`, ...).

Graphical representation of an `EBW16PathVector` (see `Draw` method)

- EC24PathVector: a sequence of color pixel values, extracted from an image profile or contour, with corresponding pixel coordinates
  (used by EasyImage::ImageToPath, EasyImage::PathToImage, ...).
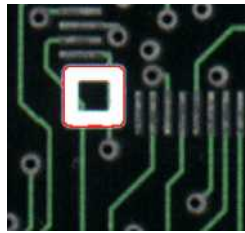
Graphical representation of an EC24PathVector (see Draw method)

- EBWHistogramVector: a sequence of frequency counts of pixels in a BW8 or BW16 image
  (used by EasyImage::IsodataThreshold, EasyImage::Histogram, EasyImage::AnalyseHistogram, EasyImage::SetupEqualize, ...).
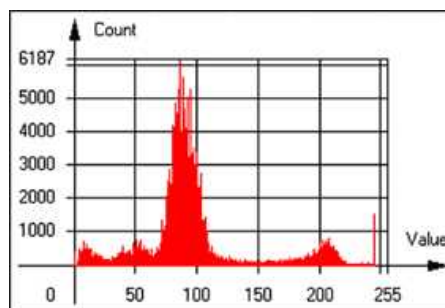
Graphical representation of an EBWHistogramVector (see Draw method)

- EPathVector: a sequence of pixel coordinates. The corresponding pixels need not be contiguous
  (used by EasyImage::PathToImage and EasyImage::Contour).
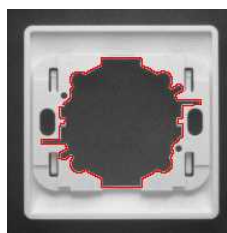
Graphical representation of an EPathVector (see Draw method)

- EPeakVector: peaks found in an image profile
  (used by EasyImage::GetProfilePeaks).

- EColorVector: a description of colors
  (used by EasyColor::ClassAverages and EasyColor::ClassVariances).

3.7. ROI Main Properties

ROIs are defined by a width, a height, and **origin** x **and** y **coordinates**.
The origins are specified with respect to the top left corner in the parent image or ROI.
The ROI must be wholly contained in its parent image.
The processing/analysis time of a BW1 ROI is faster if OrgX and Width are multiples of 8.

Save and load

You can save or load an ROI as a separate image, to be used as if it was a full image. The ROIs perform **no memory allocation** at all and never duplicate parts of their parent image, the parent image provides them with access to its image data.

The image size of the new file must match the size of the ROI being loaded into it. The image around the ROI remains unchanged.

ROI Classes

An Open eVision ROI inherits parameters from the abstract class EBaseROI.

There are several ROI types, according to their pixel type. They have the same characteristics as the corresponding image types.

- □ EROIBW1
- □ EROIBW8
- □ EROIBW16
- □ EROIBW32
- □ EROIC15
- □ EROIC16
- □ EROIC24
- □ EROIC24A

Attachment

An ROI must be attached to a parent (image/ROI) with parameters that set the parent, position and size, and these links are updated transparently, avoiding dangling pointers.
A normal image cannot be attached to another image or ROI.

Nesting

Set and Get functions change or query the width, height and position of the origin of an ROI, with respect to its immediate or topmost parent image.

An image may accommodate an arbitrary number of ROIs, which can be nested in a hierarchical way. Moving the ROI also moves the embedded ROIs accordingly. The image/ROI classes provide several methods to traverse the hierarchy of ROIs associated with an image.
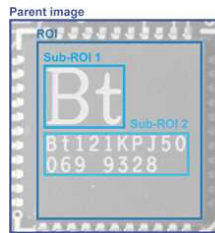
Nested ROIs: Two sub-ROIs attached to an ROI, itself attached to the parent image

Cropping

CropToImage crops an ROI which is partially out of its image. The resized ROI never grows.
An exception is thrown if a function attempts to use an ROI that has limits that extend outside of the parents.

> **NOTE**
> *(In Open eVision 1.0.1 and earlier, an ROI was silently resized or repositioned when placed out of its image and sometimes grew. If ROI limits extended outside parents, they were silently resized to remain within parent limits.)*

Resizing and moving

ROIs can easily be resized and positioned by two functions and dragging handles:

- EBaseROI.Drag adjusts the ROI coordinates while the cursor moves.

- EBaseROI.HitTest informs if the cursor is placed over a dragging handle.
  - ☐ Once the handle is known, the cursor shape can be changed by an OnSetCursor MFC event handler. HitTest is unpredictable if called while dragging is in progress.
  - ☐ HitTest can be used in an OnSetCursor MFC event handler to change the cursor shape, or before a dragging operation like OnLButtonDown,
    (or EvSetCursor and EvLButtonDown in Borland/CodeGear's OWL)
    (or FormMouseMove and FormMouseDown in Borland/CodeGear's VCL).

3.8. Arbitrarily Shaped ROI (ERegion)

See also: example: Inspecting Pads Using Regions / code snippets: ERegion

Regions or arbitrarily shaped ROI

You define and use regions of interest (ROI) to restrict the area processed with your vision tool and to reduce and optimize the processing time.

In **Open eVision**:

- An **ROI** (EROIxxx class) designates a rectangular region of interest.
- A **region** (ERegion class) designates an arbitrarily shaped ROI. With regions, you can determine precisely which part of the image, down to a single pixel, is used for your processing.

Currently, only the following **Open eVision** methods support ERegions:

Library	Method
EasyImage	EasyImage::Threshold

Library	Method
	EasyImage::AutoThreshold

Library	Method
	EasyImage::Copy

Library	Method
	EasyImage::ConvolKernel

Library	Method
	EasyImage::ConvolSymmetricKernel

Library	Method
	EasyImage::ConvolLowpass1

Library	Method
	EasyImage::ConvolLowpass2

Library	Method
	EasyImage::ConvolLowpass3

Library	Method
	EasyImage::ConvolUniform

Library	Method
	EasyImage::ConvolGaussian

Library	Method
	EasyImage::ConvolHighpass1

Library	Method
	EasyImage::ConvolHighpass2

Library	Method
	EasyImage::ConvolGradientX

Library	Method
	EasyImage::ConvolGradientY

Library	Method
	EasyImage::ConvolGradient
	EasyImage::ConvolSobelX
	EasyImage::ConvolSobelY
	EasyImage::ConvolSobel
	EasyImage::ConvolPrewittX
	EasyImage::ConvolPrewittY
	EasyImage::ConvolPrewitt
	EasyImage::ConvolRoberts
	EasyImage::ConvolLaplacianX
	EasyImage::ConvolLaplacianY
	EasyImage::ConvolLaplacian8
	EasyImage::DilateBox
	EasyImage::ErodeBox
	EasyImage::OpenBox
	EasyImage::CloseBox
	EasyImage::WhiteTopHatBox
	EasyImage::BlackTopHatBox
	EasyImage::MorphoGradientBox
	EasyImage::ErodeDisk
	EasyImage::DilateDisk
	EasyImage::OpenDisk
	EasyImage::CloseDisk
	EasyImage::WhiteTopHatDisk
	EasyImage::BlackTopHatDisk
	EasyImage::MorphoGradientDisk
	EasyImage::Median
	EasyImage::ScaleRotate
	EasyImage::DoubleThreshold
	EasyImage::Histogram
	EasyImage::Area
	EasyImage::AreaDoubleThreshold
	EasyImage::BinaryMoments
	EasyImage::WeightedMoments
	EasyImage::GravityCenter
	EasyImage::PixelCount
	EasyImage::PixelMax
	EasyImage::PixelMin
	EasyImage::PixelAverage
	EasyImage::PixelStat
	EasyImage::PixelVariance
	EasyImage::PixelStdDev
	EasyImage::PixelCompare
	EasyImage::ImageToLineSegment
	EasyImage::ImageToPath

Library	Method
Easy3D	`EDepthMapToMeshConverter::Convert`
	`EDepthMapToPointCloudConverter::Convert`
	`EStatistics::ComputePixelStatistics`
	`EStatistics::ComputeStatistics`
	`E3DObjectExtractor::Extract`
	`EZMapToPointCloudConverter::Convert`
EasyObject	`EImageEncoder::Encode`
EasyFind	`EPatternFinder::Find`
	`EPatternFinder::Learn`
EasyOCR2	`EOCR2::Read`
	`EOCR2::Detect`
EasyGauge	`EPointGauge::Measure`
	`ELineGauge::Measure`
	`ERectangleGauge::Measure`
	`ECircleGauge::Measure`
	`EWedgeGauge::Measure`
EasyMatch	`EMatcher::LearnPattern`
	`EMatcher::Match`
EasyQRCode	`EQRCodeReader::SetSearchField`
	`EQRCodeReader::Read`

> ✓ **TIP**
>
> In the future **Open eVision** releases, the support of ERegions will be gradually extended to all operators.

Creating regions

Open eVision offers multiple ways to create regions, depending on the shape you need:

The ERegion is the base class for all regions and the most versatile. It encodes a region using a Run-Length Encoded (RLE) representation.

☐ The RLE representation of a region is made of runs (horizontal, 1-pixel high slices).

☐ The runs are stored in the form of their ordinate, starting abscissa and length.
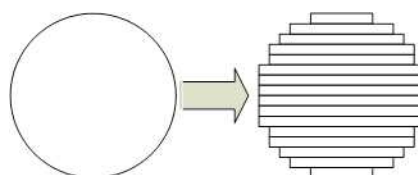
Run-Length Encoding of a circle-shaped region

To create a region, either:

- ☐ Use one of the geometry-based region classes.
- ☐ Use the result of another tool, such as **EasyFind**, **EasyMatch** or **EasyObject**.
- ☐ Combine or modify other regions.
- ☐ Use a mask image.
- ☐ Directly provide the list of runs.

Geometry-based regions

Geometry based regions are specialized classes of regions that are encompassed in simple geometries. **Open eVision** currently provides classes based on a rectangle, a circle, an ellipse or a polygon.

Use these classes to setup geometric regions and modify them with translation, rotation and scaling. The transformation operators return new regions, leaving the source object unchanged.

● ERectangleRegion

- ☐ The contour of an ERectangleRegion class is a rectangle.
- ☐ Define it using its center, width, height and angle.
- ☐ Alternatively, use an ERectangle instance, such as one returned by an ERectangleGauge instance.

Rectangle region separating a bar code from the background

● ECircleRegion

- ☐ The contour of an ECircleRegion class is a circle.
- ☐ Define it using its center and radius or 3 non-aligned points.
- ☐ Alternatively, use an ECircle instance, such as one returned by an ECircleGauge instance.
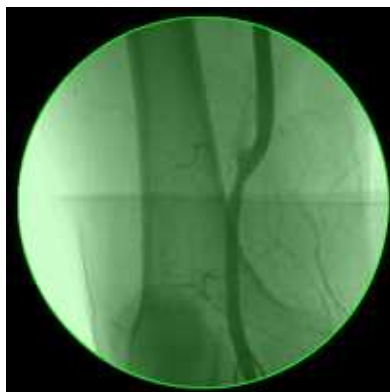
Circle region encompassing the useful part of an X-Ray image

- EEllipseRegion
    - □ The contour of an EEllipseRegion class is an ellipse.
    - □ Define it using its center, long and short radius and angle.
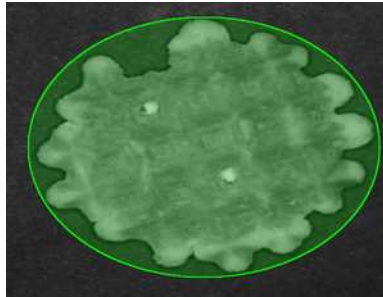
Ellipse region encompassing a waffle

- EPolygonRegion
    - □ The contour of an EPolygonRegion class is a polygon.
    - □ It is constructed using the list of its vertices.

Polygon region encompassing a key

Using the result of other tools

The ERegion class provides a set of specialized constructors to create regions from the results of another tool.

In a tool chain, these constructors restrict the processing of a tool to the area issued from the previous tool.

Open eVision provides constructors for the following tools:

- ☐ EasyFind: `EFoundPattern`
- ☐ EasyMatch: `EMatchPosition`
- ☐ EasyGauge: `ECircle` and `ERectangle`
- ☐ EasyObject: `ECodedElement`

> ✅ **TIP**
>
> When compatible, **Open eVision** also provides specialized constructors for the geometry-based regions. For instance, `ECircleRegion` provides a constructor using an `ECircle`.

Combining regions

Use the following operations to create a new region by combining existing regions:

- ● Union
  - ☐ The `ERegion::Union(const ERegion&, const ERegion&)` method returns the region that is the addition of the two regions passed as arguments.
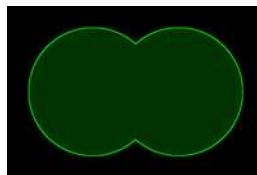
Union of 2 circles

- ● Intersection
  - ☐ The `ERegion::Intersection(const ERegion&, const ERegion&)` method returns the region that is the intersection of the two regions passed as argument.
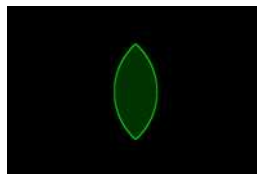
Intersection of 2 circles

- Subtraction
  - The `ERegion::Subtraction(const ERegion&, const ERegion&)` method returns the first region passed as argument after removing the second one.
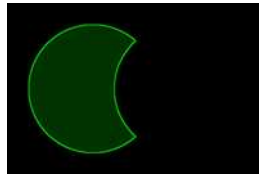
Subtraction of 2 circles

Morphological operations on regions
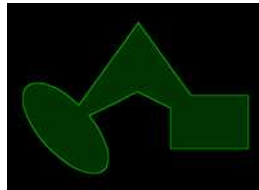
The initial arbitrary region used to illustrate the different morphological operations

- Grow
  - The `ERegion::Grow(int radius)` method returns a region that is the dilation of the region by a disk with a radius equals to the argument.
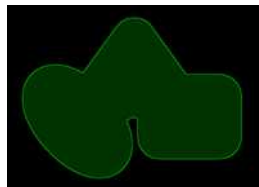
Grow of the arbitrary region

- Shrink
  - The `ERegion::Shrink(int radius)` method returns a region that is the erosion of the region by a disk with a radius equals to the argument.
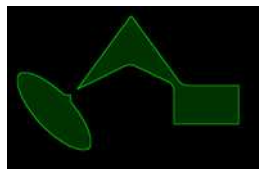
Shrink of the arbitrary region

- Contour
  - □ The `ERegion::Contour(int thickness, bool centered = true)` method returns a region that is the contour of the region.
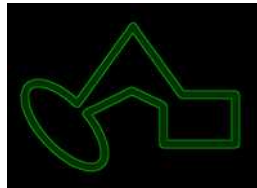
Contour of the arbitrary region

Free-hand drawing a region

- The `ERegionFreeHandPainter` class provides the methods that allow you to create a region by hand, using the mouse or any other user input method.

- The `RegionFreeHand` sample, available both in C++ and C#, shows how to use this class to draw a region on an image.

Using regions

The tools supporting regions provide methods that follow one of these conventions:
  - □ `Method(const EImage& source, const ERegion& region)`
  - □ `Method(const EImage& source, const ERegion& region, EImage& destination)`

> **NOTE**
> The source, the region and the destination must be compatible. It means that the region must at least partly fit in the source, and that source and destination must have the same size.

Preparing the region

- **Open eVision** automatically prepares the regions when it applies them to an image, but this preparation can take some time.

- If you do not want your first call to a method to take longer than the next ones, you can prepare the region in advance by using the appropriate `Prepare()` method.

- To manually prepare the regions, adapt the internal RLE description to your images.

Drawing regions

The `ERegion` classes provide several methods to display the regions:

- `ERegion::Draw()` draws the region area, in a semi-transparent way, in the provided device context.

- `ERegion::DrawContour()` draws the region contour in the provided device context.

- `ERegion::ToImage()` renders the region as a mask into the provided destination image.
  - You can configure the foreground and the background colors.
  - If you initialized your image with a width and a height, **Open eVision** renders the region inside those bounds.
  - If not, **Open eVision** resizes the image to contain the whole region.
  - Use `ToImage()` to create masks for the **Open eVision** functions that support them.

ERegions and EROIs

- The older `EROI` classes of **Open eVision** are compatible with the new regions.
- Some tools allow the usage of regions with source and/or destinations that are `ERoi` instead of `EImage` follow one of these conventions:
  - `Method(const ERoi& source, const ERegion& region)`
  - `Method(const ERoi& source, const ERegion& region, ERoi& destination)`

> **TIP**
> In that case, the coordinates used for the region are relative to the reduced ROI space instead of the whole image space .

ERegion and 3D

- The new regions are compatible with the 2.5D representations of Easy3D (`EDepthMap` and `EZMap`).
- You can also reduce the domain of processing when using these classes.

3.9. Flexible Masks

ROIs vs flexible masks

ROIs and masks restrict processing to part of an image:
  - "ROI Main Properties" on page 27 apply to all Open eVision functions. Using Regions of Interest accelerates processing by reducing the number of pixels. Open eVision supports hierarchically nested rectangular ROIs.
  - Flexible Masks are recommended to process disconnected ROIs or non-rectangular shapes. They are supported by some EasyObject and EasyImage library functions.

Flexible Masks

A flexible mask is a BW8 image with the same height and width as the source image. It contains shapes of areas that must be processed and ignored areas (that will not be considered during processing):

☐ All pixels of the flexible mask having a value of 0 define the ignored areas.

☐ All pixels of the flexible mask having any other value than 0 define the areas to be processed.
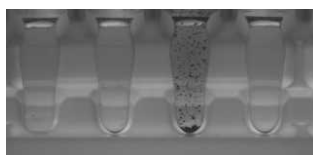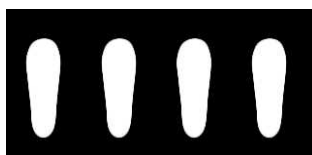
| Source image | Associated mask | Processed masked image |
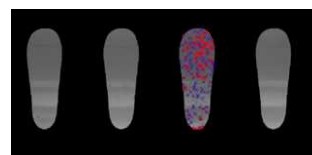
A flexible mask can be generated by any application that outputs BW8 images and by some EasyObject and EasyImage functions.

Flexible Masks in EasyImage

Code Snippets
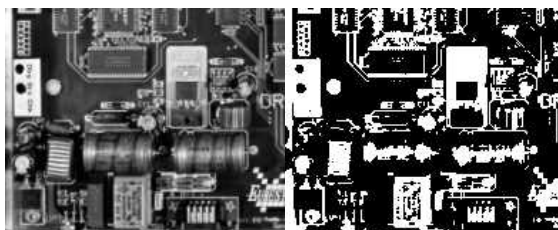
Source image (left) and mask variable (right)

Simple steps to use flexible masks in Easyimage

1. **Call the functions from EasyImage that take an input mask as an argument**. For instance, one can evaluate the average value of the pixels in the white layer and after in the black layer.

2. **Display the results.**
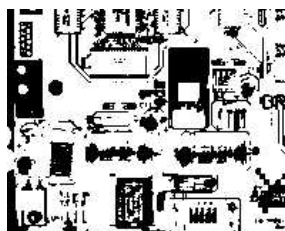
Resulting image

EasyImage Functions that support flexible masks

- `EImageEncoder.Encode` has a flexible mask argument for BW1, BW8, BW16, and C24 source images.

- AutoThreshold.

- Histogram (function `HistogramThreshold` has no overload with mask argument).

- RmsNoise, SignalNoiseRatio.

- Overlay (no overload with mask argument for BW8 source images).

- ProjectOnAColumn, ProjectOnARow (Vector projection).

- ImageToLineSegment, ImageToPath (Vector profile).

Flexible Masks in EasyObject

A flexible mask can be generated by any application that outputs BW8 images or uses the Open eVision image processing functions.

EasyObject can use flexible masks to restrict blob analysis to complex or disconnected shaped regions of the image.

If an object of interest has the same gray level as other regions of the image, you can define "keep" and "ignore" areas using flexible masks and Encode functions.

A flexible mask is a BW8 image with the same height and width as the source image.
   - A pixel value of 0 in the flexible mask masks the corresponding source image pixel so it doesn't appear in the encoded image.
   - Any other pixel value in the flexible mask causes the pixel to be encoded.

EasyObject functions that create flexible masks
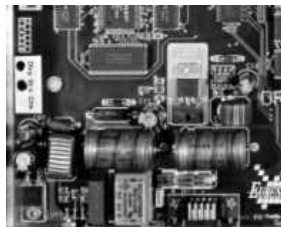
Source image

1) ECodedImage2.RenderMask: from a layer of an encoded image

1. To encode and extract a flexible mask, first construct a coded image from the source image.

2. Choose a segmentation method (for the image above the default method `GrayscaleSingleThreshold` is suitable).

3. Select the layer(s) of the coded image that should be encoded (i.e. white and black layers using minimum residue thresholding).

4. Make the mask image the desired size using `mask.SetSize(sourceImage.GetWidth(), sourceImage.GetHeight())`.

5. Exploit the flexible mask as an argument to ECodedImage2.RenderMask.
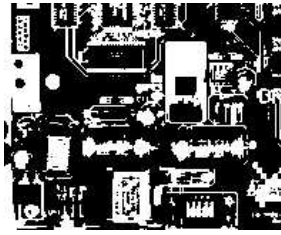
BW8 resulting image that can be used as a flexible mask

2) ECodedElement.RenderMask: from a blob or hole

1. Select the coded elements of interest.

2. Create a loop extracting a mask from selected coded elements of the coded image using ECodedElement.RenderMask.

3. Optionally, compute the feature value over each of these selected coded elements.
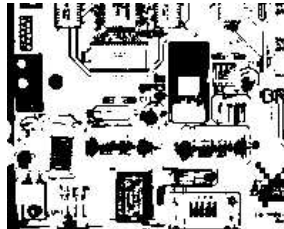
BW8 resulting image that can be used as a flexible mask

3) EObjectSelection.RenderMask: from a selection of blobs

EObjectSelection.RenderMask can, for example, discard small objects resulting from noise.
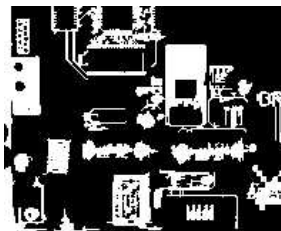
BW8 resulting image that can be used as a flexible mask

Example: Restrict the areas encoded by EasyObject
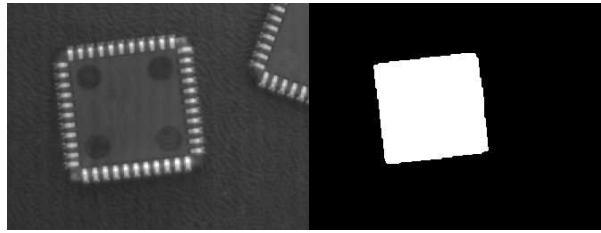
Find four circles (left) Flexible mask can isolate the central chip (right)

1. Declare a new ECodedImage2 object.

2. Setup variables: first declare source image and flexible mask, then load them.

3. Declare an EImageEncoder object and, if applicable, select the appropriate segmenter. Setup the segmenter and choose the appropriate layer(s) to encode.

4. Encode the source image. Encoding a layer with just the area in the flexible mask is then pretty straightforward.
We see that the circles are correctly segmented in the black layer with the grayscale single threshold segmenter:
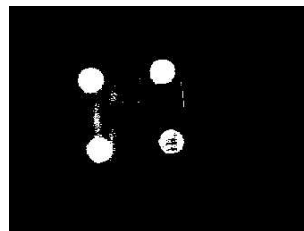
5. Select all objects of the coded image.

6. Select objects of interest by filtering out objects that are too small.

7. Display the blob feature by iterating over the selected objects to display the chosen feature.

3.10. Profile

Code Snippets

Profile Sampling

A **profile** is a series of pixel values sampled along a line/path/contour in an image.

- EasyImage.ImageToLineSegment copies the pixel values along a given line segment (arbitrarily oriented and wholly contained within the image) to a vector. The vector length is adjusted automatically. This function supports flexible masks.

- A path is a series of pixel coordinates stored in a vector.
EasyImage.ImageToPath copies the corresponding pixel values to the vector. This function supports flexible masks.

- A **contour** is a closed or not (connected) path, forming the boundary of an object.
`EasyImage.Contour` follows the contour of an object, and stores its constituent pixels values inside a profile vector.

Profile Analysis

The profile can be processed to find peaks or transitions:

- A transition corresponds to an object edge (black to white or white to black). It can be detected by taking the first **derivative** of the signal (which transforms transitions (edges) into peaks) and looking for peaks in it.
`EasyImage.ProfileDerivative` computes the first derivative of a profile extracted from a gray-level image.
The `EBW8` data type only handles unsigned values, so the derivative is shifted up by 128. Values under [above] 128 correspond to negative [positive] derivative (decreasing [increasing] slope).

- A peak is the portion of the signal that is above [or below] a given threshold - the maximum or minimum of the signal. This may correspond to the crossing of a white or black line or thin feature. It is defined by its:

  □ Amplitude: difference between the threshold value and the max [or min] signal value.

  □ Area: surface between the signal curve and the horizontal line at the given threshold.

`EasyImage.GetProfilePeaks` detects max and min peaks in a gray-level profile. To eliminate false peaks due to noise, two selection criteria are used. The result is stored in a peaks vector.

Profile Insertion Into an Image

`EasyImage.LineSegmentToImage` copies the pixel values from a vector or constant to the pixels of a given line segment (arbitrarily oriented and wholly contained within the image).

`EasyImage.PathToImage` copies the pixel values from a vector or a constant to the pixels of a given path.

4. Text and Code Reading Tools

4.1. List of Supported Codes

Linear bar codes (1D)

- License: **EasyBarCode**
- Class: EBarCodeReader

Symbology	Variants	Checksum	Error correction	Multiple codes reader	Support of grading	Sample
Code 128	—	✓	—	✓	✓	
Ean 8	—	✓	—	✓	—	
Ean 13	—	✓	—	✓	✓	
GS1-128	—	✓	—	✓	✓	
Code 39	Extended and Reduced	Optional	—	✓	—	
Code 93	Extended	✓	—	✓	—	
GS1 DataBar Omnidirectional	RSS14	✓	✓	✓	—	
GS1 DataBar Limited	RSS14 Limited	✓	✓	✓	—	
GS1 DataBar Expanded	RSS14 Expanded	✓	✓	✓	—	
PharmaCode One Track	—	—	—	✓	—	
Codabar	—	Optional	—	✓	—	
Code 2 of 5 Interleaved	—	Optional	—	✓	—	
MSI	—	✓	—	✓	—	
UPC-A	—	✓	—	✓	—	
UPC-E	—	✓	—	✓	—	

Symbology	Variants	Checksum	Error correction	Multiple codes reader	Support of grading	Sample
ADS Anker	—	✓	—	✓	—	
BC 412	—	✓	—	✓	—	
Code 11	—	Optional	—	✓	—	
Code 13	—	✓	—	✓	—	—
Code 2 of 5	Datalogic, Matrix, IATA, Industry, Compressed and Inverted	Optional	—	✓	—	
Code 32	—	✓	—	✓	—	
Code BCD Matrix	—	✓	—	✓	—	
Code CIP	—	✓	—	✓	—	—
IBM Delta Distance A	—	✓	—	✓	—	—
Plessey	—	✓	—	✓	—	
Telepen	—	✓	—	✓	—	
STK Code	—	—	—	✓	—	—
Binary Code	—	—	—	✓	—	

Mail bar codes (1D)

- License: **EasyBarCode**

- Class: `EMailBarcodeReader`

Symbology	Variants	Checksum	Error correction	Multiple codes reader	Support of grading	Sample
Intelligent Mail Barcode	—	✓	—	✓	—	
Japan Post 4-state Barcode	—	✓	—	✓	—	
POSTNET	POSTNET 5, 6, 9 and 11	✓	—	✓	—	
PLANET	—	✓	—	✓	—	

Matrix codes (2D)

- License: **EasyMatrixCode**
- Class: `EMatrixCodeReader`

Symbology	Variants	Error correction	Multiple codes reader	Support of grading	Sample
Datamatrix ECC000	—	—	✓	✓	
Datamatrix ECC050	—	✓	✓	✓	
Datamatrix ECC080	—	✓	✓	✓	
Datamatrix ECC100	—	✓	✓	✓	
Datamatrix ECC140	—	✓	✓	✓	
Datamatrix ECC200	DMRE	✓	✓	✓	

QR codes (2D)

- License: **EasyQRCode**
- Class: `EQRCodeReader`

Symbology	Variants	Error correction	Multiple codes reader	Support of grading	Sample
QRCode MicroQR	—	✓	✓	✓	
QRCode Model 1	—	✓	✓	✓	
QRCode Model 2	—	✓	✓	✓	

4.2. ECodeReader - Unified Interface

Reading Codes

ECodeReader integrates the functionality of **EasyBarCode2**, **EasyMatrixCode2** and **EasyQRCode** in a single unified interface.

In a single Read operation, **ECodeReader** locates and decodes the 3 kinds of codes supported by **Open eVision**, namely bar codes, QR codes and data matrix codes.

For more information about these codes and the associated support, please refer to the EasyBarCode2, EasyMatrixCode2 and EasyQRCode user guides.
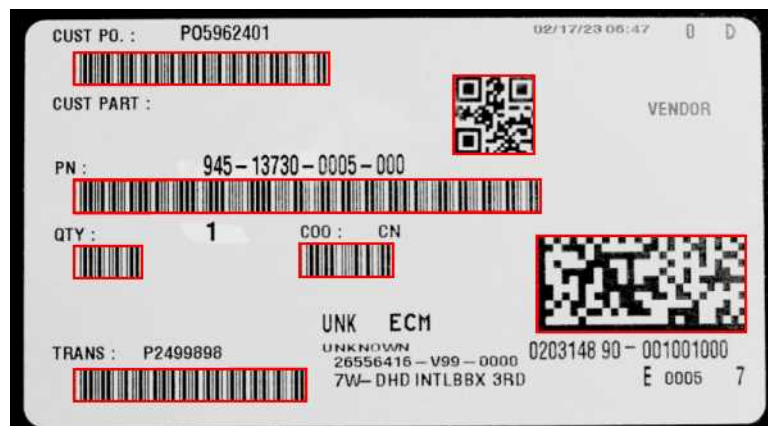
The 3 supported codes in a single image

Reading codes

Use the method `ECodeReader.Read` to locate and read all the codes in an image.

- By default, ECodeReader searches for all supported types of code in the image.
  - □ Use the property `ECodeReader.EnableCodeTypes` if you want to disable (or enable) specific code types.

- Read returns a vector of ECode, one for each code detected in the image.
  - □ Use the property `ECodeReader.MaxNumCodesPerType` to set the maximum number of codes to search for each type.

- Use the property `DecodedString` property of ECode to retrieve the contents of a code.
  - □ Use the property `CodeType` if you need to know the specific type of the code.

Code specific results

Use the following properties to retrieve the information specific to a given type of code from an ECode instance:

- `ECode.BarCode` returns an `EasyBarCode2.EBarCode` object.

  📄 For more information, refer to the documentation of EBarCode2.

- `ECode.MatrixCode` returns a `EasyMatrixCode2.EMatrixCode` object.

  📄 For more information, refer to the documentation of `EMatrixCode2`.

- `ECode.QRCode` returns an `EQRCode` object.

  📄 For more information, refer to the documentation of `EQRCode`.

NOTE: Depending on the effective type of the code, only one of the properties returns a value. The others throw an exception. To know the property to call, use `ECode.CodeType`.

Drawing codes

Use the method `DrawPosition` of the `ECode` instance to draw the position of a code.

Code type specific settings

Use the following properties to set the settings specific to one of the supported types of code:

- `ECodeReader.BarCodeReader` retrieves the underlying instance `EasyBarCode2.EBarCodeReader`.

  📄 For more information about the relevant settings, refer to the documentation of `EBarCode2`.

- `ECodeReader.MatrixCodeReader` retrieves the underlying instance `EasyMatrixCode2.EMatrixCodeReader`.

  📄 For more information about the relevant settings, refer to the documentation of `EMatrixCode2`.

- `ECodeReader.QrCodeReader` retrieves the underlying instance `EQRCodeReader`.

  📄 For more information about the relevant settings, refer to the documentation of `EQRCode`.

Multithreading

ECodeReader always parallelizes the location and the decoding of the different types of code.

- `Easy.MaxNumberOfProcessingThreads` has no effect on this behavior.

- However, `Easy.MaxNumberOfProcessingThreads` has the intended effect on the underlying code readers and speeds up the whole process.

Time-out

Use the property `TimeOut` to set a time-out to the `ECodeReader.Read` process.

- The time-out is set for each of the underlying code readers that run in parallel. And so it also limits the total `Read` processing time.

- If one of the supported types of code is not present in the image, it is recommended to disable that type. Otherwise the corresponding reader runs and may reach its time-out, slowing the whole process.

Reading Using a Grid

If the codes in the images are arranged in a regular grid-like fashion:

- Use the dedicated method overloads `ECodeReader.Read` that return `ECodeGrid` objects to improve the reliability and the speed of the reading.

- Use the methods `ECodeGrid.SetEnableCell`, `ECodeGrid.SetEnableRow`, `ECodeGrid.SetEnableColumn` or `ECodeGrid.SetEnableAll` to disable the processing of cells that you know do not contain any code.

- Use the method `GetResults` of the returned `ECodeGrid` object to retrieve the codes read in each cell of the defined grid.

Reading of codes arranged in a grid

4.3. EasyBarCode - Reading Bar Codes

Reading Bar Codes

Reference | Code Snippets

Bar code (EAN 13 symbology)

EasyBarCode can locate and read bar codes automatically.
Location can be performed manually for prototyping or when automatic mode results are unsatisfactory.

Workflow
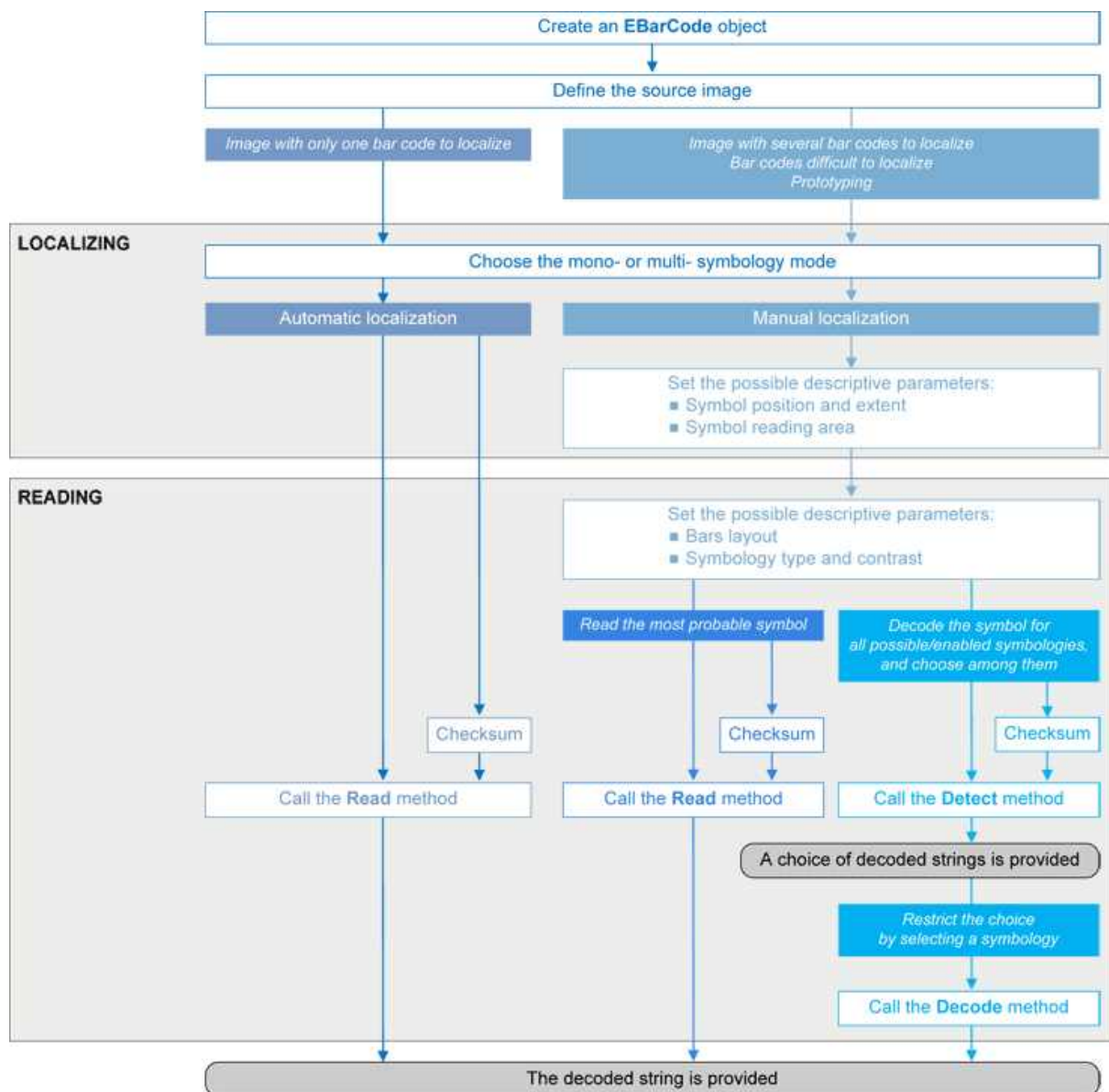
Bar code definition

A bar code is a 2D pattern of parallel bars and spaces of varying thickness that represents a character string. It is arranged according to an encoding convention (**symbology**) that specifies the character set and encoding rules.

- The bar code may be black ink on white background or inversely white ink on black background.
- The bar code should be preceded and followed by a quiet zone of at least ten times the module width (smallest bar or space thickness).
- Bars should be surrounded below and above by a quiet zone of a few pixels.
- Bars and spaces widths must be greater than or equal to 2 pixels.

Symbologies

A symbology defines the way a bar code is encoded.

Symbologies can be enabled in `StandardSymbologies` or `AdditionalSymbologies` parameters.

The standard symbologies are enabled by default:

- Code 39
- Code 128
- Code 2/5 5 Interleaved
- Codabar
- EAN 13*
- EAN 128
- MSI
- UPC A*
- UPC E

> **NOTE**
> * EAN 13 and UPC A only differ by the layout of surrounding digits.

Additional symbologies that are supported:

- ADS Anker
- Binary code
- Code 11
- Code 13
- Code 32
- Code 39 Extended (a super-set of Code 39)
- Code 39 Reduced (a subset of Code 39)
- Code 93
- Code 93 Extended
- Code 412 SEMI
- Code 2/5 3 Bars Datalogic
- Code 2/5 3 Bars Matrix
- Code 2/5 5 Bars IATA
- Code 2/5 5 Bars Industry
- Code 2/5 5 Compressed
- Code 2/5 5 Inverted
- Code BCD Matrix
- Code C.I.P
- Code STK
- EAN 8
- IBM Delta Distance A
- Plessey
- Telepen

Checksum

A checksum character enables the reader to check the bar code validity depending on the symbology:

- The checksum may be mandatory and must be checked by the reader.
- The checksum may be mandatory but may not need to be checked.
- The checksum and its verification may both be optional.

`VerifyChecksum` enables or disables (default) checksum verification.
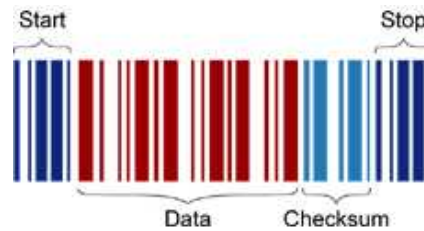
Bar code structure (Code 39)

Read a bar code

The **Automatic** mode reading algorithm locates a bar code in the field of view and Reads it.
If several bar codes are present, only one is located, like a straightforward hand-held bar code reader.

Before reading, the decoding symbologies must be specified in the StandardSymbologies, or AdditionalSymbologies properties.

Mono-symbology mode reads the bar code using the expected symbology type(s) and reports the encoded information (if readable) or the reason for failure (if not readable). There is only one interpretation for the character string.

Decoded bar code

Note: When the bar code contains \0x00 characters, the `std.string.c_str` method should not be used (since C-strings are terminated by the \0x00 character). An iterator over the characters should be used instead of a C-string.

Advanced features

Locate and Read bar code manually

If automatic localization fails or for prototyping purposes, the user can provide the **bar code position** and **reading area** to manually locate the code.

- **Bar code position** can be provided graphically by a bounding box around the bar code or by its parameters. If several symbols appear in the image, they can be processed one after the other.
- The **reading area** of the bar code is the area that is read. It should be wider than the bar code bounding box width, and less high than the bar code bounding box height. It may also be rotated relative to the bar code bounding box, to take into account slanting bars (Advanced mode!).

Bounding box — graphical appearance (manual location)
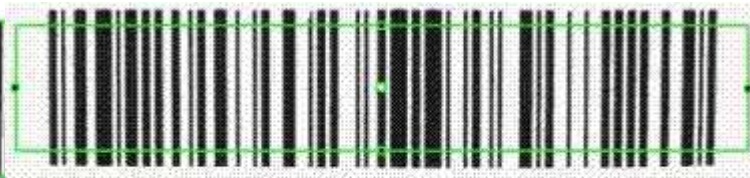
Reading area — graphical appearance (manual location)

Read all interpretations (multi-symbology mode)

Use `Detect` to report the number of possible symbologies in the `NumEnabledSymbologies` property, and list the data contents by decreasing likeliness.

Then call the `Decode` method in a loop, using `GetDecodedSymbology` to walk through the list of successful symbologies in decreasing order of likelihood.

Reading Mail Bar Codes

Reference | Code Snippets
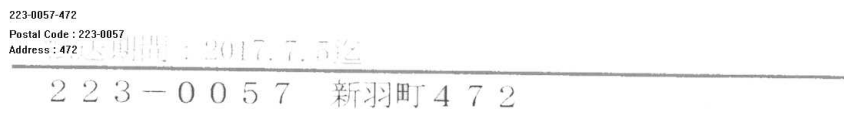
Mail bar code example

Specifications

The Mail Bar code Reader:
- Detects and decodes postal 4-state bar codes.
- Supports multiple mail bar codes in an image.
- Supports various symbologies.
- Supports the 4 main bar code orientations, with a tolerance of 3°.
- Detects bars that are at least 3 pixels wide.

Workflow
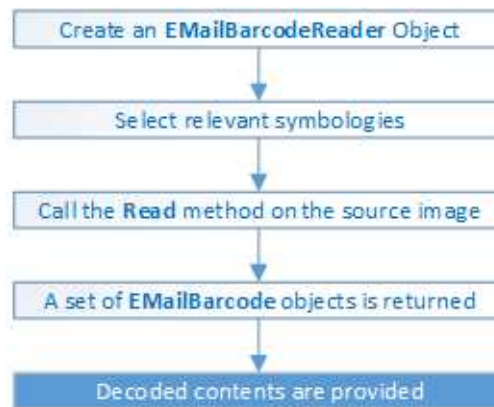
4-state bar codes

A 4-state bar code is a special kind of bar code where data is encoded on the height and position of the bars rather than their width.

Each bar can have one of 4 possible states:
- Short and centered
- Medium and elevated
- Medium and lowered
- Full height

Mail bar code symbologies

The symbology of a mail bar code specifies how to decode the bar code and how to interpret its contents.

Every country uses its own flavor of mail bar code, or symbology. Some countries, like the US, even use multiple symbologies.

As of now, the Open eVision Mail Bar code Reader supports the following symbologies:
- US: PLANET, POSTNET and Intelligent Mail
- Japan: Japan Post

Mail bar code orientation

The Open eVision Mail Bar code Reader is designed to be used in mail-handling machines. As such it is optimized to handle the 4 main orientations you encounter in such machines:
- No Rotation: The mail bar code is horizontal and read from left to right
- Rotated 90° to the right: The mail bar code is vertical and read from top to bottom

- □ Rotated 90° to the left: The mail bar code is vertical and read from bottom to top
- □ Rotated 180°: The mail bar code is upside down, horizontal, and read from right to left.

For each of these orientations, an additional rotation of -3 to 3 degrees is allowed.

Checksum

Some symbologies specify the presence of a checksum in the bar code data.

This checksum is an additional character computed from all others encoded characters. It enables the reader to check the decoded character string coherence.

- The Mail Bar code Reader allows the user to verify or not the checksum for all enabled symbologies.
- By default, checksum is not controlled.
- To enable or disable checksum verification for all enabled symbologies, set the `ValidateChecksum` property.

Reading the mail bar codes in an image

To read all the mail barcodes in a given image:

1. Create an `EMailBarcodeReader` object.
2. Optionally, select the relevant symbologies using the `ExpectedSymbologies` property.

   By default, Mail Bar code Reader will consider all supported symbologies.
3. Optionally, select the relevant orientations using the `ExpectedOrientations` property.

   By default, Mail Bar code Reader will test all supported orientations.
4. Call Read on the source image or ROI.

Each mail bar code detected is returned as an `EMailBarcode` object.

5. Each `EMailBarcode` objects contains the following information:
   - □ The decoded string, using the `Text` property.
   - □ The decoded string, split up in semantic parts, using the `ComponentStrings` property.
   - □ The bar code orientation, using the `Orientation` property.
   - □ The bar code position, using the `Position` property.

US Intelligent Mail bar code with highlighted position and decoded information

Advanced parameters

The advanced parameters of the EMailBarcodeReader object are:

- EnableDottedBarcodes activates the support for dotted barcodes (barcodes whose bars are printed with dots).

  By default, this property is set to false.
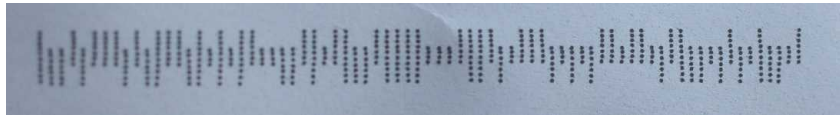
Dotted Mail Barcode

- EnableClutteredBarcodes activates the support for cluttered barcodes (barcodes in which some bars are connected).

  By default, this property is set to true.

Cluttered Mail Barcode

- ValidateChecksum activates the validation of the bar codes checksums, if present.

  By default, this property is set to false.

4.4. EasyBarCode2 - Reading Bar Codes (Improved)

EasyBarCode2 vs EasyBarCode

- As **EasyBarCode**, **EasyBarCode2** locates and reads bar codes automatically.
- Compared to **EasyBarCode**, **EasyBarCode2** has the following advantages:
  - □ A faster and more reliable detection of bar codes.
  - □ The ability to read multiple bar codes at once (with or without grid).
  - □ The ability to use an ERegion to restrict the search domain.
  - □ The ability to set a timeout.
  - □ A better support for many symbologies.
  - □ A more flexible support of checksums.
- The **EasyBarCode2** classes and methods are defined into the **EasyBarCode2** namespace.

- The following examples illustrate bar codes read by **EasyBarCode2** that cannot be read by **EasyBarCode**.
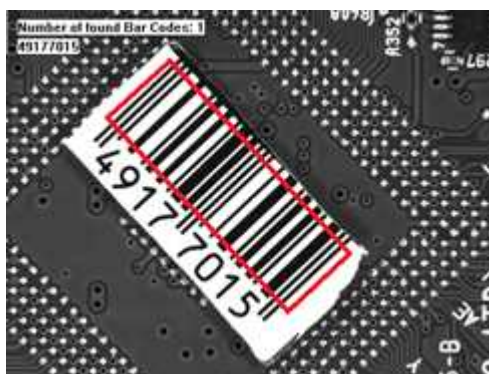
Occluded bar code (left) and a quiet zone that is too small at the top (right)

Blurry bar codes

Reading Bar Codes

Reference | Code Snippets

Bar code definition

A bar code is a 2D pattern of parallel bars and spaces of varying thickness that represents a character string.

These bars and spaces are arranged according to a convention named symbology (see below), that specifies the character set and the encoding rules.

Reading bar codes

- Use the method `EBarCode2Reader.Read` to locate the bar codes as `EBarCode` objects in an image and decode them.

- This method returns a vector of `EBarCode` objects for each bar code in the field of view, up to the maximum set with `EBarCode2Reader.MaxNumCodes`. By default, this maximum is set to 1 to optimize the process in the prevalent single-code cases.

- Use `GetDecodedString` to retrieve the contents of an `EBarCode` object.

- 📄 Refer to the following code snippet as an example: "Reading a Bar Code" on page 1

Symbologies

As stated above, a symbology defines how a bar code is encoded using its bars and spaces.

- In **EasyBarCode2**, the following symbologies are enabled by default:

◻ Code 128 (incl. Latin-1 chars)	◻ Code 39 Reduced
◻ Ean 8	◻ Code 93
◻ Ean 13	◻ Code 93 Extended
◻ GS1 128	◻ GS1 DataBar Omnidirectional
◻ Code 39	◻ GS1 DataBar Limited
◻ Code 39 Extended	◻ GS1 DataBar Expanded

- The following symbologies are also supported:

◻ Codabar	◻ Code 25 Interleaved
◻ MSI	◻ Code 32
◻ UPC A	◻ Code BCD Matrix
◻ UPC E	◻ Code CIP
◻ ADS Anker	◻ Code STK
◻ BC412	◻ IBM Delta Distance A
◻ Code 11	◻ Plessey
◻ Code 13	◻ Telepen
◻ Code 25 Datalogic	◻ Binary Code
◻ Code 25 Matrix	◻ Pharmacode (one track)
◻ Code 25 Iata	◻ RSS14
◻ Code 25 Industry	◻ RSS14 Limited
◻ Code 25 Compressed	◻ RSS14 Expanded
◻ Code 25 Inverted	

- To setup the symbologies to consider during the reading operation, use the following methods of the `EBarCode2Reader` class:

  - ◻ `EnableSymbology`

  - ◻ `EnableSymbologies`

  - ◻ `EnableAllSymbologies`

  - ◻ `EnableDefaultSymbologies`

  - ◻ `DisableAllSymbologies`

- Use `GetEnabledSymbologies` to check the enabled symbologies.

> 🗒 **NOTE**
> `EnableAllSymbologies` does not enable the `Code STK`, the `Binary Code` and the `Pharmacode` symbologies as these very permissive symbologies can generate a large amount of false positives.
> You must use `EnableSymbology` explicitly to enable the `Code STK`, the `Binary Code` and the `Pharmacode` symbologies.

- Because some symbologies are similar, a given bar code can be detected as corresponding to more than one. Use `EBarCode2.Symbologies` to retrieve all compatible symbologies, in order of decreasing confidence.

- Use `GetDecodedString` with its optional symbology parameter to retrieve the decoded string as decoded given the symbology passed as parameter. If you omit this parameter, the decoded string will be decoded using the most likely symbology.

- For the `GS1-128` and `GS1 DataBar Omnidirectional` / `Limited` / `Expanded` symbologies:
  - □ Use `GetDecodedString` to get the machine-readable code (for ex.: `]C11118011215190101`).
  - □ Use `EGs1Translator.GetHumanReadableCode` to get the human-readable version (for ex.: `(11)180112(15)190101`).

Checksum and validation

The checksum of a bar code enables the reader to validate the bar code contents.

- By default, **EasyBarCode2** checks the checksum validity when required by the symbology specifications.
  - □ The symbologies `Code39` (and variants), `Codabar`, `Code 11` and `Code 25` (and variants) define but do not enforce a checksum. We call their checksum optional. By opposition, the checksums that are always present are called mandatory. When the checksum is optional, there is no way to know if the barcode contains one from the image alone and no checksum validation is performed by default.
  - □ For the `Pharmacode` (one track) symbology, by default, no checksum is computed but additional validations are performed on the bar code vertical uniformity to rule out false positives.

- To reject bar codes if they fail mandatory or optional checksum validation or Pharmacode validation, respectively use the properties `EBarCode2Reader.ValidateMandatoryChecksum`, `EBarCode2Reader.ValidateOptionalChecksum` and `EBarCode2Reader.ValidatePharmaCode`.
  - □ Set the property to `true`, to not return bar codes failing the corresponding validation.
  - □ By default, `ValidateMandatoryChecksum` and `ValidatePharmaCode` are set to `true` and `ValidateOptionalChecksum` is set to `false`.

- Read the property `GetChecksumOK` to check the checksum status of the returned `EBarCode2` object.

- By default, the checksum characters are included into the string returned by `GetDecodedString`.
  - □ Set the optional parameter `includeChecksum` to `false` to remove these checksum characters from the returned string,

> **NOTE**
> As mentioned above, some symbologies, such as `Code 39`, have optional checksum characters. In those cases, determining their presence automatically is usually not possible. Thus, setting `includeChecksum` to `false` might crop parts of the decoded string.

Refer to the following code snippet as an example: **"Reading a Bar Code of a Specific Symbology" on page 1**

Grading Bar Codes

Reference | Code Snippets

To compute the print quality indicators as defined by the **ISO 15416** standard:

1. Set the parameter ComputeGrading to True.

2. The print quality of the bar codes is computed during the Read operation.

3. Retrieve the grades with the accessor GetGradingParameters of the class EBarCode.

▶ This will retrieve an object EBarCodeGradingParameters.

4. Retrieve the numeric grades (0 to 40) directly from the object.

5. Use the method ConvertToAlphabeticGrade to convert the numeric grades to alphabetic grades (F to A).

NOTE: At the moment, the grading is only supported for the Ean 13, Code 128 and GS1 128 symbologies. Don't hesitate to contact the support if you need the grading of other symbologies.

📄 Refer to the following code snippet as an example: "Grading a Bar Code" on page 1

Advanced Features

Retrieving the position of a bar code

- Use EBarCode2::GetPosition to retrieve the position of a detected bar code.

- This method returns an EQuadrangle object representing the detected position of the bar code in the image.

Reading using an ERegion

- Use Read(EROIBW8 field, ERegion region), the Read overload with an ERegion, to restrict the domain on which the EBarCode2Reader detects and reads the bar codes .

Reading using a grid

If the codes in the images are arranged in a regular grid-like fashion:

- Use the dedicated method `EBarCodeReader.Read` overloads that return `EBarCodeGrid` objects to improve the reliability and the speed of the reading.

- Use the methods `EBarCodeGrid.SetEnableCell`, `EBarCodeGrid.SetEnableRow`, `EBarCodeGrid.SetEnableColumn` or `EBarCodeGrid.SetEnableAll` to disable the processing of cells that you know do not contain any code.

- Use the method `GetResults` of the returned `EBarCodeGrid` object to retrieve the codes read in each cell of the defined grid.

📄 For an example, see "Reading a Grid of Bar Codes" on page 1.

Reading of bar codes arranged in a grid

Using a timeout

If you have a defined time constraint:

- Use `SetTimeout` to define a timeout for the `EBarCode2Reader` object. This timeout limits the amount of time available to the `Read` method and forces it to return early if needed.

Detecting small codes in large images

EasyBarCode2 can fail to detect small codes in large images, because it resizes the image before trying to detect the bar code.

- To solve this problem:
  - ☐ You can specify the approximate size of the smallest module in your bar code to prevent a too aggressive resizing.
  - ☐ This parameter can cause a large increase of the computation times when set to a small value.

If you have small codes in large images that you cannot read:

a. Set this parameter to 1.

b. If the reading is successful with a value of 1, try larger values, for example: 1.5, 2, 2.5, 3.

c. The ideal value should be small enough to detect your codes and large enough to not slow the processing too much.

- In the API:
  - Use the method SetMinModuleSize to specify the size of the smallest module in your bar code. The input is a floating-point value greater than or equal to 1.
  - Use SetUseMinModuleSize to enable or disable this additional treatment.

  NOTE: The value of this parameter can also be set by using the learning feature (see below).

Specifying the orientation of your codes

For most symbologies, a pattern identifies whether the codes must be read from left to right or right to left.

This is however not the case for the Code STK, Binary Code and Pharmacode symbologies.

For these symbologies, you must know whether to decode the barcode from left to right or from right to left. Use the method SetReadingOrientation to specify it.

Learning from given images to improve the detection performances

- **EasyBarCode2** can fine-tune itself to detect more images.

  For example, when

  $$max\ (barWidth,\ spaceWidth) \geq 0.08\ \times min\ (imageWidth,\ imageHeight)$$

  or (with minModuleSize set to 1) when

  $$min\ (barWidth,\ spaceWidth) \leq 0.004\ \times min\ (imageWidth,\ imageHeight)$$

  bar codes can normally not be detected and a learning is necessary.

- The learning automatically tunes 2 parameters to best fit the set of images you gave it:
  - The minModuleSize is set to the highest value that enables reading the maximum number of codes. It is set to the highest value because the higher the minModuleSize, the faster the reading.
  - The scales at which you try to find a bar code (an internal parameter not accessible to the user) are modified to read the maximum number of codes depending on the parameters of the learn method (see below).

- EBarCode2Reader.Learn has 2 Boolean parameters:
  - keepDefaultScales indicates that the scales used by default should not be disabled by the learning (True by default).
  - addAllScales indicates that all the scales that allow to detect one code should be enabled (True by default).

- Tips:
  - If you perform a learning on a few problematic images, set both parameters to True.

- If you are only interested in tuning `minModuleSize` or if you perform a learning on a representative set of problematic images, set `keepDefaultScales` to `True` and `addAllScales` to `False`.
- If you perform a learning on a representative set of all the images you want to read, set both parameters to `False`.
- The only benefit to set either of these parameters to `False` is the reading speed. In most cases, the impact should not be important, so you can leave them to `True`.

📄 Refer to the following code snippet as an example:

4.5. EasyMatrixCode - Reading Matrix Codes

EasyMatrixCode vs EasyMatrixCode2

Reference | Code Snippets

Starting with release 2.5, **Open eVision** introduces a new data matrix code reading class, named EasyMatrixCode2.

Compared to EasyMatrixCode, it offers the following benefits:
- Ability to read multiple data matrix codes in an image.
- Support for asynchronous processing.
- Improved consistency of reading and grading results.
- Improved consistency of processing time.
- Improved handling of deformed data matrix codes.

EasyMatrixCode

Specifications

Reference | Code Snippets
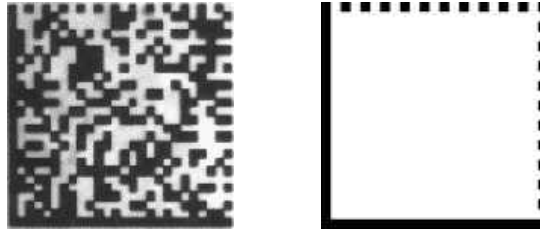
ECC 200, 26x26 cells data matrix code (left) and finder pattern (right)

In a single read operation, EasyMatrixCode locates, unscrambles, decodes, reads and grades the quality of grayscale 2D data matrix codes of any size, contrast, location and orientation (even viewed from the back on a transparent medium), providing they meet the following specifications:

- ☐ Minimum cell (= module) size: 3x3 pixels
- ☐ Maximum stretching ratio (ratio between cell width and height): 2
- ☐ Minimum quiet zone (blank zone around the matrix code) width: 3 pixels

Data Matrix Code Definition

- A data matrix code is a two-dimensional rectangular array of black and white cells which conveys a string of characters (digits, letters and special characters).
  - ☐ It is encoded to achieve maximum packing.
  - ☐ Each cell corresponds to a bit of information.
  - ☐ Additional redundant bits allow error correction for robust reading of degraded symbols.

- A data matrix code is located using the **Finder pattern**:
  - ☐ The bottom and left edges of a Data Matrix code contain only black cells.
  - ☐ The top and right edges have alternating cells.

- A data matrix code is characterized by:
  - ☐ Its logical size (number of cells).
  - ☐ Its **encoding type**: ECC 000 (odd symbol sizes, deprecated) or ECC 200 (even symbol sizes).

> **NOTE**
> The data matrix code definition is provided by ISO/IEC and approved as standard ISO/IEC 16022.

Supported Symbols

ECC other than the ECC200

NOTE: See the **ISO/IEC 16022** standard for more information.

- Error correction table

	Correctable errors %
ECC000	0
ECC050	2.8
ECC080	5.5
ECC100	12.6
ECC140	25

- Symbol table

Size	Capacity (numerical | alphanumerical | byte)														
	ECC000			ECC050			ECC080			ECC100			ECC140		
	num	alph	byte	num	alph	byte	num	alph	byte	num	alph	byte	num	alph	byte
9 × 9	6	3	1	n/a			n/a			n/a			n/a		
11 × 11	12	8	5	1	1	n/a	n/a			n/a			n/a		
13 × 13	24	16	10	10	6	4	4	3	2	1	1	n/a	n/a		
15 × 15	37	25	16	20	13	9	13	9	6	8	5	3	n/a		
17 × 17	53	35	23	32	21	14	24	16	10	16	11	7	2	1	1
19 × 19	72	48	31	46	30	20	36	24	16	25	17	11	6	4	3
21 × 21	92	61	40	61	41	27	50	33	22	36	24	15	12	8	5
23 × 23	115	76	50	78	52	34	65	43	28	47	31	20	17	11	7
25 × 25	140	93	61	97	65	42	82	54	36	60	40	26	24	16	10
27 × 27	168	112	73	118	78	51	100	67	44	73	49	32	30	20	13
29 × 29	197	131	86	140	93	61	120	80	52	88	59	38	38	25	16
31 × 31	229	153	100	164	109	72	141	94	62	104	69	45	46	30	20
33 × 33	264	176	115	190	126	83	164	109	72	121	81	53	54	36	24
35 × 35	300	200	131	217	145	95	188	125	82	140	93	61	64	42	28
37 × 37	339	226	148	246	164	108	214	143	94	159	106	69	73	49	32
39 × 39	380	253	166	277	185	121	242	161	106	180	120	78	84	56	36
41 × 41	424	282	185	310	206	135	270	180	118	201	134	88	94	63	41
43 × 43	469	313	205	344	229	150	301	201	132	224	149	98	106	70	46
45 × 45	500	345	226	380	253	166	333	222	146	248	165	108	118	78	51
47 × 47	560	378	248	418	278	183	366	244	160	273	182	119	130	87	57
49 × 49	596	413	271	457	305	200	402	268	176	300	200	131	144	96	63

ECC200

NOTE: See the **ISO/IEC 16022** standard for more information.

Square

● Symbol table

Size	Capacity			Correctable errors %	Size	Capacity			Correctable errors %
	num	alpha	byte			num	alpha	byte	
10 × 10	6	3	1	25	**44 × 44**	288	214	142	14 to 27
12 × 12	10	6	3	25	**48 × 48**	348	259	172	14 to 27
14 × 14	16	10	6	28 to 39	**52 × 52**	408	304	202	15 to 27
16 × 16	24	16	10	25 to 38	**64 × 64**	560	418	277	14 to 27
18 × 18	36	25	16	22 to 34	**72 × 72**	736	550	365	14 to 26
20 × 20	44	31	20	23 to 38	**80 × 80**	912	682	453	15 to 28
22 × 22	60	43	28	20 to 34	**88 × 88**	1152	862	573	14 to 27
24 × 24	72	52	34	20 to 35	**96 × 96**	1392	1042	693	14 to 27
26 × 26	88	64	42	19 to 35	**104 × 104**	1632	1222	813	15 to 28
32 × 32	124	91	60	18 to 34	**120 × 120**	2100	1573	1047	14 to 27
36 × 36	172	127	84	16 to 30	**132 × 132**	2608	1954	1301	14 to 26
40 × 40	228	169	112	15 to 28	**144 × 144**	3116	2335	1555	14 to 27

Rectangular

● Symbol table

Size	Capacity			Correctable errors %
	num	alpha	byte	
8 × 18	10	6	3	25
8 × 32	20	13	8	24
12 × 26	32	22	14	23 to 37
12 × 36	44	31	20	23 to 38
16 × 36	64	46	30	21 to 38
16 × 48	98	72	47	18 to 33

Workflow

Reference | Code Snippets
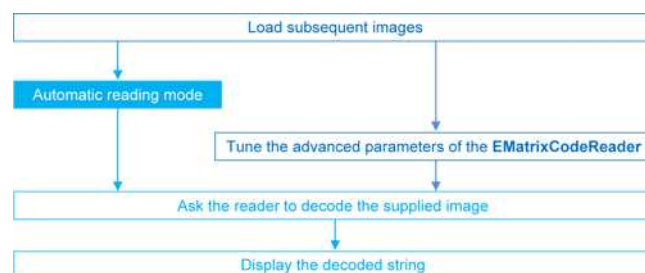
Reading a Matrix Code

Reference | Code Snippets

You can read the matrix code in an image automatically, using the Read method.

This method returns an EMatrixCode instance that contains the following information about the found data matrix code:

- □ Its decoded string,
- □ Its position in the image,
- □ Its logical size,
- □ Its encoding type,
- □ Its grading results,
- □ Methods to draw the data matrix code on the source image.

Learning a Matrix Code

Reference | Code Snippets

To search for specific features and speed up your processing, learn a Matrix code model.

Workflow
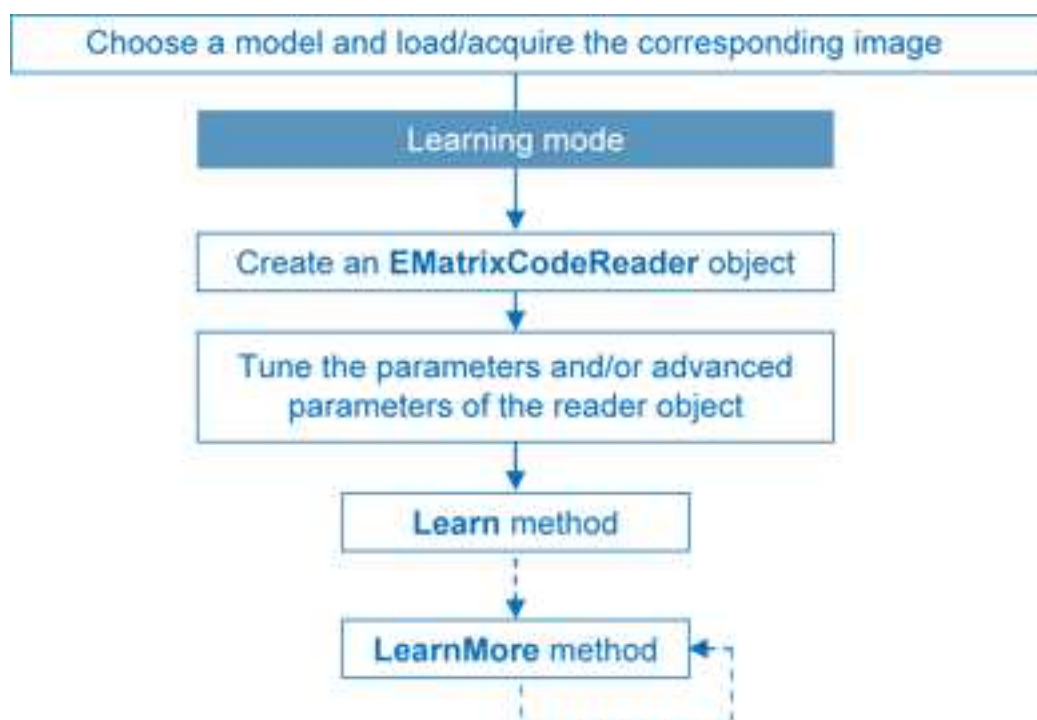
1. Load the image of the matrix code you want to learn.

2. Learn the model:

   □ Use the `Learn` method with `Contrast`, `Family`, `Flipping`, `Logical Size` parameters.

   □ If you need to learn several matrix codes, use `LearnMore` and pass additional sample images.

   □ Call `Learn` to replace `EMatrixCodeReader` parameters (calling `Learn` several times does not accumulate results, while `LearnMore` does).

3. Tune `search parameters` to be efficient and either:

   □ Read only matrix codes that match a sample matrix code,

   □ Or read only matrix codes that have the same properties (`Contrast`, `Family`, `Flipping`, `Logical Size`) as the learned one,

   □ Or disregard a search parameter of the learned matrix code `SetLearnMaskElement`, for example to read only unflipped matrix codes. Just remove the default parameters, then add new ones.

4. Ask `EMatrixCodeReader` to decode the supplied image.

5. Display the `decoded string`.

6. Save the state of the reader object using `Save`.

Restoring the state of an EMatrixCodeReader

To restore the state of an `EMatrixCodeReader` and use it to read a matrix code:

1. `Load` an image.

2. Restore the reader state from the given file using `Load`.

3. `Read` the image.

4. Display the `decoded string`.

Computing the Print Quality

Reference | Code Snippets

To compute the print quality indicators as defined by BC11, ISO 15415, ISO/IEC TR 29158 (formerly known as AIM DPM-1-2006) and SEMI T10-0701 standards, retrieve the grades with the `GetIso15415GradingParameters`, `GetIso29158GradingParameters` and `GetSemiT10GradingParameters` accessors of the `EMatrixCode` class.

> **NOTE**
> The print quality of the matrix codes is computed during the `Read` operation, only if the `ComputeGrading` parameter is set to `true`.

Using GS1 Data Matrix Codes

Reference | Code Snippets

EasyMatrixCode is able to find and decode GS1-compliant data matrix codes.

The GS1 standard adds semantic identifiers to the contents of a data matrix code. These identifiers are interpreted in an easy and consistent way.

The structure of GS1-compliant content is as follows:

$$]d2 \, [GS1] \, \{Id1\} \, \{Value1\} \, [GS1] \, \{Id2\} \, \{Value2\} \cdots$$

where:

- "]d2" is the string identifying a GS1-compliant stream,
- [GS1] is the GS1 escape character (0x1d),
- {Id} is an application identifier,
- {Value} is the value associated with that identifier.

Example

The string:

$$]d2 \, [GS1] \, 11 \, 180112 \, [GS1] \, 15 \, 190101$$

is interpreted as follows:

- It contains two GS1 parts: 11180112 and 15190101.
- The first (11180112) is composed of the identifier 11 and the value 180112, meaning that the product has a production date (the meaning of identifier 11) of January 12th, 2018.
- The second (15190101) is composed of the identifier 15 and the value 190101, meaning that the product has a best before date (the meaning of identifier 15) of January 1st, 2019.

> ✓ TIP
> For more information, see https://www.gs1.org/

EasyMatrixCode2

Specifications

Reference | Code Snippets
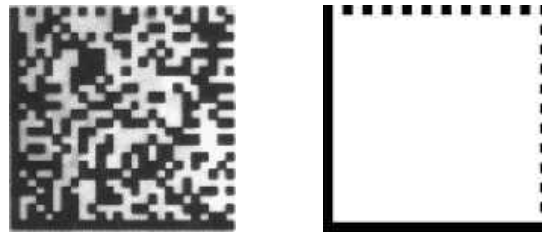
ECC 200, 26x26 cells data matrix code (left) and finder pattern (right)

In a single read operation, EasyMatrixCode2 locates, unscrambles, decodes, reads and grades the quality of grayscale 2D data matrix codes of any size, contrast, location and orientation (even viewed from the back on a transparent medium), providing they meet the following specifications:

- ☐ Minimum cell (= module) size: 3x3 pixels
- ☐ Minimum quiet zone (blank zone around the matrix code) width: 1 pixel

All the functionality of EasyMatrixCode2 is available for testing in **Open eVision Studio**, except for the StopProcess method (for asynchronous processing).

> **NOTE**
> The relevant classes of the EasyMatrixCode2 library are stored in the name space "EasyMatrixCode2".

Data Matrix Code Definition

- A data matrix code is a two-dimensional rectangular array of black and white cells which conveys a string of characters (digits, letters and special characters).
  - ☐ It is encoded to achieve maximum packing.
  - ☐ Each cell corresponds to a bit of information.
  - ☐ Additional redundant bits allow error correction for robust reading of degraded symbols.

- A data matrix code is located using the **Finder pattern**:
  - ☐ The bottom and left edges of a Data Matrix code contain only black cells.
  - ☐ The top and right edges have alternating cells.

- A data matrix code is characterized by:
  - □ Its logical size (number of cells).
  - □ Its **encoding type**: ECC 000 (odd symbol sizes, deprecated) or ECC 200 (even symbol sizes).

> **NOTE**
> The data matrix code definition is provided by ISO/IEC and approved as standard ISO/IEC 16022.

Supported Symbols

ECC other than the ECC200

NOTE: See the **ISO/IEC 16022** standard for more information.

- Error correction table

	Correctable errors %
ECC000	0
ECC050	2.8
ECC080	5.5
ECC100	12.6
ECC140	25

- Symbol table

Size	Capacity (numerical | alphanumerical | byte)														
	ECC000			ECC050			ECC080			ECC100			ECC140		
	num	alph	byte	num	alph	byte	num	alph	byte	num	alph	byte	num	alph	byte
9 × 9	6	3	1	n/a			n/a			n/a			n/a		
11 × 11	12	8	5	1	1	n/a	n/a			n/a			n/a		
13 × 13	24	16	10	10	6	4	4	3	2	1	1	n/a	n/a		
15 × 15	37	25	16	20	13	9	13	9	6	8	5	3	n/a		
17 × 17	53	35	23	32	21	14	24	16	10	16	11	7	2	1	1
19 × 19	72	48	31	46	30	20	36	24	16	25	17	11	6	4	3
21 × 21	92	61	40	61	41	27	50	33	22	36	24	15	12	8	5
23 × 23	115	76	50	78	52	34	65	43	28	47	31	20	17	11	7
25 × 25	140	93	61	97	65	42	82	54	36	60	40	26	24	16	10
27 × 27	168	112	73	118	78	51	100	67	44	73	49	32	30	20	13
29 × 29	197	131	86	140	93	61	120	80	52	88	59	38	38	25	16
31 × 31	229	153	100	164	109	72	141	94	62	104	69	45	46	30	20
33 × 33	264	176	115	190	126	83	164	109	72	121	81	53	54	36	24
35 × 35	300	200	131	217	145	95	188	125	82	140	93	61	64	42	28
37 × 37	339	226	148	246	164	108	214	143	94	159	106	69	73	49	32
39 × 39	380	253	166	277	185	121	242	161	106	180	120	78	84	56	36
41 × 41	424	282	185	310	206	135	270	180	118	201	134	88	94	63	41
43 × 43	469	313	205	344	229	150	301	201	132	224	149	98	106	70	46
45 × 45	500	345	226	380	253	166	333	222	146	248	165	108	118	78	51
47 × 47	560	378	248	418	278	183	366	244	160	273	182	119	130	87	57
49 × 49	596	413	271	457	305	200	402	268	176	300	200	131	144	96	63

ECC200

NOTE: See the **ISO/IEC 16022** standard for more information.

Square

- Symbol table

Size	Capacity			Correctable errors %	Size	Capacity			Correctable errors %
	num	alpha	byte			num	alpha	byte	
10 × 10	6	3	1	25	44 × 44	288	214	142	14 to 27
12 × 12	10	6	3	25	48 × 48	348	259	172	14 to 27
14 × 14	16	10	6	28 to 39	52 × 52	408	304	202	15 to 27
16 × 16	24	16	10	25 to 38	64 × 64	560	418	277	14 to 27
18 × 18	36	25	16	22 to 34	72 × 72	736	550	365	14 to 26
20 × 20	44	31	20	23 to 38	80 × 80	912	682	453	15 to 28
22 × 22	60	43	28	20 to 34	88 × 88	1152	862	573	14 to 27
24 × 24	72	52	34	20 to 35	96 × 96	1392	1042	693	14 to 27
26 × 26	88	64	42	19 to 35	104 × 104	1632	1222	813	15 to 28
32 × 32	124	91	60	18 to 34	120 × 120	2100	1573	1047	14 to 27
36 × 36	172	127	84	16 to 30	132 × 132	2608	1954	1301	14 to 26
40 × 40	228	169	112	15 to 28	144 × 144	3116	2335	1555	14 to 27

Rectangular

- Symbol table

Size	Capacity			Correctable errors %
	num	alpha	byte	
8 × 18	10	6	3	25
8 × 32	20	13	8	24
12 × 26	32	22	14	23 to 37
12 × 36	44	31	20	23 to 38
16 × 36	64	46	30	21 to 38
16 × 48	98	72	47	18 to 33

Extended rectangular data matrix DMRE

NOTE: See the **ISO/IEC 21471** standard for more information.

- Symbol table

Size	Capacity			Correctable errors %	Size	Capacity			Correctable errors %
	num	alpha	byte			num	alpha	byte	
8 x 48	36	25	16	21 to 36	**20 x 36**	88	64	42	19 to 35
8 x 64	48	34	22	21 to 36	**20 x 44**	112	82	54	19 to 34
8 x 80	64	46	30	20 to 35	**20 x 64**	186	124	82	17 to 31
8 x 96	76	55	36	21 to 38	**22 x 48**	144	106	70	17 to 32
8 x 120	98	72	47	20 to 36	**24 x 48**	160	118	78	17 to 31
8 x 144	126	93	61	18 to 33	**24 x 64**	216	160	106	15 to 28
12 x 64	86	63	41	19 to 34	**26 x 40**	140	103	68	18 to 32
12 x 88	128	94	62	18 to 33	**26 x 48**	180	133	88	16 to 30
16 x 64	124	91	60	18 to 34	**26 x 64**	236	175	116	15 to 28

Workflow

Reference | Code Snippets
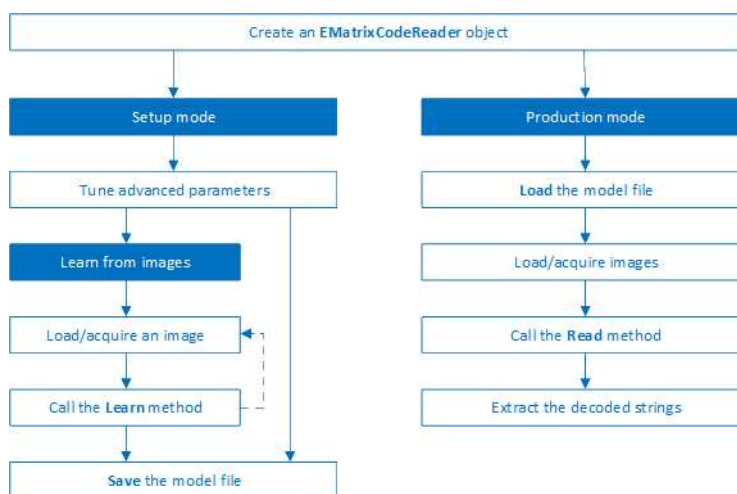
1. Load the image.
2. Read the data matrix codes in the image using `EMatrixCodeReader.Read()`.
3. Loop on found data matrix codes.
4. Display the decoded text.

Reading a Matrix Code

Reference | Code Snippets | dedicated code snippet: "Reading Data Matrix Codes" on page 182

You can read the matrix code in an image automatically as follows:

a. Create an `EMatrixCodeReader` object.

b. Call the `Read` method to return a vector containing the detected and decoded matrix codes in the image.

The `Read` method provides the following overloads:

□ One overload that takes an `ERegion` object as an additional parameter to specify more precisely the search area.

□ One overload to specify a search grid when your matrix codes are placed in a regular fashion.

The `EMatrixCode2.EMatrixCode` instances contain the following information for each found data matrix code:

□ Its decoded string,

□ Its position in the image,

□ Its logical size,

□ Its encoding type,

□ Its grading results,

□ Methods to draw the data matrix code on the source image.

Learning a Matrix Code

Reference | Code Snippets | dedicated code snippet: "Learning a Data Matrix Code" on page 183

To improve the processing times and to read small codes, learn a matrix code model from representative images as follows:

1. Load the image of the matrix code you want to learn from.

2. Call the `Learn` method to learn from the image.

3. Repeat with additional images if necessary.

4. Save the `EMatrixCodeReader` state to the disk with the `Save` method.

By default, the `Learn` method explores the full range of the `EMatrixCodeReader` internal parameters to search for data matrices in any type of context.

● If the learning process finds valid data matrices, the internal parameters that were necessary to find them are enabled, set and ordered in a way meant to improve both reading speed and capability in similar contexts.

● The user-defined advanced parameters (`MaxNumCodes`, `Timeout`, `ReadMode` and `ComputeGrading`) are not affected by the `Learn` method.

● After setting `MatrixCodeDimensionsRange`, a call to `Read` or `Learn` restricts the processing around that range. This can dramatically improve the speed of the process.

  NOTE: However, if you set the property after the learning, the learning is reset and must be redone.

- If the Learn method is not able to detect any code in the image, it throws an exception.

  NOTE: The internal processing structure is not affected in this situation.

- If the Learn method has been called, all the subsequent images read or learned must have the same dimensions as the one used in the first learn call, otherwise an exception is thrown.

Restoring the state of an EMatrixCodeReader

- To restore a previously saved EMatrixCodeReader state, call the Load method.

- To restore the default state of an EMatrixCodeReader instance, call the ResetLearning method.

Computing the Print Quality

Reference | Code Snippets | dedicated code snippet: "Grading a Data Matrix Code" on page 184

To compute the print quality indicators as defined by BC11, ISO 15415, ISO/IEC TR 29158 (formerly known as AIM DPM-1-2006) and SEMI T10-0701 standards, retrieve the grades with the GetIso15415GradingParameters, GetIso29158GradingParameters and GetSemiT10GradingParameters accessors of the EMatrixCode2.EMatrixCode class.

> **NOTE**
> The print quality of the matrix codes is computed during the Read operation, only if the ComputeGrading parameter is set to true.

Using GS1 Data Matrix Codes

Reference | Code Snippets

EasyMatrixCode2 is able to find and decode GS1-compliant data matrix codes.

The GS1 standard adds semantic identifiers to the contents of a data matrix code. These identifiers are interpreted in an easy and consistent way.

The structure of GS1-compliant content is as follows:

$$]d2\,[GS1]\,\{Id1\}\,\{Value1\}\,[GS1?]\,\{Id2\}\,\{Value2\}\cdots$$

where:
- "]d2" is the string identifying a GS1-compliant stream,
- [GS1] is the GS1 escape character (0x1d),
- [GS1?] means that the [GS1] escape character is present there if the previous application identifier has a variable size value,
- {Id} is an application identifier,
- {Value} is the value associated with that identifier.

Example

The string:

$$]d2\,[GS1]\,10\,GR\text{-}1\text{-}GNU\,[GS1]\,111180112151901012112345$$

is interpreted as follows:

]d2	[GS1]	10	GR-1-GNU	[GS1]	11	180112	15	190101	21	12345
		Id1	Value1		Id2	Value2	Id3	Value3	Id4	Value4

- The first Id is 10, this means the first value is the batch / lot number whose size is at most 20 chars.
    - □ Thus, the product has a batch / lot number of GR-1-GNU.
    - □ As batch / lot numbers are of a variable size, a separator is required after GR-1-GNU.
- The second Id is 11, this means the second value is the production date which always consists of 6 digits.
    - □ The product has a production date of 180112 (12 of January 2018).
    - □ As production dates have a fixed size, no group separator is required after it.
- The third Id is 15, this means the third value is the "best before date" which always consists of 6 digits.
    - □ The product has a "best before date" of 190101 (1st of January 2019).
    - □ As "best before dates" have a fixed size, no group separator is required after it.
- The fourth Id is 21, this means the fourth value is the serial number whose size is at most 20 chars.
    - □ The product has a serial number of 12345.
    - □ Altough serial numbers are of variable size, 12345 is the last part of the decoded string, so there is no group separator after it.

📄 For more information, see https://www.gs1.org/

- Use `EGs1Translator::GetHumanReadableCode` to get the human-readable version:

$$(10)\,GR\text{-}1\text{-}GNU\,(11)\,180112\,(15)\,190101\,(21)\,12345$$

Asynchronous Processing

Reference | Code Snippets

EasyMatrixCode2 supports asynchronous processing. This means that you can launch multiple processing threads in parallel, each reading the matrix codes in its own image.

From the main thread, to manually stop the Read method in any of these processing threads at any time, use the StopProcess method.

When you manually stop the Read method:
- □ The search for matrix codes stops immediately, whether it has found matrix codes in the image or not.
- □ To retrieve all matrix codes found before the manual stop, use the GetReadResults accessor.

Reading Using a Grid

If the codes in the images are arranged in a regular grid-like fashion:

- Use the dedicated method `EMatrixCodeReader.Read` overloads that return `EMatrixCodeGrid` objects to improve the reliability and the speed of the reading.

- Use the methods `EMatrixCodeGrid.SetEnableCell`, `EMatrixCodeGrid.SetEnableRow`, `EMatrixCodeGrid.SetEnableColumn` or `EMatrixCodeGrid.SetEnableAll` to disable the processing of cells that you know do not contain any code.

- Use the method `GetResults` of the returned `EMatrixCodeGrid` object to retrieve the codes read in each cell of the defined grid.

📄 For an example, see

Reading of data matrices arranged in a grid

Returning Unreadable Codes

Some codes may have their data part so damaged that they cannot be decoded.

- However, you can set EnableReturnUnreliableCodes to True to return them all the same.
  - ☐ These codes are placed at the end of the return vector.
  - ☐ Calling EMatrixCode::GetIsReliable on them returns False.
- Use the usual methods to retrieve the estimated logical size as well as the position of the unreliable codes detected.

Top: input images
Bottom: read results with unreliable detections depicted in red

- Since the data part is no longer a rejection criterion, the likelihood that a false positive unreliable code is returned increases.
  - ☐ To filter out these false positive detections, use the value returned by EMatrixCode::GetReliabilityScore as a filtering threshold. This score translates as a confidence that the detected object is an actual data matrix code.

Advanced Parameters

Reference | Code Snippets

Tune the following parameters to optimize the performance of **EasyMatrixCode2**.

- The MaxNumCodes parameter:
  - ☐ Tells the EMatrixCode2Reader the number of codes that can be in the image.
  - ☐ Affects the computational time of the Read method.

- The `Timeout` parameter:
  - □ Limits the amount of time that the `Read` and `Learn` methods may take to process a single image.
  - □ Is defined in microseconds.
  - □ Is set, by default, to a value that exceeds one hour.

- The `ReadMode` parameter affects the behavior of the `Read` method:
  - □ The setting `EReadMode_Speed` results in the shortest processing times and the `Read` method stops as soon as one of the following is true:
    - The method has found `MaxNumCodes` codes.
    - The method reaches the `Timeout` time limit.
    - The `Read` process is completely finished.
  - □ The setting `EReadMode_Quality` results in the best grading results and the `Read` method keeps trying to improve its detection until one of the following is true:
    - The method reaches the `Timeout` time limit.
    - The `Read` process is completely finished.

- The `ComputeGrading` parameter:
  - □ Determines if the `Read` method computes the grading properties of the `EMatrixCode2.EMatrixCode` object.
  - □ Is set to `False` by default.

- The `MatrixCodeDimensionsRange` parameter:
  - □ Sets the range of valid lengths in pixels that the sides of the data matrices must satisfy to be detected.
  - □ You can also use this parameter to extend the reading capability to codes relatively small compared to the input image.

After the tuning:
  - □ Use the `Save` method to store the state of the `EMatrixCode2Reader` on the disk.
  - □ Use the `Load` method, at any time, to restore the saved state.

> ✅ **TIP**
> The `Save` and `Load` methods also store the effects of `Learning`.

4.6. EasyQRCode - Reading QR Codes
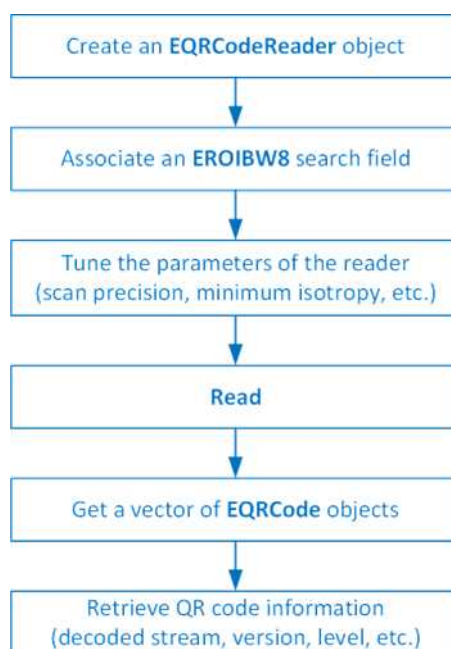
Workflow

EasyQRCode

EasyQRCode detects QR (Quick Response) codes in an image, decodes them, and returns their data.

Error detection and correction algorithms ensure that poorly-printed or distorted QR codes can still be read correctly.

Workflow

QR Codes Specifications

QR code definition

A QR code is a square array of dark and light dots. One dot (or "*module*") represents one bit of information.

QR codes contain various types of data and can be different models, versions, and levels. They always contain a message, metadata about alignment, size, format, and error correction bits. They comply with the international standard ISO/IEC 18004 (1, 2 and 2005).

QR code structure

The QR code symbol consists of an *encoding region*, containing data and error correction codewords, and of *function patterns*, containing symbol metadata and position data.

A QR code must be structured with the following elements:
- *Quiet zone*: blank margin around the QR code
- *Finder patterns*: recognizable zones identifying a QR code
- *Extension patterns*: markers for the alignment of the QR code (model 1)
- *Alignment patterns*: markers for the alignment of the QR code (models 2 and 2005)
- *Timing Patterns*: data giving the module size (in pixels)
- *Format information*: zones providing the QR code level
- *Version information*: data giving the QR code size, for instance 25 x 25 modules (models 2 and 2005)
- *Data contents and error correction codewords*: the primary information carried by the symbol, with additional information for error correction

Variants of this structure exist, according to the model, format, or version of the QR code. For instance, model 1 QR codes do not feature alignment patterns but extension patterns. Micro QR codes include only one finder pattern, and no alignment pattern.
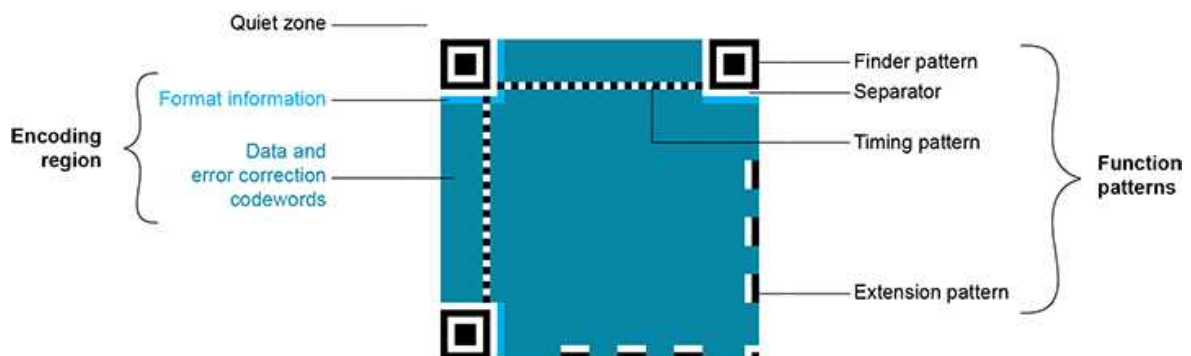
Structure of a model 1 QR code symbol
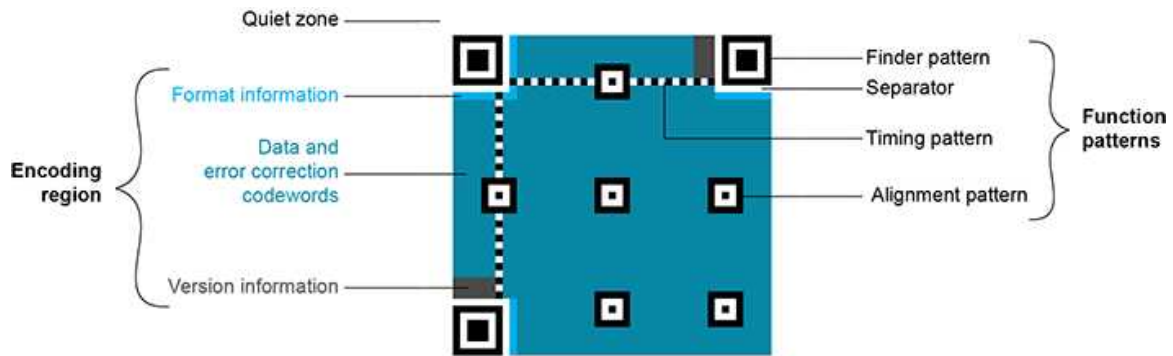
Structure of a QR code 2005 symbol
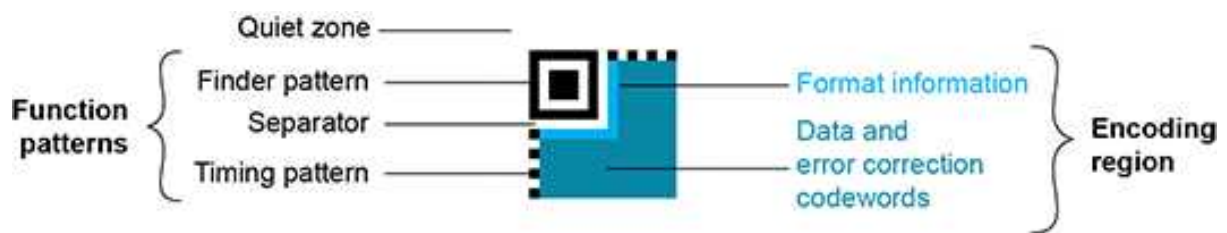
Structure of a Micro QR code symbol

QR code subtypes

A QR code can be one of the following subtypes:
- *Basic*: the default subtype.
- *ECI* (Extended Channel Interpretation): the ECI subtype provides a consistent method to embed interpretation information of data in the QR code. The ECI protocol is defined in the AIM Inc. International Technical Specification. (ECI is not available for Micro QR code symbols.)
- *GS1*: the data contained in the QR code are formatted in accordance with the GS1 General Specification.
- *AIM*: the data contained in the QR code are formatted in accordance with a specific industry application previously agreed with AIM Inc. The application indicator value is embedded in the QR code data.

Data types

The QR code data can be any mix of these types:
- *Numeric data* (0-9)
- *Alphanumeric data* (0-9, A-Z, /,$, %...)
- *Byte data* (possibly ECI-encoded)
- *Kanji characters*

95

Byte data interpretation

In a QR code, the byte data can represent any information. Their interpretation depends on the subtype of the QR code:

- Basic subtype:
  - If some byte data are present in the QR code, you need to know how to interpret them.
  - Use the `EByteInterpretationMode` enum to select the corresponding byte interpretation mode (see the retrieving decoded data section in for more details).

- ECI-encoded byte data:
  - The ECI subtype provides an ECI table indicator.
  - This indicator defines the character set to use to interpret the byte data.
  - **EasyQRCode** currently supports the UTF8 conversion table (ECI table indicator 26).

Models (Standards)

- *Model 1*: original QR code international standard, with versions ranging from 1 to 14. Note that the "version" of a QR code is the symbol size (in number of modules). It does not relate to the version of the standard, which is called the "model".
- *Model 2*: improvement of model 1. It provides versions from 1 to 40. It defines alignment patterns to improve reading of distorted QR codes, or QR codes printed on curved surfaces.
- *Model 2005*: improvement of model 2, including white-on-black QR codes, and mirror symbol orientation.
- *Micro QR codes*: smaller QR codes, from version *M1* to version *M4*. They have been introduced to save printing space.

Versions (Symbol Size)

- *QR codes*: from version *1* (21 x 21 modules) to version *40* (177 x 177 modules), with an increment of +4 x +4 modules (version 2: 25 x 25 modules, version 3: 29 x 29 modules, ..., version 39: 173 x 173 modules).
- *Micro QR codes*: version *M1* (11 x 11 modules), version *M2* (13 x 13 modules), version *M3* (15 x 15 modules), version *M4* (17 x 17 modules).

Examples of QR codes
From left to right:
Micro QR code, version M3, 15 x 15 modules,
Model 2 QR code, version 4, 33 x 33 modules, 67-114 characters,
Model 2 QR code, version 40, 177 x 177 modules, 1852-4296 characters

Levels (Error Correction)

QR codes contain error correction data. The standard offers the following levels of error correction:

- *L*: (low) about 7% of codewords can be restored
- *M*: (medium) 15%
- *Q*: (quality) 25%
- *H*: (high) 30% (not available for Micro QR codes)

For Micro QR code symbols, the available error correction levels depend on the version:

- M1 has only error detection
- M2 and M3 support L and M levels
- M4 supports L, M and Q levels

QR code geometry

When the QR code reader finds an array of dots that could match a QR code, it returns the "geometry" of this QR code candidate.

A `QR code geometry` is a set of points:

- It contains the coordinates of the corners of the QR code quadrangle (bottom left, top left, top right, bottom right).
- It contains the coordinates of the `finder pattern centers` (bottom left, top left, top right).
- For a Micro QR code symbol, the coordinates for a single `finder pattern center` (link) are returned.

EasyQRCode uses a float coordinate system and the origin (0.0, 0.0) is the top left corner of the top left pixel of the image.

QR code geometry

Read a QR code

Reading a QR code returns information about every `QR codes` found in the given search field (see "Reading QR Codes" on page 98).

Reading QR Codes

Read a QR code

1. Set a `search field` on an `EROIBW8` image.
   - ☐ To restrict the field further, use an optional ERegion.
   - ☐ To read multiple QR codes placed in a regular fashion, use a specific overload of `Read` to use a grid.

2. If needed, tune the parameters to restrict the number of operations to process.

3. The QR code reader scans the image and searches for 3 finder patterns that could match a QR code, with the following requirements:
   - ☐ Minimum quiet zone (blank zone around the QR code) width: 3 pixels.
   - ☐ Minimum module size: 3 x 3 pixels.
   - ☐ Minimum isotropy: 0.5.
   - ☐ Maximum corner deformation: 15° (corner angles can range from 75° to 105°).

4. The QR code reader uses the `gravity center` of the `QR code geometries` to sort the QR code candidates in line then columns order starting from the top left corner of the image.

5. The QR code reader decodes the QR candidates and returns the `QR code`: `model`, `version`, `level`, `geometry` and the `decoded data` as described below.

6. The reader can report the amount of `unused error correction`.
   - ☐ Close to 1, very few errors were corrected when decoding the data. The decoding is highly reliable, and the QR code is of good quality.
   - ☐ Close to 0, many errors were corrected when decoding the data. The decoding is reliable, but the QR code quality is poor.
   - ☐ -1, error correction failed. Decoding was not performed.

Tune the search parameters

- `Scan precision`: You can change the scan precision to scan the search field with:
  - ☐ A fine precision (recommended for small QR codes)
  - ☐ A coarse precision (recommended for medium to large QR codes)

- `Minimum score`: The QR code reader searches for this QR code finder pattern:

  - ☐ A perfect match returns a pattern finder score of 1.
  - ☐ Less accurate matches return lower scores.
  - ☐ The minimum score allowed by default is 0.65 - you can tune this.

- **Minimum isotropy**: The isotropy of a QR code represents its rectangular deformation.
  - □ Perfectly square QR codes have an isotropy of 1 (short side divided by long side, whether the rectangle is vertical or horizontal).
  - □ **EasyQRCode** can detect rectangle QR codes with an isotropy down to 0.5.
  - □ The default `minimum isotropy` is 0.8, it can be tuned from 0 to 1.

Square and rectangular QR codes (isotropy = 1, 0.5, and 0.5 from left to right)

- **Model** and **Version**: By default, the QR code reader searches for QR codes of model 1 and 2, and all versions.
  - □ You can shorten the process by specifying the QR code `model(s)` and a range of versions (from 1 (`minimum`) to 40 (`maximum`)) to search for.
  - □ By default, the QR code reader does not search for Micro QR code symbols.

Retrieve the decoded data

Retrieving methods

To retrieve the decoded data, you can (in growing complexity order):

1. Use the `GetDecodedString` method of an `EQRCode` object.
   - □ This method returns an UTF-8 formatted string that contains the concatenated data of the QR code.
   - □ It can take an `EByteInterpretationMode` as argument.
   - □ For the GS1 QR codes:
     - Use `GetDecodedString` to get the machine-readable code (for ex.: `]Q31118011215190101`).
     - Use `EGs1Translator::GetHumanReadableCode` to get the human-readable version (for ex.: `(11)180112(15)190101`).

2. Use the `GetDecodedString` method of the `EQRCodeDecodedStreamPart` objects.
   - □ This method is called on a part and returns an UTF-8 formatted string that contains the data of this part.
   - □ It can take an `EByteInterpretationMode` as argument.
   - □ Concatenate the decoded string of each part.

3. Use the `GetDecodedData` method of the `EQRCodeDecodedStreamPart` objects.
   - □ This method is called on a part and returns a vector of bytes that contains the data of this part.
   - □ Interpret the data according to the `coding mode` of the QR code and the `encoding` of each part.
   - □ Concatenate the interpreted data of each part.

Interpreting the encoded data

The QR code data can be encoded in either alphanumeric, numeric or byte modes. If a QR code contains bytes, the interpretation mode of these bytes can be embedded in the QR code through the ECI protocol or you must specify or know it.

Use the dedicated EByteInterpretationMode for this purpose:

- EByteInterpretationMode_Hexadecimal
  - □ Converts all bytes to their hexadecimal values (2 characters per byte).
  - □ The escape character 0xEFBFBD surrounds the converted byte parts.
  - □ This mode overrides the ECI table indicator if it is present.

- EByteInterpretationMode_UTF8
  - □ Converts all bytes to UTF-8 if possible.
  - □ The GetDecodedString method throws an EException if the data are not UTF-8 compatible.

- EByteInterpretationMode_Auto
  - □ Converts all bytes in the best possible way following the ECI protocol.

The decoded string returns the concatenated data of the QR code in UTF-8 format:
  - □ If bytes are present in the QR code data without ECI, specify the byte interpretation mode when you call the GetDecodedString method.
  - □ If bytes are present in the QR code data with ECI encoding, use the corresponding byte interpretation table (currently, only table ECI 26: UTF-8 is available).
  - □ The hexadecimal byte interpretation mode does not throw an exception and returns all bytes parts present in the data in their hexadecimal form (2 characters per byte) surrounded by the 0xEFBFBD escape character.
  - □ See the code snippet "Retrieving Information of a QR Code" on page 187.

The decoded stream class consists of:
  - □ A coding mode (basic, ECI, FNC1/GS1 or FNC1/AIM).
  - □ An application indicator (if the coding mode is FNC1/AIM, otherwise 0).

The decoded data:
  - □ Is accessible from each part of the decoded stream.
  - □ Is interpreted according to its encoding (numeric, alphanumeric, byte or Kanji) and the ECI table indicator (if the coding mode is ECI, otherwise -1).
  - □ Can be the raw bit stream (the bit data after unmasking and error correction, but before decoding as a vector of bytes).
  - □ Can be the corresponding decoded string (specify a byte interpretation mode if the encoding is byte without ECI coding mode or if the ECI table is not supported).
  - □ See also the code snippet "Retrieving the Decoded Data (Advanced)" on page 189.

Computing the print quality

- The print quality indicators as defined by **ISO 15415** and **ISO/IEC TR 29158** (formerly known as **AIM DPM-1-2006**) are computed during the Read operation, but only if EQRCodeReader::ComputeGrading is set to TRUE.
  - □ The print quality is not yet supported for Micro QR code models.

- Use the EQRCode::GetIso15415GradingParameters and EQRCode::GetIso29158GradingParameters methods to retrieve the grades.

- Using the grading:
  - ☐ For more accurate results, it requires modules to be at least 10 pixels in width.
  - ☐ It requires a 1-module quiet zone for grading.
  - ☐ It evaluates the `Fixed Pattern Damage` with a 4-module quiet zone around the finder patterns. If this condition is not met, the `Fixed Pattern Parameter Grade` is returned as -1. This result affects the overall symbol grade.
  - ☐ The `Version Additional Parameter` is returned as -1 when it is not applicable. In this case, this result is ignored in the overall symbol grade.
  - ☐ The implementation follows closely the standard but the grades also depend on the decoding algorithm. So the results may slightly differ according to the **Open eVision** version.
- The print quality computation is not yet available for Micro QR codes.

Multithreading

The `Read` method supports multithreading.

- Multithreading splits the load between the detection methods (such as `AdaptiveThreshold` and `Gradient`) and decodes multiple candidates in parallel. This is useful when there are several codes in the image.

Reading Using a Grid

If the codes in the images are arranged in a regular grid-like fashion:

- Use the dedicated method `EQRCodeReader.Read` overloads that return `EQRCodeGrid` objects to improve the reliability and the speed of the reading.
- Use the methods `EQRCodeGrid.SetEnableCell`, `EQRCodeGrid.SetEnableRow`, `EQRCodeGrid.SetEnableColumn` or `EQRCodeGrid.SetEnableAll` to disable the processing of cells that you know do not contain any code.
- Use the method `GetResults` of the returned `EQRCodeGrid` object to retrieve the codes read in each cell of the defined grid.

📄 For an example, see .

Reading of QR codes arranged in a grid

4.7. EasyOCR - Reading Texts

Workflow

Reference | Code Snippets

EasyOCR

EasyOCR optical character recognition library reads short texts (such as serial numbers, part numbers and dates).

It uses font files (pre-defined OCR-A, OCR-B and Semi standard fonts, or other learned fonts) with a template matching algorithm that can recognize even badly printed, broken or connected characters of any size.

There are 4 steps to recognizing characters:

| 1. Raw image | 2. Object segmentation | 3. Character isolation | 4. Character recognition |
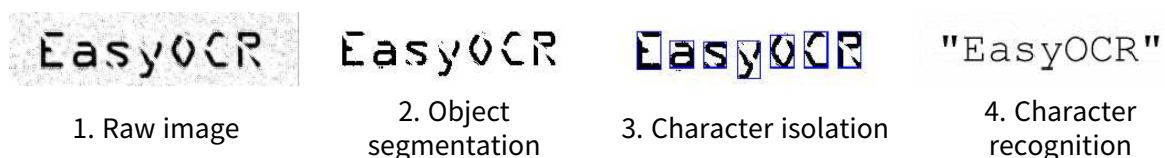
Workflow
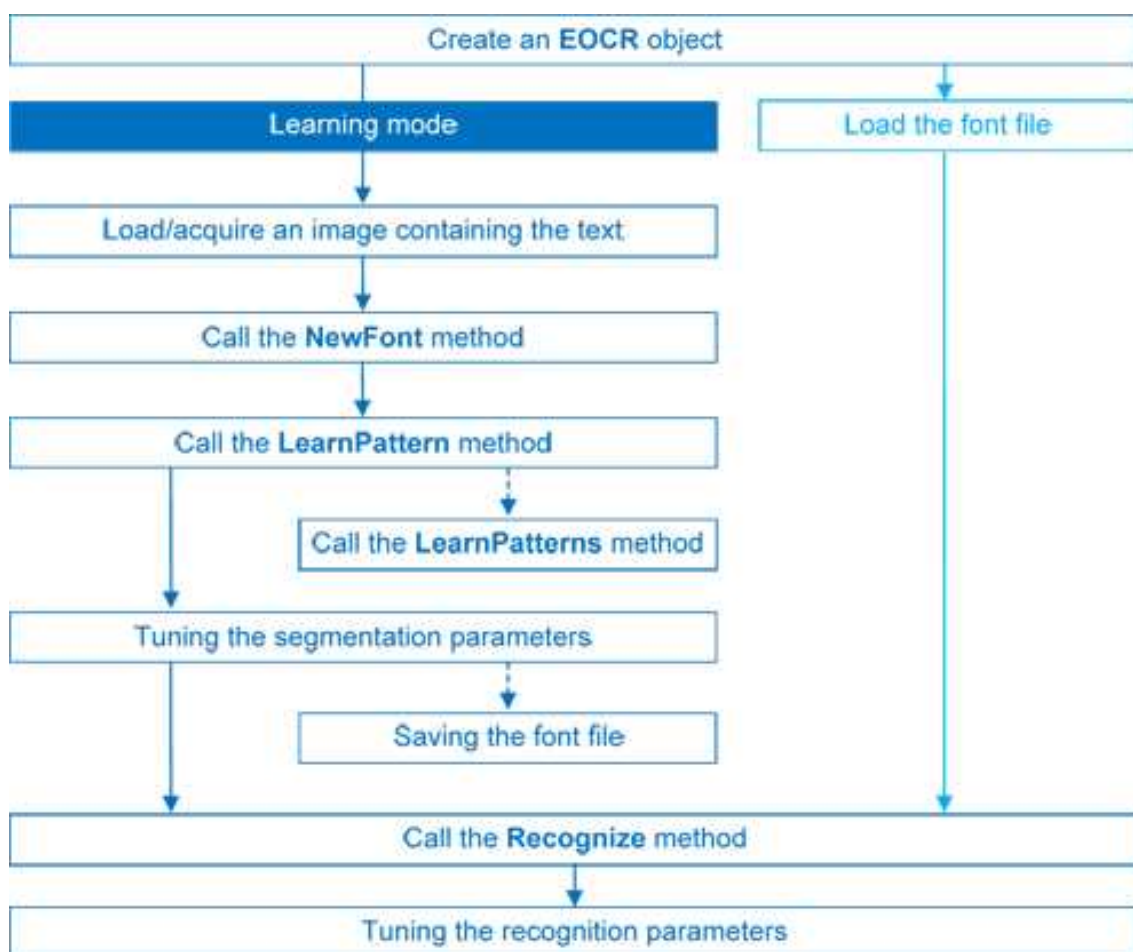
Learning Process

You can learn characters to create a font file if required.
Characters are presented one by one to EasyOCR which analyzes them and builds a database of characters called a font. Each character has a numeric code (usually its ASCII code) and belongs to a character class (which may be used in the recognition process).

Font files are created as follows:

1. NewFont clears the current font.

2. LearnPattern or LearnPatterns adds the patterns from the source image to the font. Patterns are ordered by their index value, as assigned by the FindAllChars process. The patterns in a font are stored as a small array of pixels, by default 5 pixels wide and 9 pixels high. This size can be changed before learning, using parameters PatternWidthand PatternHeight.

3. RemovePattern removes unwanted patterns (optional).

4. Save writes the contents of the font to a disk file with parameter values: NoiseArea, MaxCharWidth, MaxCharHeight, MinCharWidth, MinCharHeight, CharSpacing, TextColor.

Segmenting

Segmenting

1. **EasyOCR** analyses the blobs to locate the characters and their bounding box, using one of two segmentation modes:
   - □ **keep objects** mode: one blob corresponds to one character.
   - □ **repaste objects** mode: the blobs are grouped into characters of a nominal size. This is useful when characters are broken or made up of several parts. When a blob is too large to be considered a single character, it can be split automatically using CutLargeChars.
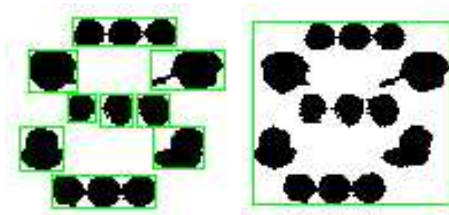

Character segmentation by blob grouping

2. Filters remove very large and very small unwanted features.

3. **EasyOCR** processes the character image to normalize the size into a bounding box, extracts relevant features, and stores them in the font file. The patterns in a font are stored as arrays of pixels defined by PatternWidth and PatternHeight (by default 5 pixels wide and 9 pixels high).

Segmentation parameters

Segmentation parameters must be the same during learning and recognition. Good segmentation improves recognition.

- The Threshold parameter helps separate the text from the background.
  A too high value thickens black characters on white background and may cause merging, a too small value makes parts disappear.
  If the lighting conditions are very variable, automatic thresholding is a good choice.
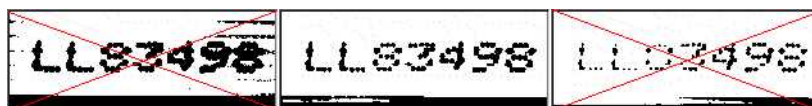
Too high threshold value (left), Threshold adjustment (middle), Too low threshold value (right)

- NoiseArea: Blob areas smaller than this value are discarded. Make sure small character features are preserved (i.e., the dot over an "i" letter).

- MaxCharWidth, MaxCharHeight: Maximum character size. If a blob does not fit in a rectangle with these dimensions, it is discarded or split into several parts using vertical cutting lines. If several blobs fit in a rectangle with these dimensions, they are grouped together.

- MinCharWidth, MinCharHeight: Minimum character size. If a blob or a group of blobs fits in a rectangle with these dimensions, it is discarded.

- CharSpacing: The width of the smallest gap between adjacent letters. If it is larger than MaxCharWidth it has no effect.
  If the gap between two characters is wider than this, they are treated as different characters. This stops thin characters being incorrectly grouped together.

- RemoveBorder: Blobs near image/ROI edges cannot normally be exploited for character recognition. By default, they are discarded.

Recognition

Recognition

The characters are compared to a set of patterns, called a **font**. A character is recognized by finding the best match between a character and a pattern in the font. After the character has been located, it is normalized in size (stretched to fit in a predefined rectangle) for matching. The normalized character is compared to each normalized template in the font database and the best matches are returned.

1. Load: reads a pre-recorded font from a disk file.

2. BuildObjects: The image is segmented into **objects** or blobs (connected components) which help find the **characters**. This step can be bypassed if the exact position of the characters is known. If the character isolation process is bypassed, you must specify the known locations of the characters: AddChar and EmptyChars.

3. FindAllChars: selects the objects considered as characters and sorts them from top to bottom then left to right.

4. `ReadText`: performs the matching and filters characters if the marking structure is fixed or a character set filter was provided.
   Character recognition: The characters are compared to a set of patterns, called a **font**. The best match is stretched to fit in a predefined rectangle and compared to each normalized template in the font database.
   A **Character set filter** can improve recognition reliability and run time by restricting the range of characters to be compared. For instance, if a marking always consists of two uppercase letters followed by five digits, the last of which is always even, it is possible to assign each character a class (maximum 32 classes) then set the character filter to allow the following classes at recognition time: two uppercase, four even or odd digits, one even digit.

Steps 2 to 4 can be repeated at will to process other images or ROIs. The `Recognize` method can be used as well.

Additional information, such as the geometric position of the detected characters, can be obtained using: `CharGetOrgX`, `CharGetOrgY`, `CharGetWidth`, `CharGetHeight`, ...

`CompareAspectRatio` makes character and font comparison sensitive to the difference between narrow and wide characters. It improves recognition when characters look like each other after size normalization.

Recognition parameters

- `MaxCharWidth`, `MaxCharHeight`: if a blob does not fit within a rectangle with these dimensions, it is not considered as a possible character (too large) and is discarded. Furthermore, if several blobs fit in a rectangle with these dimensions, they are grouped together, forming a single character. The outer rectangle size should be chosen such that it can contain the largest character from the font, enlarged by a small safety margin.

- `MinCharWidth`, `MinCharHeight`: if a blob or a group of blobs does fit in a rectangle with these dimensions, it is not considered as a possible character (too small) and is discarded. The inner rectangle size should be chosen such that it is contained in the smallest character from the font, shrunk by a small safety margin.

- `RemoveNarrowOrFlat`: Small characters are discarded if they are narrow **or** flat. By default they are discarded when they are both narrow **and** flat.

- `CharSpacing`: if two blobs are separated by a vertical gap wider than this value, they are considered to belong to different characters. This feature is useful to avoid the grouping of thin characters that would fit in the outer rectangle. Its value should be set to the width of the smallest gap between adjacent letters. If it is set to a large value (larger than `MaxCharWidth`), it has no effect.

- `CutLargeChars`: when a blob or grouping of blobs is larger than `MaxCharWidth`, it is discarded. When enabled, the blob is split into as many parts as necessary to fit and the amount of white space to be inserted between the split blobs is set by `RelativeSpacing`. This is an attempt to separate touching characters.

- `RelativeSpacing`: when the `CutLargeChars` mode is enabled, setting this value allows specifying the amount of white space that should be inserted between the split parts of the blobs.

Too large character

Too small character

Too close characters
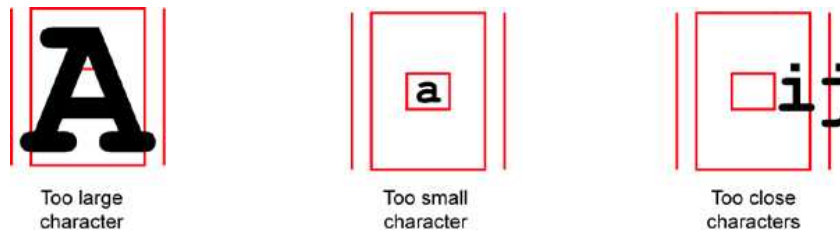
Invalid recognition settings

Advanced tuning

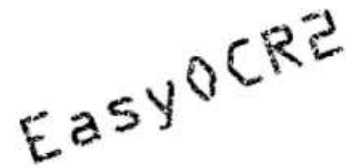These recognition parameters can be tuned to optimize recognition:

- CompareAspectRatio: when this setting is on, **EasyOCR** is less tolerant of size and takes into account the measured aspect ratio. Using this mode improves the recognition when characters look similar after size normalization as it enforces the difference between narrow and wide characters.

- Filtering the characters (in the ReadText method), can be used if the marking structure is fixed.

- When objects are larger than the MaxCharWidth property, they can be split into as many parts as needed, using vertical cutting lines.

- ESegmentationMode, character isolation mode defines how characters are isolated:
  - **Keep objects** mode: a character is a blob; no attempt is made to group blobs, thus damaged characters cannot be handled and small features such as accents and dots may be discarded by the minimum character size criterion.
  - **Repaste objects** mode: blobs are grouped to form distinct **characters** if they fit in the maximum character size and are not separated by a vertical gap, thus preserving accents and dots.

- By default, **EasyOCR** considers two characters to be on a different line if their bottom lines are too different (around 30% of the character height). Use the property LineSpacingMode to change that behavior.

4.8. EasyOCR2 - Reading Texts (Improved)

Introduction

Purpose and Principles

EasyOCR2 is an optical recognition tool designed to read short texts such as serial numbers, expiry dates or lot codes printed on labels or on parts.

Source image

It uses an innovative segmentation method to detect the blobs in an image, then it places textboxes over the detected blobs following a user-defined topology.

Image segmentation (left) and textboxes fitting (right)

The topology specifies the number of lines, words and characters in the text. You can also specify a character type (letter, digit and/or symbol) for each character in the text, to improve the recognition rate and speed.

EasyOCR2 supports the rotation of the text up to 360 degrees, handles non-uniform illumination and textured backgrounds as well as dot-printed or fragmented characters.

To recognize characters, **EasyOCR2** uses a pretrained classifier, powered by Deep Learning technologies, or a classifier that you trained on your character database.
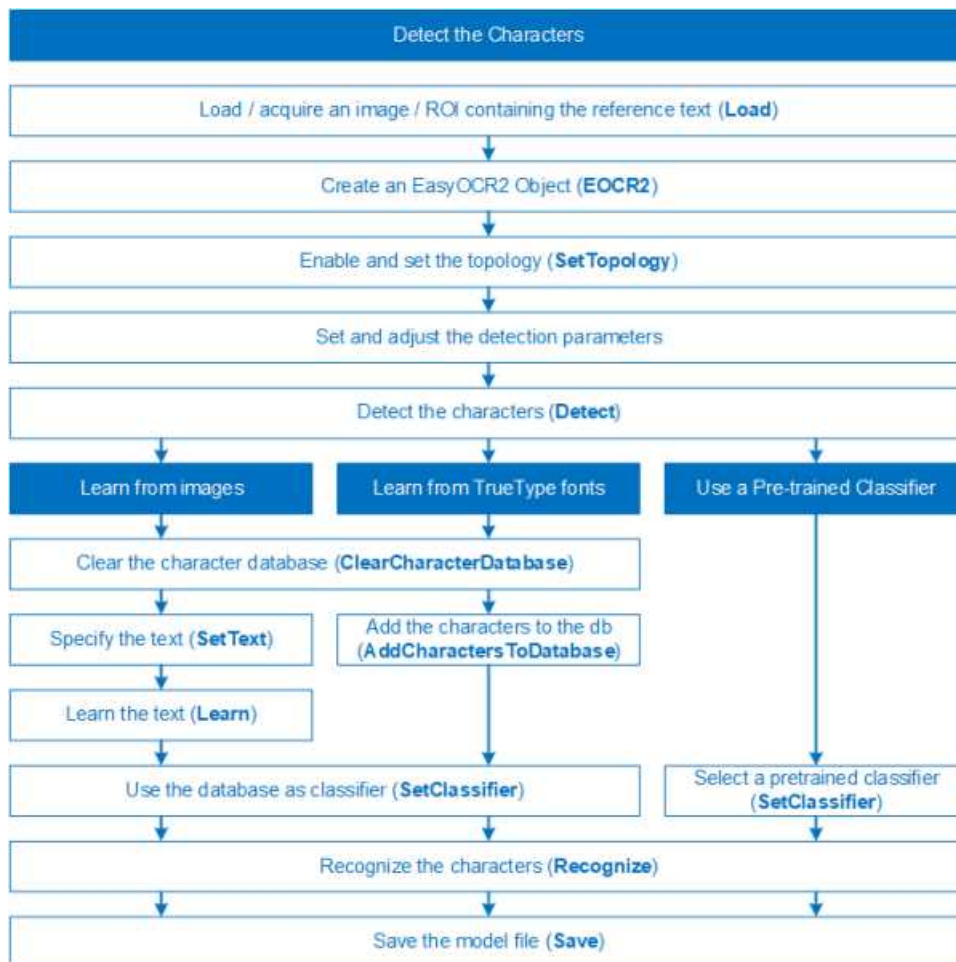
For each input character:
☐ The classifier calculates a score for all candidate outputs.
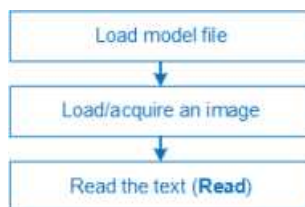☐ It returns the candidate with the highest score as the recognition result.

Text recognition

Workflow

Learning mode

Production mode

EasyOCR2 vs EasyOCR

EasyOCR2 gives better results than **EasyOCR** when dealing with:
- Unknown text rotation.
- Dotted or fragmented characters.
- Non-uniform illumination or textured backgrounds.
- Complex text topologies.

- When you have the TrueType font files that match the text, you can use these font files directly with **EasyOCR2** for the recognition, while you cannot do it with **EasyOCR**.

- Using the provided pretrained classifiers, **EasyOCR2** can perform the recognition without any preliminary training.

Using EasyOCR2

Detect the Characters

> **See also:** code snippet: "Detecting Characters" on page 192

Detection

EasyOCR2 uses an innovative segmentation method to detect the blobs in an image. Then it places textboxes over the detected blobs following a user-defined topology specifying the number of lines, words and characters in the text (see "Set the Topology" on page 113).

You can specify a character type (letter, digit and/or symbol) for each character in the text, to improve the recognition rate and speed.

Process

To find characters in an image with **EasyOCR2**:
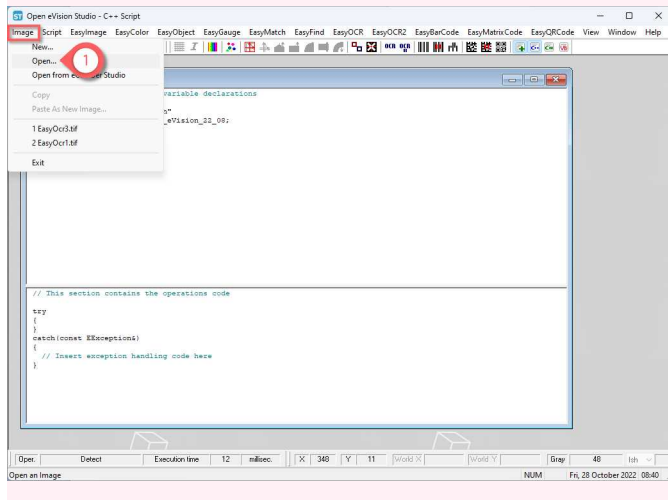
1. Load your image.

   `Code`

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

In the main menu:
1. `Image` > `Open` > select your image (`EasyOcr3.tif`).
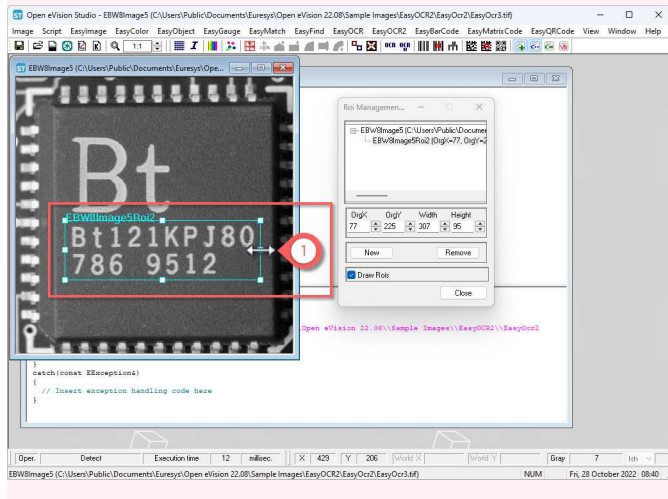

2. Attach an ROI to your image.

Code

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

Studio

In the image window:
1. Right-click > `New ROI` > move and resize in the image > `Close`.

3. Create an EOCR2 instance.

Code

```
EOCR2 ocr2;
```

Studio

In the main menu:
1. EasyOCR2 > New OCR2 Tool > name your tool.

In the tool window, in the Settings tab:
2. Select your Source Image.
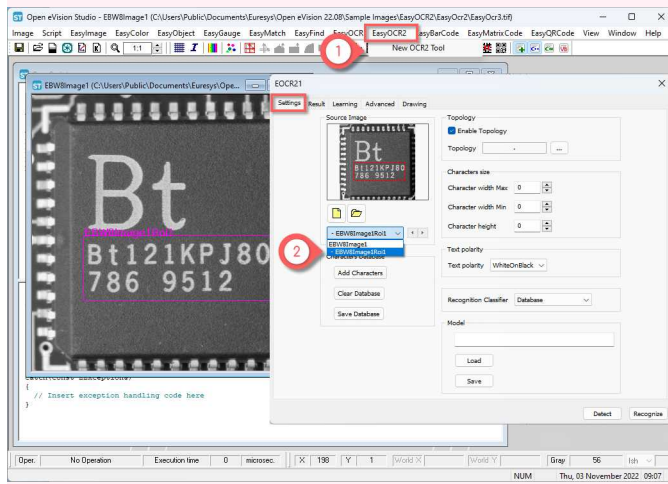
4. Enable or disable the topology.

Code

```
ocr2.SetEnabledTopology(true);
```

Studio

In the Settings tab:
1. Check or uncheck Enable Topology.

5. If you are using a topology, configure it as detailed in "Set the Topology" on page 113.

6. Set the following mandatory parameters:
   - `CharsWidthRange`: search for characters with a width in this range.
   - `CharsHeight`: search for characters with this height.
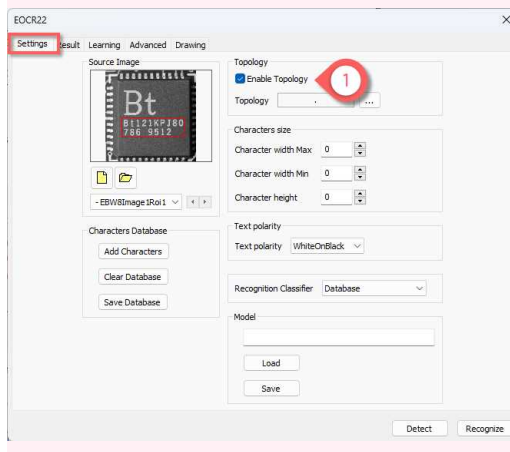   - `TextPolarity`: search for light characters on a dark background or vice versa.

   Code
   ```
   ocr2.SetCharsWidthRange(EIntegerRange(25,25));
   ocr2.SetCharsHeight(37);
   ocr2.SetTextPolarity(EasyOCR2TextPolarity_WhiteOnBlack);
   ```

   Studio

   In the `Settings` tab:
   1. Set the `Character width Max` and the `Character width Min`.
   2. Set the `Character height`.
   3. Select the `Text polarity`.
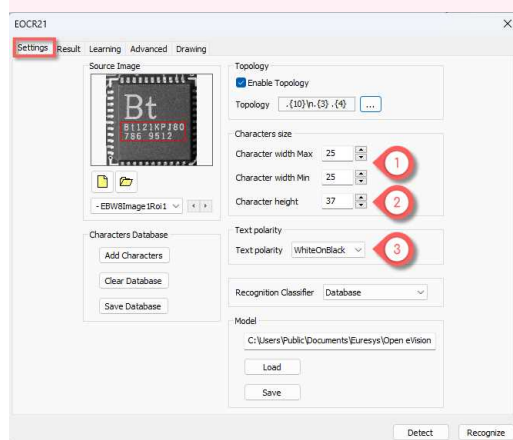
7. According to your application and needs, adjust the additional parameters:
   - "Segmentation Parameters" on page 125.
   - "Detection Parameters" on page 128.
   - If you do not use a topology: "No Topology Parameters" on page 132

8. **EasyOCR2** segments the image, finding blobs that represent (parts of) the characters.

   ► Blobs that are too large or too small to be considered as parts of a character are filtered out.

   ► **EasyOCR2** fits character boxes to the detected blobs according to a given `topology` and `detectionMethod`.

   ► The detection returns an `EOCR2Text` structure that contains a textbox and a bitmap image for each character, hierarchically stored in `EOCR2Line` -> `EOCR2Word` -> `EOCR2Char` structures.

   Code

   ```
   EOCR2Text text = ocr2.Detect(roi);
   ```

   Studio

   In any tab:
   1. `Detect`
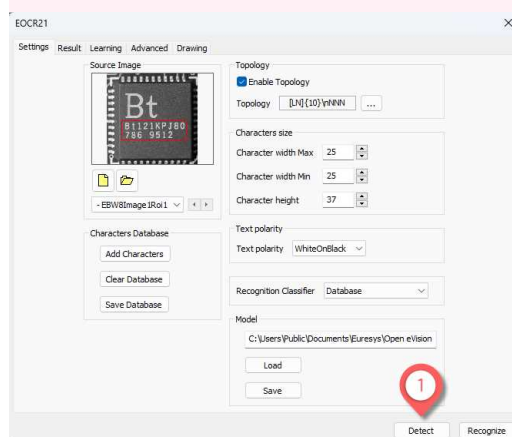
   ► **EasyOCR2** extracts the pixels inside each character box from the image.

   Use the resulting character-images to perform:
   □ "Learn the Characters" on page 117
   □ "Recognize the Characters" on page 120

Set the Topology

📖 The following parameters are set and used in the process: "Detect the Characters" on page 109.

Topology

📖

The parameter `Topology` specifies the structure of the text (number of lines, words and characters) as well as the type of characters in the text.

The box-fitting method uses this parameter to structure the textboxes it fits to the detected blobs. And the recognition method limits the number of candidates for each character based on the information of the topology.

- The topology uses the following modified regular expression (Regex) wildcards:
    - "." (dot) represents any character (not including a space).
    - "L" represents an alphabetic character:
        - "Lu" represents an uppercase alphabetic character.
        - "Ll" represents a lowercase alphabetic character.
    - "N" represents a digit.
    - "P" represents a punctuation character among: ! " # % & ' ( ) * , - . / : ; < > ? @ [\ ] _ { | } ~
    - "S" represents a symbol among: $ + - < = > | ~
    - "\n" represents a line break.
    - " " (space) represents a space between two words.
- You can combine these wildcards.
    - For example: [LN] represents an alphanumeric character.
- To specify multiple characters:
    - Add {n} at the end for n characters.
    - If the amount of characters is uncertain, specify {n,m} for a minimum of n characters and a maximum of m characters.

Pretrained classifiers

- Currently, when you use pretrained classifiers, not all types of character are recognized:
    - Only "Lu", "N" and "P" are supported.
    - Using "." in your topology only results in a character that is either a uppercase letter, a number or a punctuation character.
    - Using "L" only results in an uppercase letter.
    - Using another character type throws an exception.

Examples

- `[LuN]{3,5}PN{4} \n .{5} LL` represents a text comprised of 2 lines:
    - The first line has 1 word:
        - The word has 3 to 5 uppercase alphanumeric characters followed by a punctuation character and 4 digits.
    - The second line has 2 words:
        - The first word has 5 characters (of any type).
        - The second word has 2 letters (upper- or lowercase).
- `L{3}P N{6} \n L{3}P NNPN{4}` represents a text with 2 lines:
    - The first line has 2 words:
        - The first word has 3 uppercase letters followed by a punctuation mark.
        - The second word has 6 digits.
    - The second line also has two words:
        - The first word has 3 uppercase letters followed by a punctuation mark.
        - The second word has 2 digits, followed by a punctuation mark and 4 digits.

- .{10} \n .{7} \n .{5} .{5} \n .{5} .{7} represents a text with 4 lines:
  □ The first line contains 1 word of 10 characters (of any type).
  □ The second line contains 1 word of 7 characters
  □ The third line contains 2 words, each of 5 characters.
  □ The fourth line contains 2 words of 5 and 7 characters respectively.

Process

📄 Detect the characters in your image as described in .

1. Set the topology either as text.

Code

```
ocr2.SetEnabledTopology(true);
ocr2.SetTopology(".{10}\n.{3} .{4}");
```
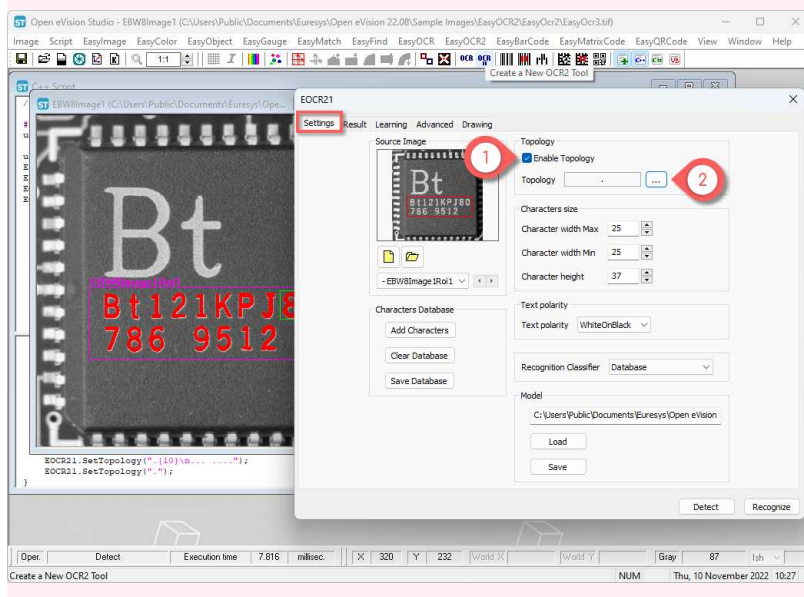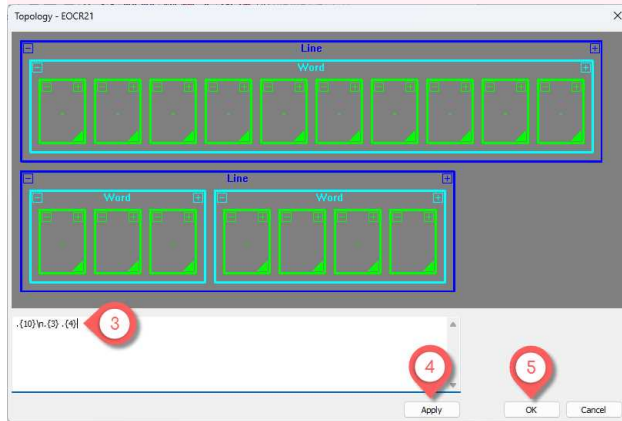
Studio

In the `Settings` tab:
1. Check `Enable Topology`.
2. `...` to open the topology editor

3. Enter the topology.
4. `Apply` to update the wizard view.
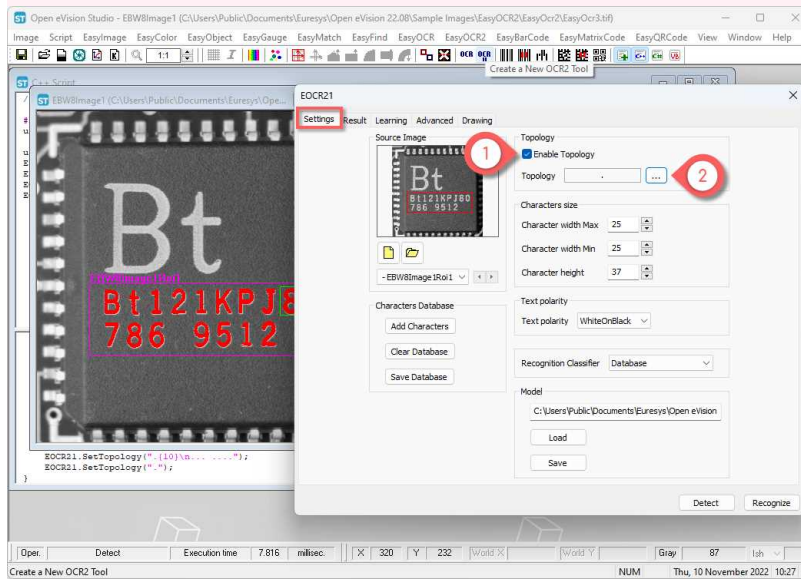**5.** `OK`

2. Or using the wizard.

`Studio`

In the `Settings` tab:
1. Check `Enable Topology`.
2. `...` to open the topology editor

3. Use [+] and [-] to add or remove a line, a word or a character.
4. Click the corner to select the character type(s).
5. OK
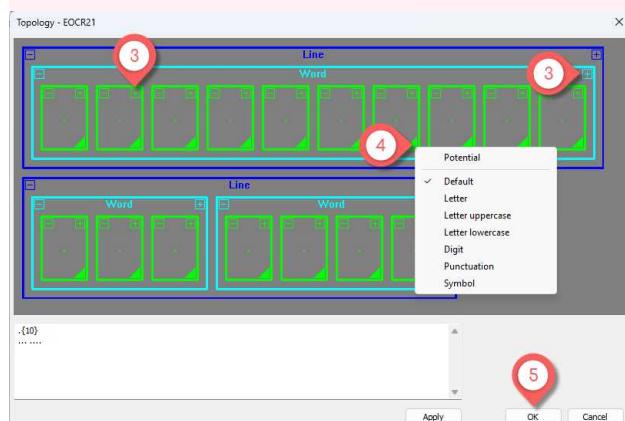
Learn the Characters

See also: "Learning Characters" on page 193

Learning

In order to recognize characters, **EasyOCR2** can use a database of known reference characters. You can generate this character database from images and/or from TrueType system fonts.

Process

📄 Detect the characters in your image as described in **"Detect the Characters" on page 109**.

1. Set the correct values of the text.

   Code

   ```
   text.SetText("Bt121KPJ80\n786 9512");
   ```

   Studio

   In the Learning tab:
   1. Select an element (character, word, line or text) in the image (see "View Elements in Open eVision Studio" on page 123).
   2. Enter the correct corresponding text.
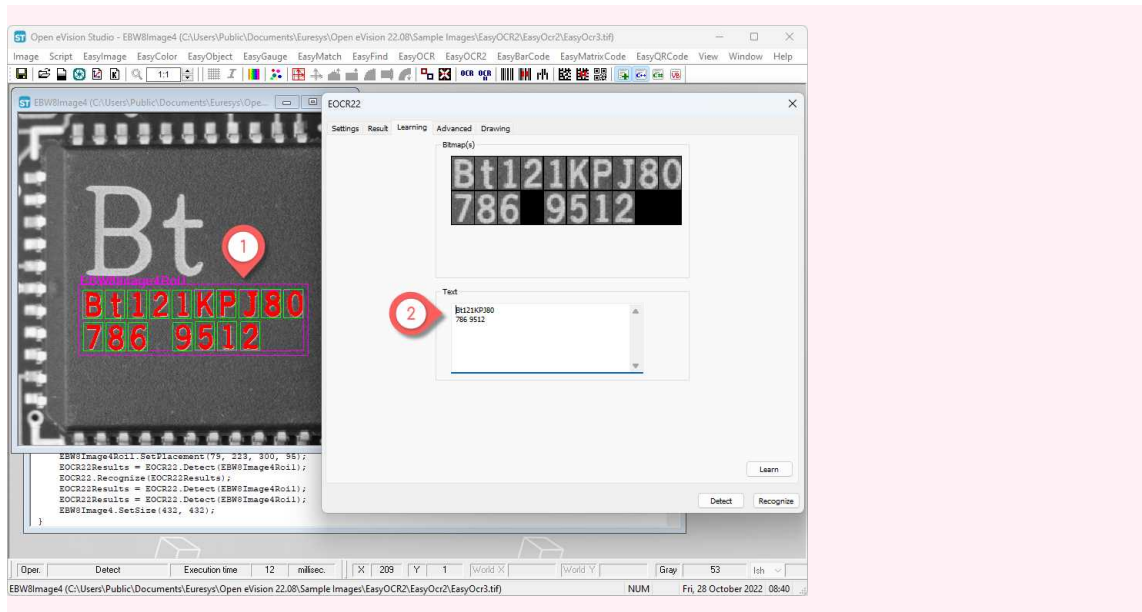
2. Add the detected characters and their correct value to the current character database.

Code

```
ocr2.Learn(text);
```

Studio

In the Learning tab:
1. Learn
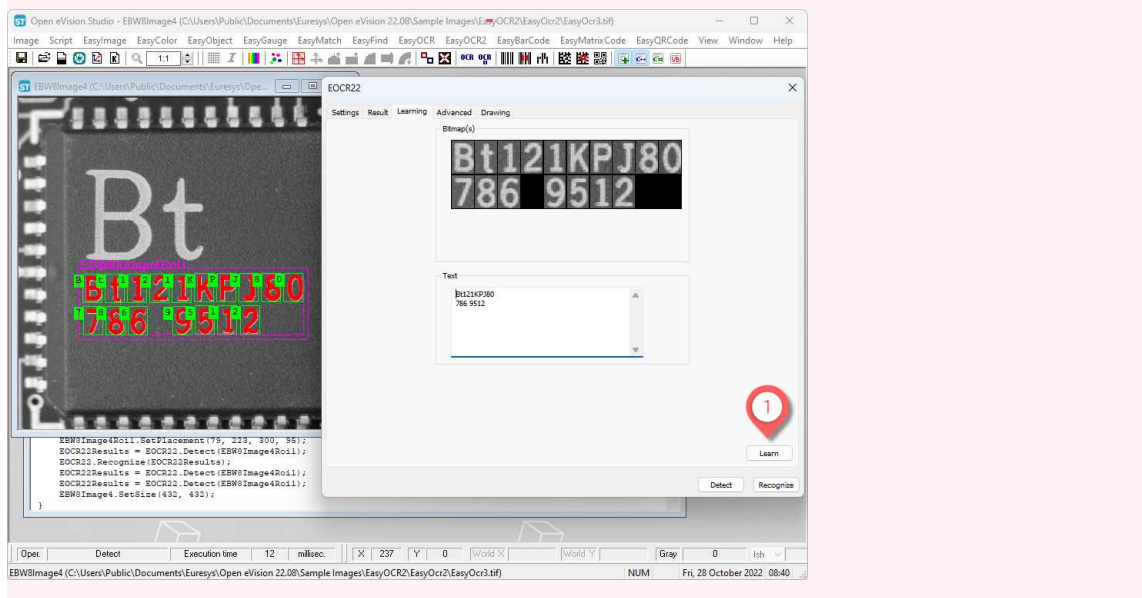
3. Save the current character database.

Code

```
ocr2.SaveCharacterDatabase("myDB.o2d");
```

Studio

In the Settings tab:
1. Save Database > enter the name of your database (.o2d file).

118
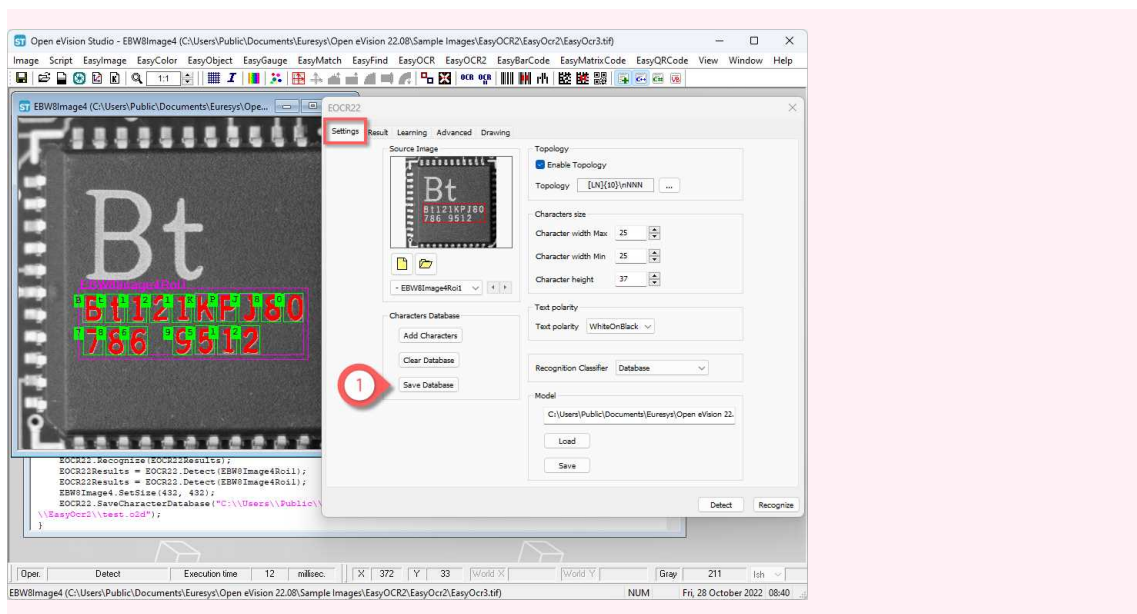
4. Alternatively, save the model file, including the detection parameters and the created character database.

Code

```
ocr2.Save("myModel.o2m");
```

Studio

In the Settings tab:
1. Save > enter the name of your model (.o2m file).
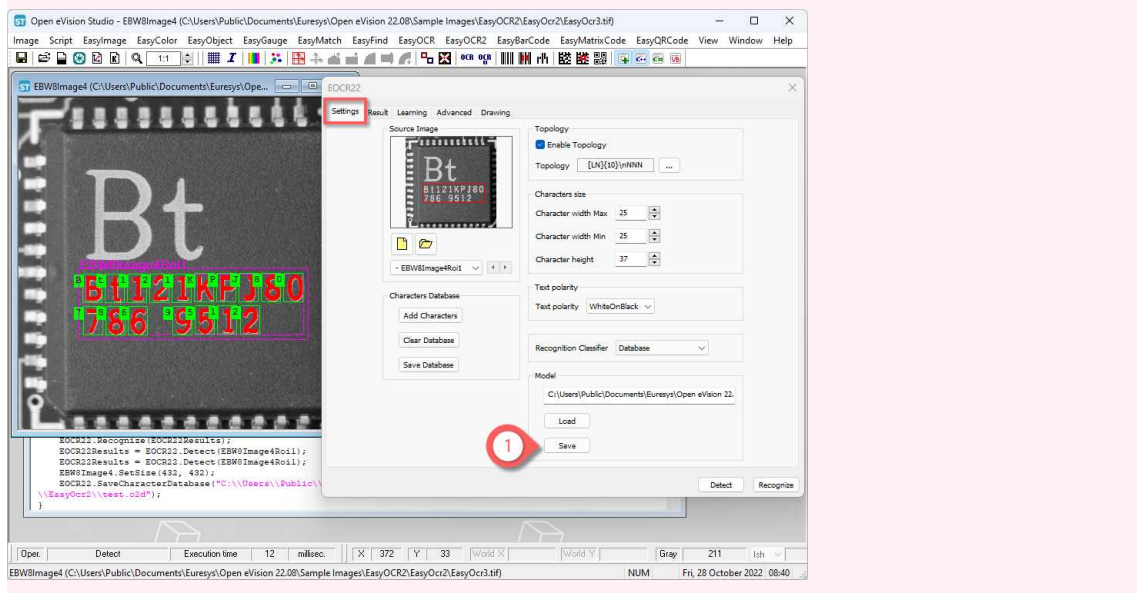
Learn characters from a True Type font

- Use AddCharactersToDatabase with the path to the True Type font to learn its characters.

Code

```
ocr2.AddCharactersToDatabase("C:\Windows\Fonts\Arial.ttf");
```

119

Clear the database

- Use ClearCharacterDatabase to clear the current character database of any existing learning.

Code

```
ocr2.ClearCharacterDatabase();
```

Studio

In the Settings tab:
1. Clear Database
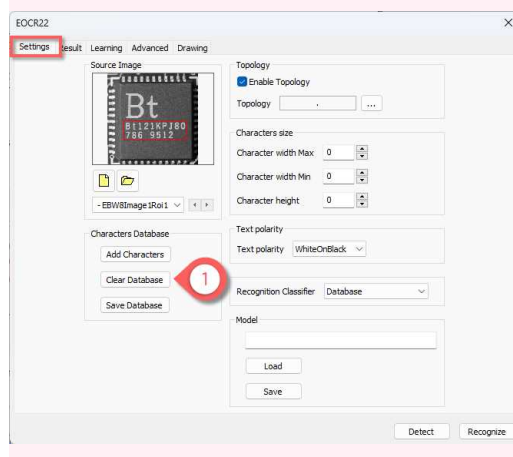
Recognize the Characters

> **See also:** code snippets: "Reading Using TrueType Fonts" on page 194, "Reading Using EOCR2 Character Database" on page 195, "Reading Using EOCR2 Model File" on page 196

Recognition

> 📖
>
> To recognize characters, **EasyOCR2** uses a pretrained classifier or a classifier that you trained on your character database.
>
> For each input character:
> - The classifier calculates a score for all candidate outputs.
> - It returns the candidate with the highest score as the recognition result.
>
> Use the topology to pass information to the classifier about each character. This reduces the number of candidates and improves the recognition rate (see "Set the Topology" on page 113).

📖
- Use the method Read or Recognize to retrieve a string with the recognition results.
  - ☐ Call Read to detect and recognize the characters in one step.
  - ☐ Call Detect to extract the text from the image then Recognize to recognize the extracted text. This allows you to modify elements of the detected text before the recognition.
- To access more information about the results, use the method ReadText that returns an EOCR2Text structure with:
  - ☐ The coordinates and the size of each textbox,
  - ☐ A bitmap image of each textbox,
  - ☐ A list of the recognition scores for each character.

Process

📄 Detect the characters in your image as described in "Detect the Characters" on page 109.

1. Select the Classifier used by EOCR2 for recognition.

   By default:
   - ☐ EOCR2Classifier_DatabaseClassifier: EOCR2 uses the current character database.

   Pretrained classifiers used in different contexts:
   - ☐ EOCR2Classifier_Industrial_A_Z_0_9_P for characters used in an industrial context without a specific font.
   - ☐ EOCR2Classifier_OCRA_A_Z_0_9_P for characters using the OCR-A font.
   - ☐ EOCR2Classifier_SEMI_A_Z_0_9_P for characters using the SEMI-OCR font.

   Code

   ```
   ocr2.SetClassifier(EOCR2Classifier_Database);
   ```

   Studio

   In the Settings tab:
   1. Select a Recognition Classifier.
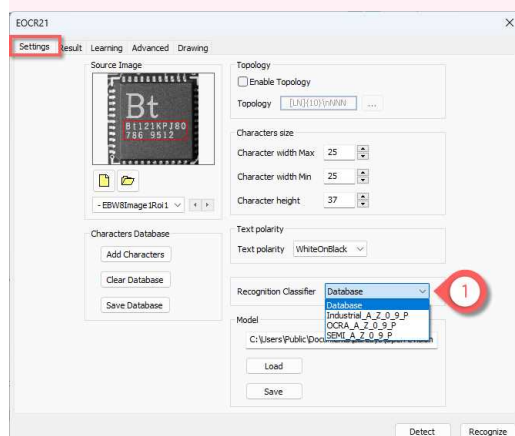
2. Recognize or read the decoded text.

Code

```
EOCR2Text text = ocr2.Detect(roi);
std::string result = ocr2.Recognize(EOCR2Text);
```

or

```
std::string result = ocr2.Read(roi);
```

Studio

In any tab:
1. Recognize
2. The results are displayed on the image.
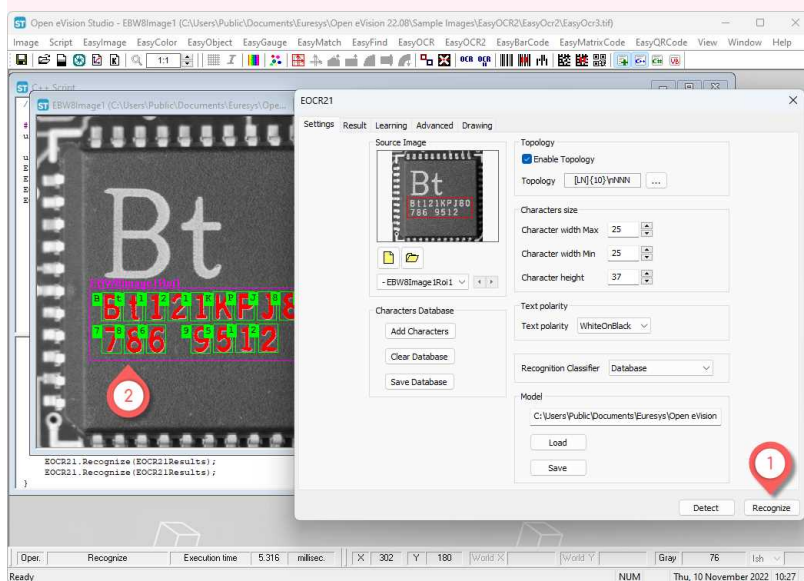
Read a model file

- Use Load to read the model (.o2m) file from your disk.

  The model file contains:
  □ All the detection parameters,
  □ The topology,
  □ The reference character database.

Overriding the classifier

- Use SetClassifier with a symbol and a classifier to override the current classifier and assign another classifier (EOCR2Classifier) to a specific symbol or a combination.

NOTE: Currently you can override only the EOCR2Classifier_DatabaseClassifier.

Accelerate the recognition

A character is expected to be recognized in 4 ms, half of this with multithreading and even less with a GPU.

Multithreading

- With pretrained classifiers, use `Easy::GetMaxNumberOfProcessingThreads` to multithread the recognition and to determine how many threads you can use for the recognition.

GPU acceleration

- You can also use a compatible GPU to accelerate even more the recognition.
  - □ Use `Easy::IsGPUAvailable` to determine if there is a compatible GPU available.
  - □ Use `EOCR2:SetEnableGPU(true)` to enable the GPU-recognition.

Open eVision Studio Tools

View Elements in Open eVision Studio

Drawing

In **Open eVision Studio**, you can display the elements you want on your image (blobs, characters, words, lines, text and results). You can also choose a color for each element and configure the display of the results.

> ✓ **TIP**
> Enable the drawing of the elements that you need to select during the learning process ("Learn the Characters" on page 117).

Process

1. In **Open eVision Studio**, display the elements you want in the chosen drawing color.

   `Studio`

   In the `Drawing` tab:
   1. In the `Segmentation` area > check to display the blobs > `...` to choose the color.
   2. In the `Detection` area > check to display the elements > `...` to choose the color.
   3. In the `Recognition` area > check to display the results > configure the display.
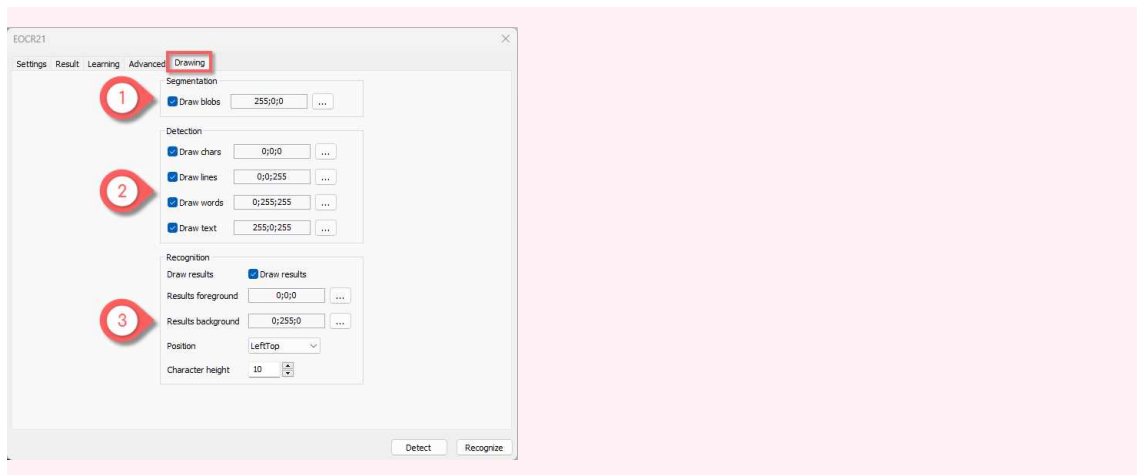
View Results in Open eVision Studio

Drawing

In **Open eVision Studio**, you can display the results of a character reading including the matching score for all possible candidates. The list of candidates depends on the topology.

Process

1. In **Open eVision Studio**, display the results for the selected character.

    Studio

    In the image:
    1. Select a character in your image.

    In the `Result` tab:
    2. The table lists the score for each candidate.
    3. The character with the higher score is selected as recognized.
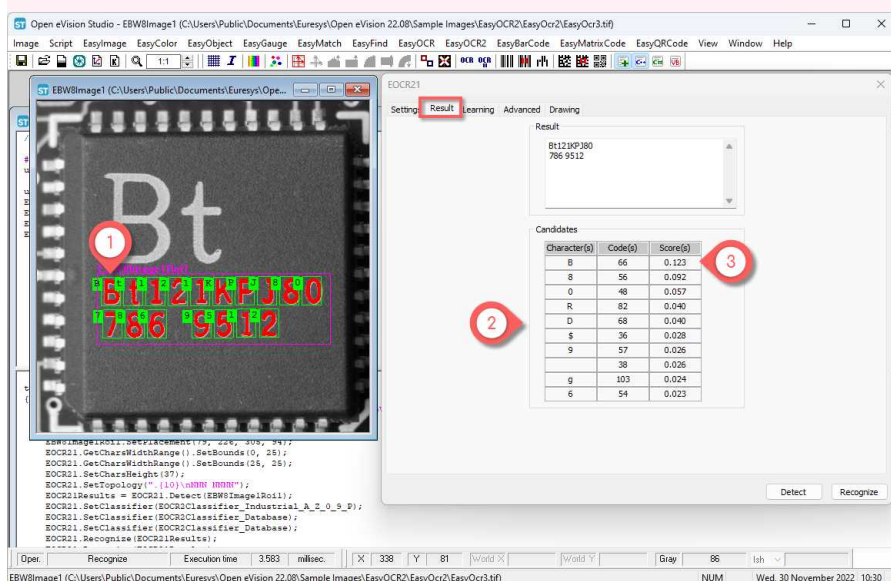
Setting the Parameters

Segmentation Parameters

📄 The following parameters are set and used in the process: **"Detect the Characters" on page 109**.

Select the segmentation method

- Use the parameter SegmentationMethod to select the algorithm used for segmentation:
  - ☐ EOCR2SegmentationMethod_Global: the global segmentation uses a simple threshold. It is faster and best suited for a clear background.
  - ☐ EOCR2SegmentationMethod_Local (default): the local segmentation is more complex and is best suited for a non uniform background.

Configure the global segmentation

- Use the parameter GlobalSegmentationThresholdMode to set the threshold computation method for the segmentation. This parameter is a EThresholdMode.

- Use the parameter EnableSecondPassGlobalSegmentation to perform the segmentation twice during the first pass to have more accurate results when the text background is not just plain.
  - ☐ This functionality is only available when GlobalSegmentationThresholdMode is set to the minimum residue (MinResidue).

Code

```
ocr2.SetSegmentationMethod(EOCR2SegmentationMethod_Global);
ocr2.SetGlobalSegmentationThresholdMode(MinResidue);
ocr2.SetEnableSecondPassGlobalSegmentation(True);
```

In the `Advanced` tab:
1. Set the `Segmentation Method` to `Global`.
2. `Global Segmentation Method` > `...` to select the threshold.
3. Select a threshold mode.
4. Check `Second Pass Segmentation`.
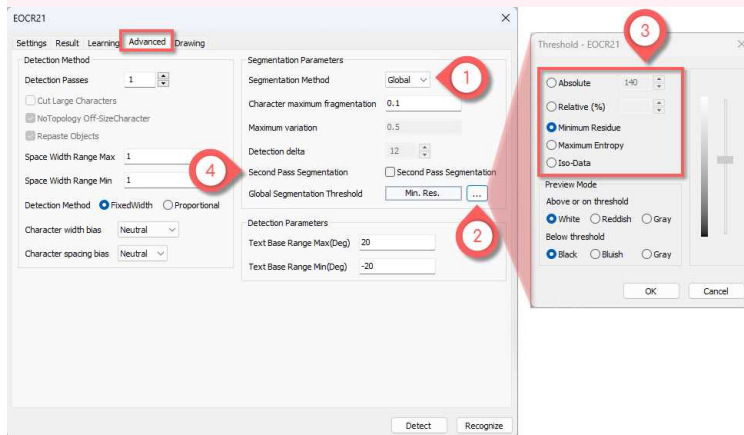
Configure the local segmentation

- Use the parameter `MaxVariation` to define how stable a blob in the image should be in order to be considered a potential character. A region with clearly defined edges is generally considered stable while a blurry region is not.
  - Set this parameter between 0 and 1 (the default setting is 0.25).
  - A low setting allows only very stable blobs.
  - A high setting allows detection of blobs that are more unstable.

- Use the parameter `DetectionDelta` to set the range of grayscale values used to determine the stability of a blob.
  - Set this parameter between 1 and 127 (the default setting is 12).
  - With a low setting, the algorithm is more sensitive to noise.
  - With a high setting, the algorithm is insensitive to blobs that have a low contrast with the background.

Code

```
ocr2.SetSegmentationMethod(EOCR2SegmentationMethod_Local);
ocr2.SetMaxVariation(0.5);
ocr2.SetDetectionDelta(12);
```

In the Advanced tab:
1. Set the Segmentation Method to Local.
2. Set the Maximum variation.
3. Select the Detection delta.
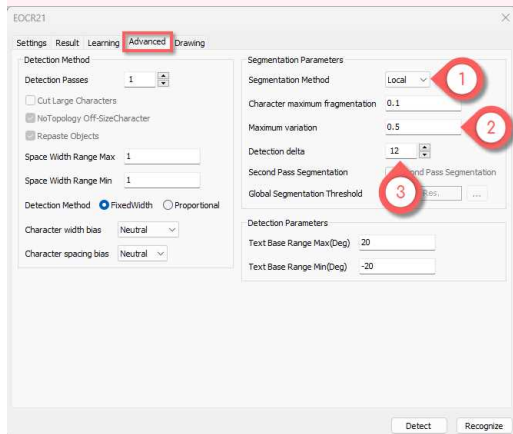
Adjust the fragmentation

- Use the parameter CharsMaxFragmentation to set how small a blob can be for the segmentation algorithm to consider it as (part of) a character.
  - Set this parameter between 0 and 1 (the default setting is 0.1).
  - The minimum allowed area of a blob is given by:

$$minArea = CharsMaxFragmentation \times CharsHeight \times min\left(CharsWidthRange\right)$$
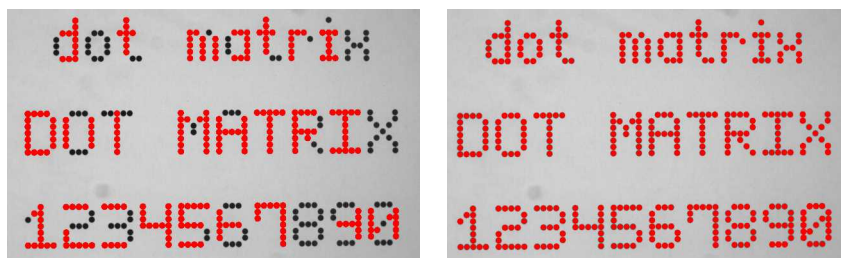
Left: CharsMaxFragmentation = 0.1 (default) leads to incomplete segmentation results
Right: CharsMaxFragmentation = 0.01 gives better results

```
ocr2.SetCharsMaxFragmentation(0.1);
```

127

Studio

In the Advanced tab:

1. Set the Character maximum fragmentation.
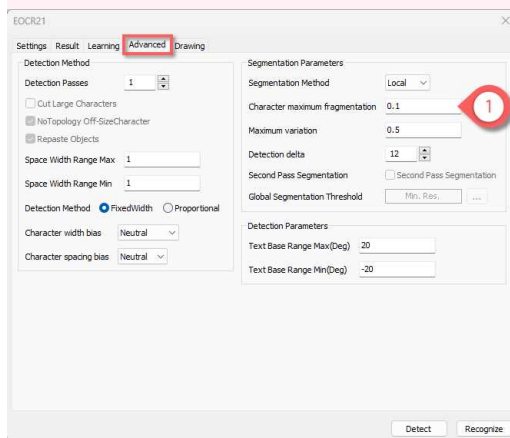
Detection Parameters

📄 The following parameters are set and used in the process: "Detect the Characters" on page 109.

Select the detection method

- If the topology is required, use the parameter DetectionMethod to select the algorithm used for fitting.
  - □ EOCR2DetectionMethod_FixedWidth (default) is optimized for texts using a fixed-width font (including dotted text).

FIXED WIDTH

A fixed-width font, processed with EOCR2DetectionMethod_FixedWidth

  - □ EOCR2DetectionMethod_Proportional is optimized for texts using a proportional font.

PROPORTIONAL

A proportional font, processed with the EOCR2DetectionMethod_Proportional

📖
- With the fixed-width method, all the character boxes have the same width and they do not necessarily fit tightly around the characters.

- With the proportional method, the character boxes fit tightly around the characters. If any character falls outside the range of allowed widths, the detection fails.

Configure the fixed-width font detection

- Use the parameter `RelativeSpacesWidthRange` to specify how wide the spaces between the words may be.

  The box-fitting method tests the following range of spaces:

  $$min\ (\text{RelativeSpacesWidthRange})\ \times\ min\ (\text{charsWidthRange})$$
  $$\leq space \leq$$
  $$max\ (\text{RelativeSpacesWidthRange})\ \times\ max\ (\text{charsWidthRange})$$

NOTE: The lower bound of this parameter is also used when the topology is not required.

- The parameter `CharsWidthBias` biases the optimization toward wider or narrower character boxes.

- The parameter `CharsSpacingBias` biases the optimization toward smaller or larger spacing between character boxes.

  Code

  ```
  ocr2.SetDetectionMethod(EOCR2DetectionMethod_FixedWidth);
  ocr2.SetRelativeSpacesWidthRange(0.90f, 1.10f);
  ocr2.SetCharsWidthBias(Neutral);
  ocr2.SetCharsSpacingBias(Neutral);
  ```

  Studio

  In the `Advanced` tab:
  1. In the `Detection Method`, check `FixedWidth`.
  2. Enter `Space Width Range Max` and `Space Width Range Min` as float.
  3. Select the `Character width bias`.
  4. Select the `Character spacing bias`.
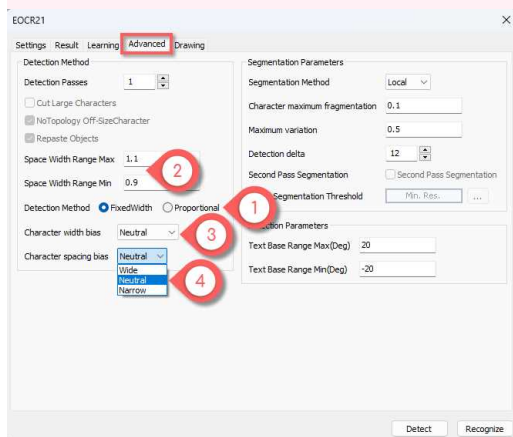
Configure the proportional font detection

- Set the parameter `EnableCutLargeCharacter` if you want that the detection tries to split segmented blobs that are too wide into different characters.

NOTE: This parameter is also used when the topology is not required.

Code

```
ocr2.SetDetectionMethod(EOCR2DetectionMethod_Proportional);
ocr2.SetEnableCutLargeCharacter(true);
```

Studio

In the `Advanced` tab:
1. In the `Detection Method`, check `Proportional`.
2. Check `Cut Large Characters`.
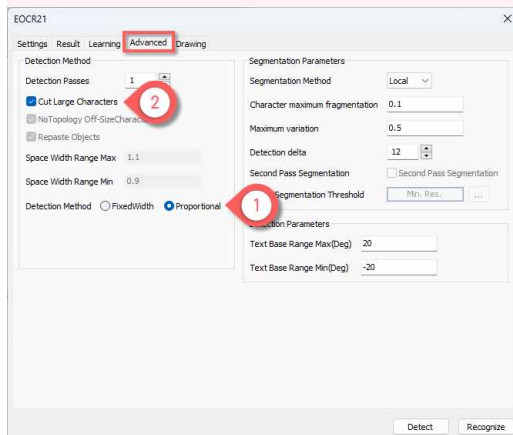
Set the text rotation angle

- Use the parameter `TextAngleRange` to specify how the text in the image may be oriented.

  The box-fitting method tests the following range of rotation angles:

  $$min\,(\text{TextAngleRange}) \leq angle \leq max\,(\text{TextAngleRange})$$

  - The angles are defined with respect to the horizontal.
  - To set the unit for the angles (degrees, radians, revolutions or grades), use `easy.SetAngleUnit`.
  - The default setting is [-20, 20] degrees.

  Code

  ```
  ocr2.SetTextAngleRange(-25.00f, 25.00f);
  ```

Studio

In the Advanced tab:
1. Enter `Text Base Range Max(Deg)` and `Text Base Range Min(Deg)` as float.
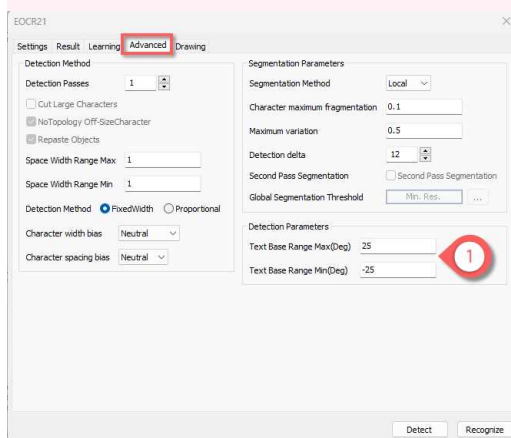
Set a second detection pass

- Use the parameter `NumDetectionPasses` to configure a second pass for the fitting of textboxes.
  - □ The first pass fits textboxes to all detected blobs.
  - □ The second pass selects only the blobs that are covered by the textboxes from the first pass. It fits textboxes to that subset of blobs, possibly resulting in a more optimal fit.
  - □ Set this parameter to either 1 or 2.
  - □ The default setting is 1.

Left: `NumDetectionPasses` = 1 and the text angle estimate is slightly off
Right: `NumDetectionPasses` = 2 and the text angle estimate is better

Code

```
ocr2.SetDetectionPasses(2);
```

In the Advanced tab:
1. Set the Detection Passes to 1 or 2.
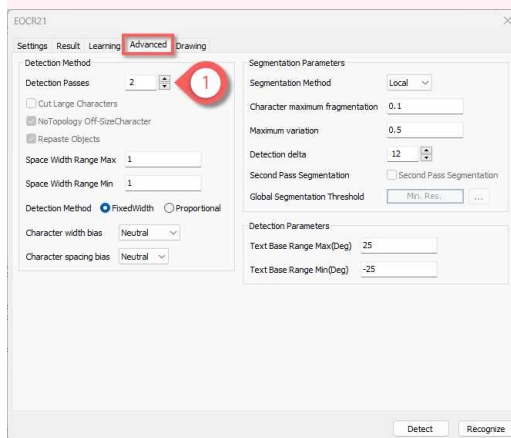
No Topology Parameters

📄 The following parameters are set and used in the process: "Detect the Characters" on page 109.

If you do not use a topology, **EasyOCR2** fits boxes to the detected blobs as best as it can.

Detection parameters

You can use the following parameters that are described in "Detection Parameters" on page 128:

- ☐ In the section "Configure the proportional font detection": EnableCutLargeCharacter.
- ☐ In the section "Configure the fixed-width font detection": RelativeSpacesWidthRange.
- ☐ In the section "Set the text rotation angle": TextAngleRange.

Detect characters that are out of the size range

- ● Use the parameter EnableOffSizeCharacter to allow the detection of characters whose size (width and height) is out of the size range but that are in the vicinity of characters in the valid size range.

Code

```
ocr2.SetEnableOffSizeCharacter(true);
```

In the Advanced tab:
1. Check NoTopology Off-SizeCharacter.
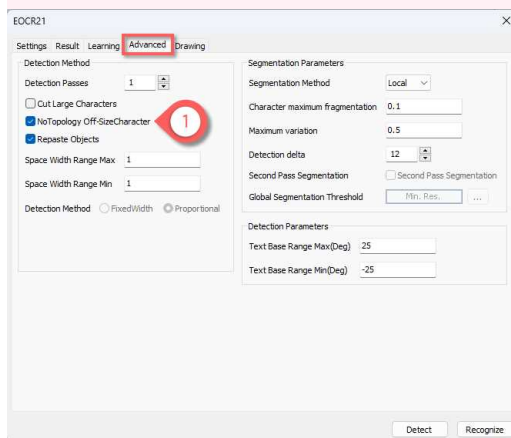
Group blobs of a same character

- Use the parameter RepasteObjects to group the blobs that the detection considers as belonging to the same character.
  - □ This can improve the detection when you have a clean segmentation and the characters are close to each other.
  - □ The default setting is True and the detection tries to merge the blobs.

```
ocr2.SetRepasteObjects(true);
```

In the Advanced tab:
1. Check Repaste Objects.
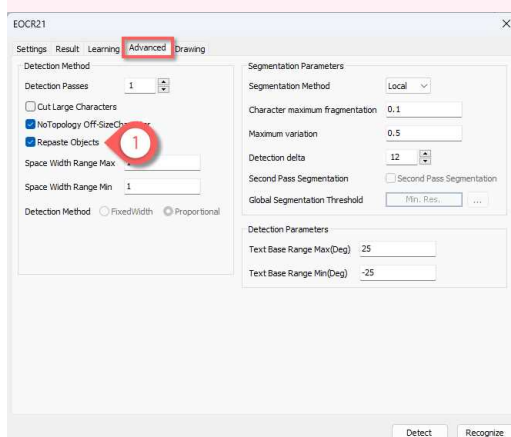
4.9. Code Grading

What Is Grading?

The "code grading", or "code quality verification", is a process that assesses the quality of a printed or engraved code.

- Different grading standards exist, each pertaining to one or more code types, that specify several grading parameters and how to compute them. In addition to these parameters, a global grade is often computed to summarize the overall quality of the code.

- When a code receives a failing grade, identifying which grade(s) caused the failure can help to locate the physical problem.

- The grades are generally returned:
  - As letters, from A (best) to F (worst).
  - As numbers: from 4.0 (best) to 0.0 (worst).

- While grading is meant to assess the print quality of a code, it also requires specific capture conditions to give accurate results. However, if these conditions are not met, the grading results might give insight on how you can improve your acquisition setup for better results.

- The grading standards currently implemented within **Open eVision** are:
  - **ISO/IEC 15416** for 1D bar codes
  - **ISO/IEC 15415** for data matrix codes and QR codes
  - **ISO/IEC 29158** for data matrix codes and QR codes
  - **SEMI T10-0701** for data matrix codes

How to Compute the Grading with Open eVision

Read the grades in Open eVision Studio

In **Open eVision Studio**:

1. Open an image.

2. Check `Compute Grading` in the first tab of the tool.

3. Read a bar code, a matrix code or a QR code.

4. Open the tab(s) named `Grading` or after the standard name.

▶ The tables list the grades (both as numbers and as letters) and the measurement values.
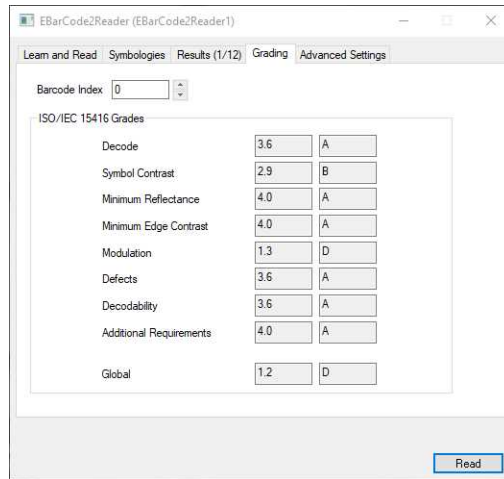
The ISO 15416 grades in EasyBarCode2
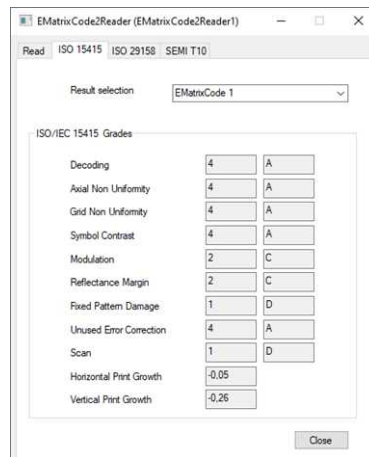
The ISO 15415 grades and values in EasyMatrixCode2
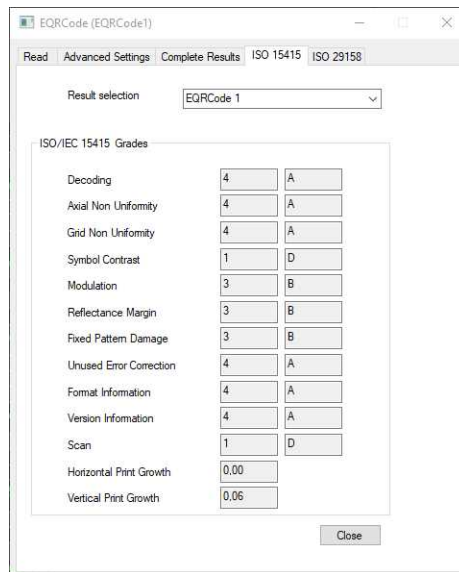
The ISO 15415 grades and values in EasyQRCode

Compute the grades in the API

In **Open eVision**, each type of code is handled by a specific reader class:
- ☐ 1D Barcode: EasyBarCode2::EBarCodeReader
- ☐ QR Codes: EQRCodeReader
- ☐ Data Matrix Codes: EasyMatrixCode2::EMatrixCodeReader

While the reader classes are different, the way to compute and retrieve the grading results is the same:

1. Instantiate the relevant reader class.

2. Use `SetComputeGrading(true)` to enable the grading computation.

3. For **ISO 29158** only, use `SetIso29158CalibrationParameters` to set up the calibration parameters (see "ISO/IEC 29158 for Data Matrix and QR Codes" on page 146).

4. Use Read on an image containing a code to read and grade the code.

5. Use the corresponding accessor(s) to retrieve the calibration results for the relevant grade(s):
- ☐ For **ISO 15415**: `GetIso15415GradingParameters`
- ☐ For **ISO 15416**: `GetIso15415GradingParameters`
- ☐ For **ISO 29158**: `GetIso29158GradingParameters`
- ☐ For **SEMI T10-0701**: `GetSemiT10GradingParameters`

ISO/IEC 15416 for 1D Bar Codes

ISO 15416 is a standard that establishes the guidelines on how to assess the print quality of linear (1D) bar code symbols.

It provides a set of quality indicators that give insight on specific areas of the bar code quality. You can use these indicators to compute an overall grade for the inspected bar code.

> **NOTE**
>
> In **Open eVision** 23.12, **ISO 15416** grading is only available for the **Ean13**, **Code128** and **Gs1-128** symbologies.
> You can find more information on these symbologies on the ISO website (must be purchased) and on the GS1 website (for free).

The scan lines
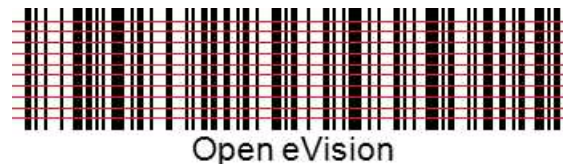
Each grade is computed on 10 horizontal scan lines:

1. A margin of 10% is removed at the top and at the bottom of the bar code.

2. The 10 scan lines are distributed evenly on the remaining bar code.

3. A grade between 4.0 and 0.0 is computed for each line.

4. The global grade is the average of the grades for each line.

The decoding

For each scan line, a global threshold is computed:

1. All pixels above the threshold belong to a space and all pixels below to a bar.

2. **Open eVision** computes the exact bar / space widths with sub-pixel interpolation.

3. **Open eVision** decodes the scan line with the reference decoding algorithm of the symbology.

> **NOTE**
>
> This decoding process is imposed by **ISO 15416**.
> **Open eVision** uses a different and more robust algorithm (based on gradients instead of thresholds) to locate the bars and the spaces of a bar code. Thus, **Open eVision** can recognize and read bar codes that are otherwise not decoded by this simpler **ISO 15416** process (for example when the illumination is not uniform along the bar code).

The grade values

Each grade is represented by a number from 4.0 to 0.0 with 0.1 steps or by letters from A to F.

- □ In this document, we use the numbers notation.
- □ **Open eVision** returns integers from 40 to 0 instead of float values from 4.0 to 0.0 to avoid any rounding issue.

Numeric grade	Alphabetic grade
4.0 to 3.5	A
3.4 to 2.5	B
2.4 to 1.5	C
1.4 to 0.5	D
0.4 to 0.0	F

The ISO 15416 quality indicators

The decode grade

- For each scan line, the *decode grade* is set to 4.0 if the decoding succeeds and 0.0 otherwise.

Symbol Contrast grade

- The *symbol contrast grade* indicates the fraction of the total image contrast used by the bar code for each scan line.
- ► If the printing contrast is too weak, if the lighting is not bright enough or if the camera exposure time or gain is too small, the grade is low.

The bars and the spaces are too close on the gray scale

The minimum reflectance grade

- For each scan line, the *minimum reflectance grade* is set to 4.0 if the lowest gray value is less than half of the highest gray value and 0.0 otherwise.
- ► If the printed barcode or lighting is too bright or if the camera exposure or gain is too high, the grade is low.

The minimum edge contrast grade

- The *minimum edge contrast grade* indicates the smallest contrast between two adjacent bar and space (including the quiet zones). For each scan line, this grade is 4.0 if the minimum edge contrast is at least 15% (38 gray values) and 0.0 otherwise.

▶ A dirty background can result in a low grade.

One space is too dark

The modulation grade

- The *modulation grade* indicates the importance of the minimum edge contrast relative to the symbol contrast.

▶ A dirty background can result in a low grade.

There is a light defect in the bar

The defects grade

- The *defects grade* indicates the importance of the irregularities found within the elements and the quiet zones.
  - □ Damaged or dirty bars or spaces can result in a low grade.

The decodability grade

- The *decodability grade* indicates the importance of the differences between the distances measured and expected.

▶ If the distances measured are far from those expected, the grade is low.

- Bad measured distances can have several causes:
  - ☐ The code is badly printed.
  - ☐ The image acquired by the camera is blurry / noisy.
  - ☐ The image acquired by the camera has a resolution that is too small.

The bar is 1 px too large and the space is 1 px too thin

The additional requirements grade

- The *additional requirements grade* indicates a requirement that is specific to a symbology.

▶ **Code128**, **Gs1_128** and **Ean13** grade the size of the quiet zone of the bar code. If the quiet zone is too small for a line, the associated grade is 0.0.

The global grade

- For each line, the *global grade* is the smallest of all the other grades.

▶ If two scan lines yield different decoded strings, the *global grade* is set to 0.0, irrespective of the other scan lines.

ISO/IEC 15415 for Data Matrix and QR Codes

ISO 15415 is a standard that establishes the guidelines on how to assess the print quality of 2D bar code symbols, either multirow bar codes or two-dimensional matrix symbologies such as data matrix and QR codes.

It provides a set of quality indicators that give insight on specific areas of the bar code quality. You can use these indicators to compute an overall grade for the inspected bar code.

The ISO 15415 as a print quality assessment tool

To be used as a print quality assessment tool as intended, the **ISO 15415** standard requires very specific acquisition conditions.

These conditions might include, but are not limited to:
- ☐ A camera perpendicular to the plane of the code to be assessed.
- ☐ 4 light sources, placed at the 4 cardinal points around the code and providing specific illumination at a 45° angle.
- ☐ An 8-bit gray-scale digitization.
- ☐ A 1:1 magnification lens.
- ☐ At least 5 pixels per module on the produced image.
- ☐ A symbol centered in the image.

For more information about those conditions, please refer to chapter 7 of the standard.

The ISO 15415 as a symbol and/or setup quality assessment tool

Obviously, the conditions above are difficult to meet. However, the **ISO 15415** standard is still useful as an assessment tool of your symbol and/or setup quality even if all those conditions are not met.

In the following paragraphs, we will give you insights on how to use the returned grades to that effect.

> **NOTE**
>
> As **Open eVision** assesses one image at a time, it provides, according to the standard, a scan grade.
> To compute an **ISO 15415** overall grade, you need to make a total of 5 acquisitions with different symbol rotations, grade each of these acquisitions using **Open eVision** and then average the results.

The ISO 15415 quality indicators

The decode grade

- The *decode grade* indicates if the symbol is readable (if it can be correctly decoded) by the symbology reference decoding algorithm.

The symbol contrast grade

- The *symbol contrast grade* is a measure of the relative difference of reflectance between the brightest and the darkest module in the symbol.

▶ If this grade is low, the detection and the digitization are more difficult as it is difficult to separate the code from the background.
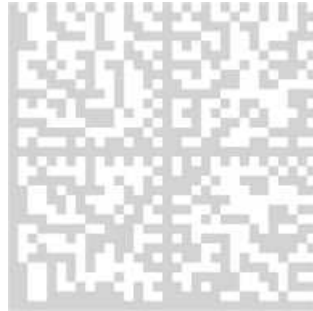
A data matrix with a low contrast

The modulation grade

- The *modulation grade* is a measure of the uniformity of the reflectance of the dark and light modules.

▶ If this grade is low, the digitization is more difficult, as the variations prevent finding an easy way to separate white from black.
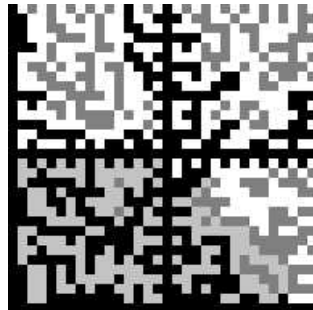
A data matrix with a low modulation grade

The reflectance margin grade

- The *reflectance margin grade* is a measure of how the modules colors are distinguishable relative to the global threshold. This global threhold is the mean reflectance of the brightest and the darkest module.

▶ If this grade is low, the digitization is less reliable. As the cells are too close to the separation limit, the light and dark cells can be confused.

A data matrix with a low modulation grade

The fixed pattern damage grade

- The fixed patterns of the symbols characterize the symbols as such. The *fixed pattern damage grade* indicates the likelihood that the symbol is correctly located and identified in the image.
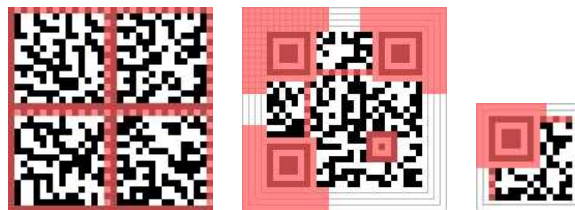
The areas relevant for this grade

The axial nonuniformity grade

- The *axial nonuniformity grade* is a measure of the isotropy of the principal axis of the symbol relative to its corresponding ideal symbol geometry.

▶ Usually, a low grade here only causes issues if a column or a row becomes too small to be accurately sized during the gridding.

A data matrix stretched along its horizontal axis

The grid nonuniformity grade

- The *nonuniformity grade* is a measure of the maximum relative deviation of the intersection of the lines of the estimated grid from to its corresponding intersection in an ideal grid.

▶ The lower the grade, the less uniform the shapes of the modules of the symbol are. However, this grade relies on a maximum deviation. It is thus quite unstable and its practical usage is limited.
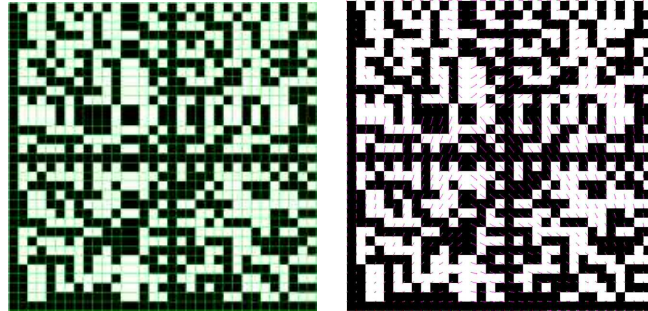
On the left, the grid is computed from a data matrix in which the vertical and horizontal lines comprise one specific module and it had been stretched
On the right, the green dots correspond to the expected positions of the corners of the modules if the grid had been uniform

The unused error correction grade

- The *unused error correction grade* indicates the extent to which the error correction algorithm capacity was used to decode the symbol.

▶ A low grade is a sign that any further degradation of the acquisition conditions will lead to a decoding failure.
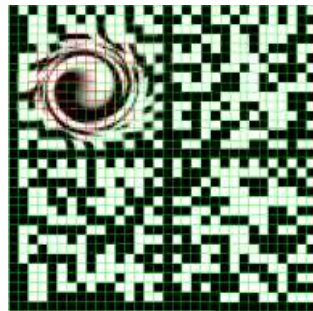
A data matrix with damaged modules that requires the use of error correction

The format information grade (QR codes and micro QR codes)

- The *format information grade* indicates the readability of the format information blocks.

▶ The lower the grade, the more errors must be corrected to recover the error correction level and the masking pattern that are necessary to reliably decode the symbol.

The areas relevant for this grade

The version information grade (QR codes)

- The *version information grad*e indicates the readability of the version information blocks.

▶ The lower the grade, the more errors must be corrected to recover the version of the symbol.

NOTE: This is relevant only for versions ≥ 7.

The areas relevant for this grade

The scan grade

- The *scan grade* is the lowest grade value of all the grade indicators. It is a measure of the overall quality of the code in the image.

ISO/IEC 29158 for Data Matrix and QR Codes

The **ISO 29158** standard is a modification and an extension to the **ISO 15415** standard designed to be more adapted to the grading of DPM (Direct Part Mark) codes.

- It specifies a new acquisition methodology as well as new quality indicators specifically aimed at DPM symbols.

- Compared to **ISO 15415**, the new acquisition methodology simplifies the acquisition setup at the cost of calibrating of this setup and providing the resulting calibration parameters to the grading process.

Calibrating an ISO 29158 setup

To perform the calibration, you must record the image of an "ideal" (perfect) code on the setup.

To have a successful calibration:

1. Set up the Illumination and acquisition parameters (gain, exposure…) so that the *Mean Light* (ML, the mean value of the pixels of the center of the light cells) is in the range considered valid by the standard (70%~86% of the maximum gray level).

   NOTE: To compute the *Mean Light* with **Open eVision**, grade the symbol with the default calibration parameters and retrieve it from the structure returned by `GetIso29158CalibrationParameters`.

2. Record the ML as `MLCal`.

Compute the other calibration parameters:

3. `RCal`, the calibration reflectance, is the maximum reflectance of the calibration symbol, that is the gray level of the lightest cell.
   - This gray level is computed as the mean gray level on the aperture.
   - The aperture is a circle of a radius 50% or 80% of the cell width and centered on the middle of the cell.

4. `SRCal`, the calibration system response, represents the parameters used to set the brightness of the image at the calibration time.
   - It can be the gain, the exposure, the global illumination due to the lighting angle or even a combination of those.
   - Its nature does not matter as long as the same measure is used for `SRCal` and `SRTarget`.

   NOTE: `SRTarget` is the same measure as `SRCal` but taken at the grading time. Since it is part of the calibration parameters, it should be computed at grading time and passed along the other parameters using `SetIso29158CalibrationParameters` just before calling `Read`.

ISO 291528 Quality Indicators

The cell contrast grade

- The *cell contrast grade* is a measure of the relative difference between the mean reflectance of the brightest and of the darkest modules in the symbol.

▶ If the grade is low, the digitization is more difficult as it is difficult to separate the dark from the light cells.

The cell modulation grade

- The *cell modulation grade* is a measure of the variability and the reliability of cell colors. In essence, it is a combination of the *modulation grade* and *reflectance margin grade* of **ISO 15415**. It is thus a composite grade.

▶ If this grade is low, the digitization is more difficult, as the variations prevent finding an easy way to separate the white from the black. It is also less reliable, as the cells might be too close from the threshold for a clear dark/light classification.

The fixed pattern damage grade

The fixed patterns of the symbols are what characterize the symbols as such.

- The *fixed pattern damage grade* is a measure of the likelihood that the symbol is correctly located and identified in the image.

▶ If this grade is low, the detection and the grid determination are more difficult.
  □ Finder pattern damages can prevent the recognition of a candidate as a valid code.
  □ Timing pattern damages can prevent the correct computation of the grid.
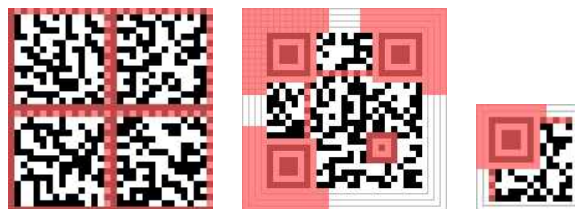
The areas relevant for this grade

The minimum reflectance grade

- The *minimum reflectance grade* represents the difference between the calibration conditions and the acquisition conditions.

▶ It is set to 0.0 if the difference is too big for the grading results to be accurate and 4.0 otherwise.

SEMI T10-0701 for Data Matrix Codes

SEMI T10-0701 is a grading standard intended to provide quality indicators for DPM (Direct Part Mark) Data Matrix codes. It does not provide guidelines for paper-printed ones.

- **SEMI T10-0701** does not provide grades as defined by the other standards. It does not give ratings (A to F or 4.0 to 0.0) but only numerical values.

- The interpretation of these numerical values is left to the application and can depend on the application context.

NOTE: **SEMI T10-0701** is pixel-based, in contrast to the other standards. As such, it can be more suited to computer vision purposes, as the specifications are designed with computer processing in mind.

SEMI T10-0701 quality indicators

The symbol contrast

- The *symbol contrast* measures the relative distinctiveness between the light and the dark parts of the code.

- The closer the *symbol contrast* is to 100%, the more the marks and the spaces are color-separated and the code easier to read.

- A mark is defined as a cell of the data matrix that is modified by the marking process (where the substrate is altered, usually resulting in a darker color) while a space, conversely, is left untouched.

▶ A low contrast score can indicate either a bad printing quality, a bad image contrast or an incorrect code illumination. The detection and the digitization are more difficult as it is difficult to separate the code from the background.
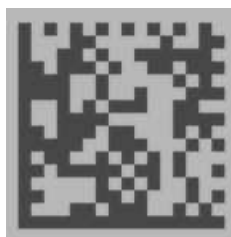
A low contrast data matrix

The symbol contrast SNR

- The *symbol contrast SNR* (Signal to Noise Ratio) measures the relative strength of the noise in the code. You can also see it as the ratio of the useful dynamic (symbol contrast) used to separate white from black.

- The higher this value, the less noise is present in the image of the code.

► If this grade is low, the digitization is more difficult as it is difficult to separate light from dark.

A noisy data matrix

The mark growth

● The *mark growth* is the measure of the relative size of a mark compared to a space.
  □ A value around 50% is ideal, as it means that the marks and the spaces are about the same size.
  □ A value over 50% means that the marks cells are bigger than the spaces, hinting at overmarking.
  □ A value under 50% hints at undermarking.

● The *mark growth* is computed as 2 separate values: the *horizontal mark growth* and the *vertical mark growth*.

► If the *mark growth* deviates from the ideal value (50%), the digitization is more difficult as some cells eat up part of the neighboring cells.

A data matrix showing mark growth

The data matrix mark misplacement

● The *data matrix mark misplacement* is the measure of the relative distance between the ideal position of a mark and its real position in the data matrix. The center of the mark is used as its position.

● The bigger this value, the more the data matrix is potentially deformed.

● The *data matrix mark misplacement* is computed as 2 separate values: The *horizontal data matrix mark misplacement* and the *vertical data matrix mark misplacement*.

▶ This grade is just an indicator of the cell position errors. Unless extreme, it should not hinder the processing.
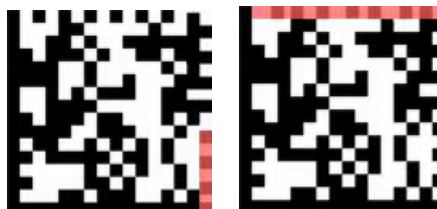
Data matrices showing vertical (left) and horizontal (right) mark misplacement

The cell defects

● If the data matrix can be decoded, the correct placement of the marks and the spaces can be determined, and the *cell defects* value can be computed.

● The *cell defects* value is the measure of the ratio of incorrect pixels (pixels that have the wrong color) to the total number of pixels in the data matrix code.

▶ A high *cell defects* value hints at mark damage, space pollution, difficulties to separate the marks from the spaces or that there is a lot of noise in the image. This can hinder the reading of the code.
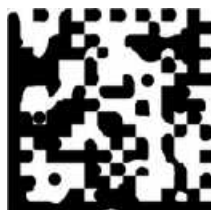
A data matrix showing cell defects

The finder pattern defects

● The *finder pattern defects* is the same kind of measure as *cell defects* but computed only on the cells of the finder pattern of the data matrix code.

▶ A high Finder Pattern Defects value hints at damage to the Finder Pattern, especially if the Cell Defect is otherwise markedly lower. Finder pattern damage might prevent the recognition of a candidate as a valid code.

A data matrix showing finder pattern defects

The unused error correction

- The *unused error correction* is the measure of how much of the error correction capability of the data matrix code was not used.

- The closer this value is to 0, the more error correction was applied to read the code.

▶ A low *unused error correction* can hint at errors in the code itself, difficulties to correctly determine the color of a cell or damage in the marks. It is also a sign that any further degradation of the acquisition conditions will lead to a decoding failure.

The data matrix cell size

- The *data matrix cell size* is the mean size, in pixels, of a cell in the data matrix.

- The *data matrix cell size* is computed as 2 separate values: the *data matrix cell width* (horizontal) and the *data matrix cell height* (vertical).

▶ You can use these values to compute the code anisotropy.

Implementation Specifics and Limitations

General limitations of the grading process

- **Open eVision** is an image processing software, and, as such, its inputs are arrays of pixels. Most standards discussed in this document provide guidelines based not on pixels, but on continuous surfaces presenting a continuous reflectance.

- Most standards also impose constraints on lighting and camera placement that **Open eVision** cannot check or enforce.

- All standards also make the hypothesis that the position of the code is perfectly known. That, in practice, is usually not true, especially for 1D bar codes.

▶ So, there is no guarantee that **Open eVision** returns exactly the same grade as other software tools (assuming there is no error in those), as choices made to alleviate or circumvent the previous points can differ.

- If you face important differences in grades for the same image or are unsure why some of your images are graded the way they are, feel free to contact **Euresys**' technical support.

Implementation specifics

- Given the pretty strict scope and requirements of the grading standards, it can be difficult, if not impossible, to grade some codes when the acquisition conditions are not perfect.

▶ Because of this, **Open eVision** sometimes implements some of the grading processes in a manner slightly different than the canonical one to allow a little more leeway in these conditions.

- These adaptations, however, are made in such a way that if the grading process is made within the requirements of the standard, it yields the required grades as values, as verified by the usage of tools such as conformance calibration test cards.

References

- **ISO/IEC 15415** - Bar code symbol print quality test specification – Two dimensional symbols: https://www.iso.org/standard/54716.html

- **ISO/IEC 15416** - Bar code symbol print quality test specification – Linear symbols: https://www.iso.org/standard/65577.html

- **ISO/IEC 15417** - Code 128 bar code symbology specification: https://www.iso.org/standard/43896.html

- **ISO/IEC 16022** - Data Matrix bar code symbology specification: https://www.iso.org/standard/44230.html

- **ISO/IEC 18004** - QR Code bar code symbology specification: https://www.iso.org/standard/62021.html

- **ISO/IEC 29158** - Direct Part Mark (DPM) Quality Guideline: https://www.iso.org/standard/69411.html

- **SEMI T10-0701** - Test Method for the Assessment of 2D Data Matrix Direct Mark Quality: https://store-us.semi.org/products/t01000-semi-t10-test-method-for-the-assessment-of-2d-data-matrix-direct-mark-quality

- **GS1** General Specifications: https://www.gs1.org/standards/barcodes-epcrfid-id-keys/gs1-general-specifications

5. Using Open eVision Studio

5.1. Selecting your Programming Language

When you start **Open eVision Studio** for the first time, the following welcome screen is displayed:
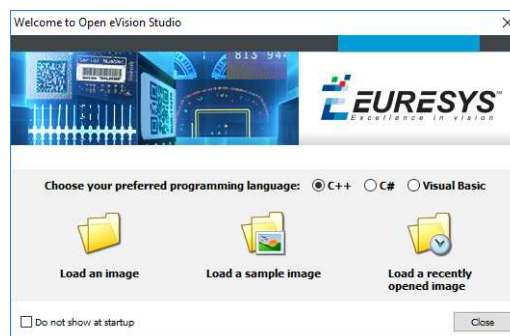
1. Select your programming language.

> ✓ **TIP**
>
> Your selection is saved and your programming language will be automatically selected next time you start **Open eVision Studio**.

> 🗒 **NOTE**
>
> When you change your programming language, any script present in the scripting window is automatically deleted and the window content is reset.

2. Click on one of the `Load` buttons to already load one or several images for later processing.

3. Check the `Do not show at startup` box to hide this welcome screen next time you start **Open eVision Studio**.

> ✓ **TIP**
>
> To access this welcome screen at any time, and change this setting, go to the `Help` > `Welcome Screen` menu.

5.2. Navigating the Interface
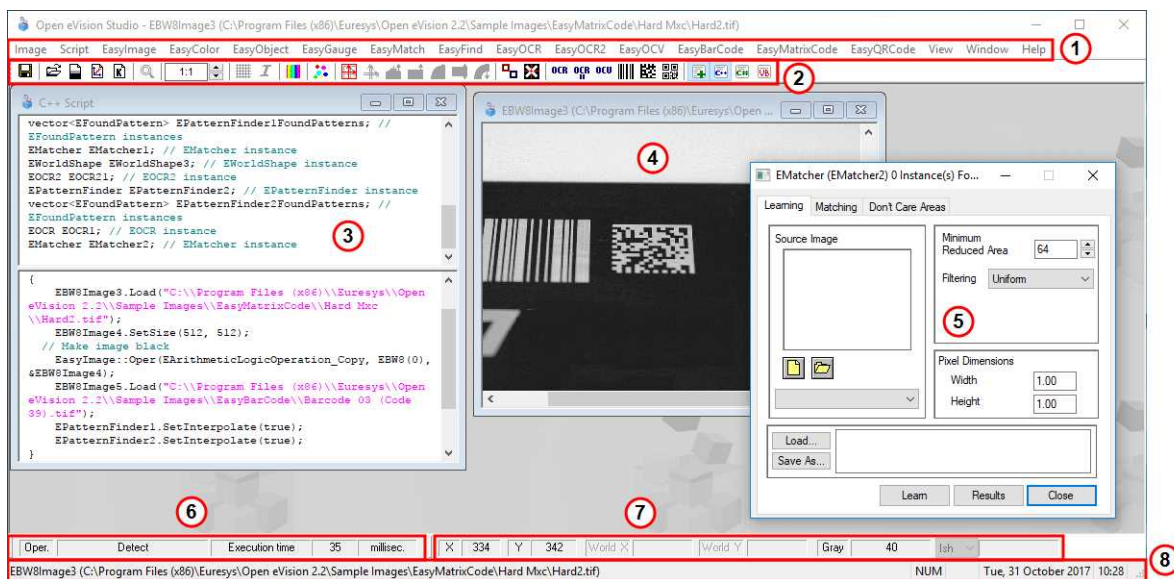
Open eVision Studio graphical user interface (GUI) is organized as follows:

1. The *main menu bar* gives you access to the functions and tools of all libraries.

> ✅ **TIP**
>
> **Open eVision Studio** does not require any license and allows you to test all libraries. Of course, if you copy code from **Open eVision Studio** in your own application but you do not have the required license, you will receive a "missing license" error at run-time.

2. The *main toolbar* gives you quick access to main **Open eVision** objects such as images, shapes, gauges, bar codes, matrix codes...

3. The *script window* displays the code, in the programming language you selected, corresponding to the actions you perform in **Open eVision Studio**. You can save or copy this code in your own application at any time.

4. The *image windows* display the open images that you can process using the libraries and tools.

5. The *tool windows* enable you to easily configure all the available tools. The corresponding settings are automatically added in the script window for easy reuse.

> ✅ **TIP**
>
> Most tool windows are floating and you can easily move them outside the **Open eVision Studio** main window to make better use of your screen size.

6. The *execution time bar* displays the precise time taken for the execution of the selected functions (measured in milliseconds or microseconds) on your computer. This accurate measurement helps you to evaluate the performance of your application.

7. The *color toolbar* displays current information such as the X and Y coordinates of the cursor on an image and the corresponding pixel value.

8. The *status bar* displays general information about the application such as the active image file path...

5.3. Running Tools on Images

Step 1: Selecting a Tool

When you use **Open eVision Studio**, the first step is to select the library and the tool you want to use on your image.

To do so:

1. In the main menu bar, click on the library you want to use.

2. Click on the tool you want to use.

> ✓ **TIP**
> All libraries (except EasyImage, EasyColor and EasyGauge) expose only one tool named `New Xxx Tool`. Some of these libraries also expose additional functions.

3. In the dialog box, enter a `Variable name` for the variable that is automatically created and that will contain the result of the processing.
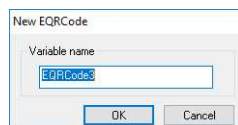
Example of variable creation dialog box for EasyQRCode

4. Click `OK`.

The selected tool dialog box opens.
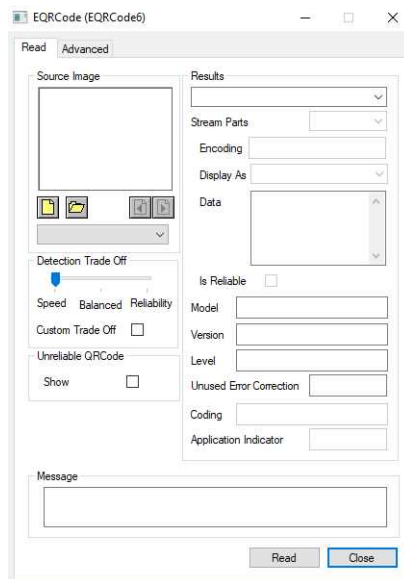
Example of variable creation dialog box for EasyQRCode

The next step is "Step 2: Opening an Image" on page 156.

Step 2: Opening an Image

Once you have selected your library and your tool, you need to open an image to apply this tool.

In the `Source Image` area of the selected tool dialog box:

1. Open an image:

   □ Click on the `Open an Image` button and select one or several (using SHIFT and CTRL) images on your computer.

   □ Or select one of the images (or one of the ROIs, if any) already open in the drop-down list.

> **NOTE**
> You can select only images with an appropriate file format (JPG, PNG, TIFF or BMP) and in 8- and/or 24-bit depending on the library.
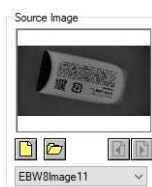
2. If you selected several images, activate one with the `Load Previous` or `Load Next` buttons.

The tool is automatically applied on any loaded image and, at this stage, the result is displayed based on the tool default settings.

The next step is "Step 3: Managing ROIs" on page 157.

Step 3: Managing ROIs

In some cases, most often to decrease the processing time or to single-out the object you want to read, you do not want to process the whole image but only one or several well defined rectangular parts of this image, or ROIs (Regions Of Interest).

> ✓ **TIP**
>
> In **Open eVision**, ROIs are attached to an image and exist only as long as the parent image is available.

Creating a ROI

1. Open the image:
   - □ If the image is already open, activate the corresponding image window.
   - □ If the image is not open yet, go to the main menu: `Image` > `Open...` to open one.

2. To create an ROI, go to the main menu: `Image` > `ROI Management...`.

The `ROI Management` window is displayed as illustrated below.
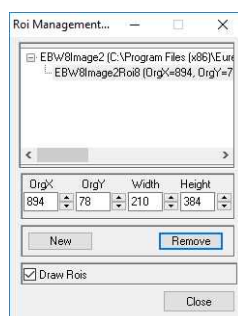
3. Select the image in the tree.

4. Click on the `New` button.

5. In the dialog box, enter a `Variable name` for the new ROI.

The ROI is represented as a color rectangle on your image as illustrated below.
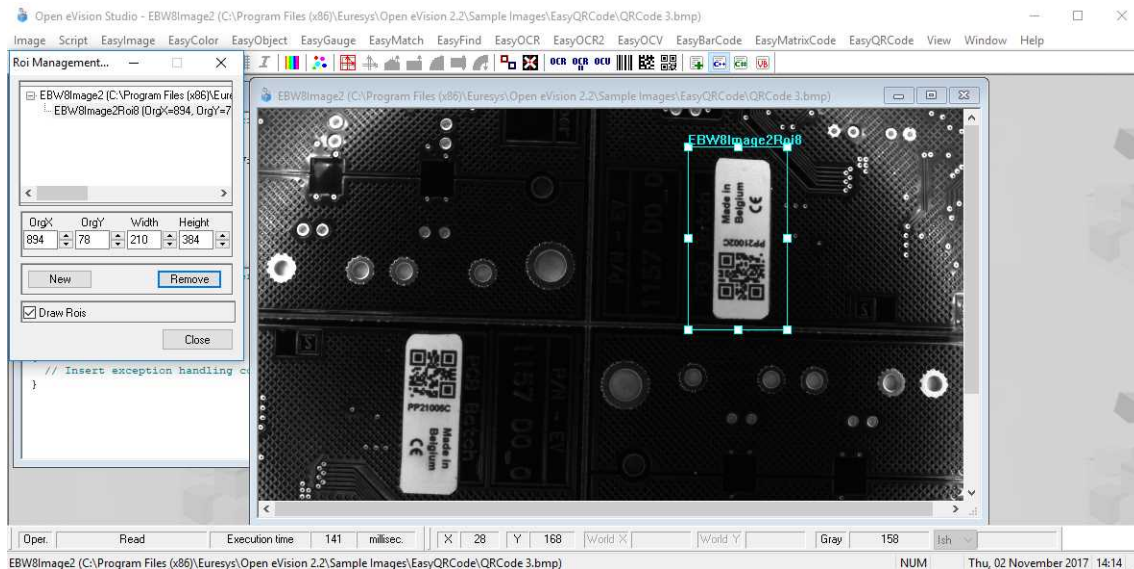
6. Drag the ROI corner and side handles to move it to the required position.

7. Click on the Close button to close the ROI Management window .

The next step is "Step 4: Configuring the Tool" on page 159.

Managing ROIs

You can add, change and remove ROIs.

> ✅ **TIP**
>
> An image can have several ROIs. Each ROI can be attached directly to the image (meaning that its position is relative to the image) or to another ROI (meaning that its position is relative to this 'parent' ROI).

1. To manage ROIs, go to the main menu: Image > ROI Management....

The ROI Management window is displayed with the ROI relation tree as illustrated below.

If the Draw Rois box is checked, all ROIs are displayed on the image with a different color.
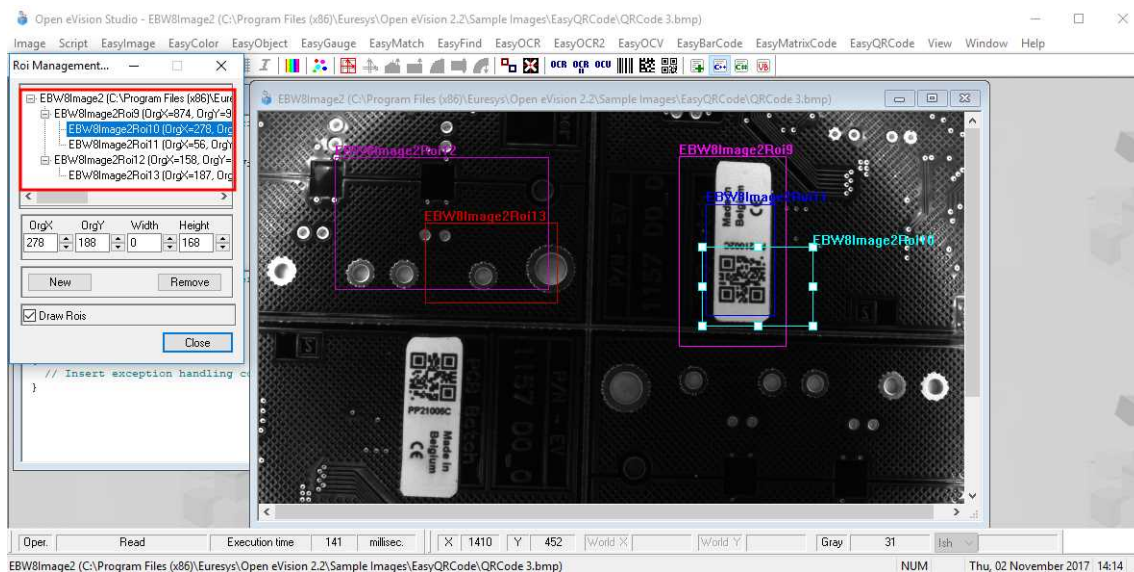
2. Select an ROI in the ROI relation tree.

3. Drag the ROI corner and side handles to change the position and size of the selected ROI (as well as the position of all ROIs attached to it if any).

4. Click on the New button to add a new ROI attached to the selected ROI.

> **TIP**
> Select the image at the top of the ROI relation tree to attach the ROI directly to the image.

5. Click on the Remove button to delete the selected ROI (and all ROIs attached to it if any).

6. Click on the Close button to close the `ROI Management` window.

Step 4: Configuring the Tool

Once your image, including its ROIs if you created some, is ready, you need to configure your tool.

In the tool window:

1. Open the various tabs.

> **TIP**
> When you create a new tool, all parameters are set with their default value.
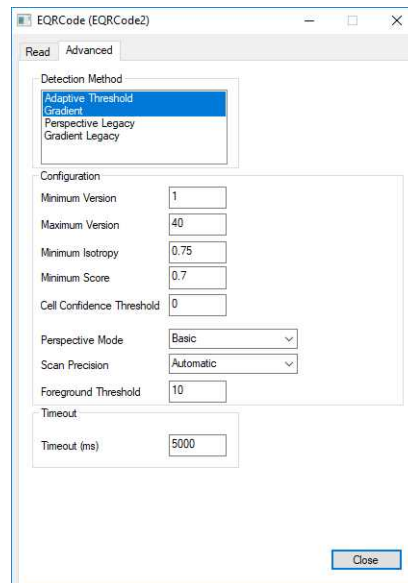
Example of the parameter tab of an EasyQRCode tool

2. In each tab, set the value of the parameters as desired.

Please refer to the "Functional Guide" and to the "Reference Manual" for detailed information about the parameters, their function and their default value.

For specific actions such as learning or using gauges, please refer to the "Functional Guide".

3. Run the tool and analyze the results as described in the next step "Step 5: Running the Tool and Checking Execution Time" on page 160.

Step 5: Running the Tool and Checking Execution Time

Once your tool parameters are set, run your tool and, if desired, check the execution time on your computer.

In the tool window:

1. Click on the `Read`, `Detect`, `Results` or `Execute` button (depending on the library function), to run the tool on the selected image.

2. Check the results on the image and in the Results field or area as illustrated below.
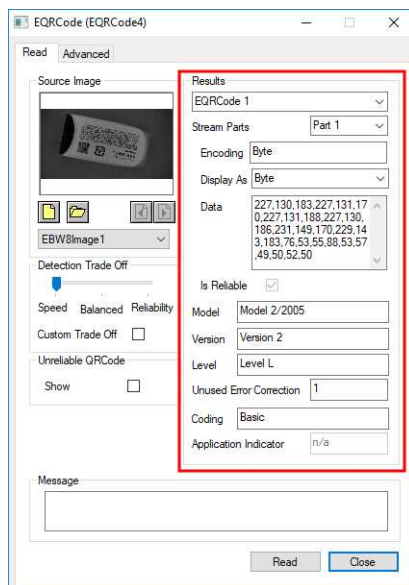
Example of results after reading a QRCode

3. If you do not have the expected results:
   □ Try to change your parameters (start with default values then change one parameter at a time).
   □ If your image is not good enough, try to enhance it as described in .

4. Check the execution time in the execution time bar at the bottom left of the main **Open eVision Studio** window.
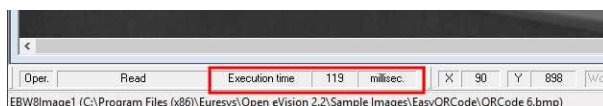
The execution time
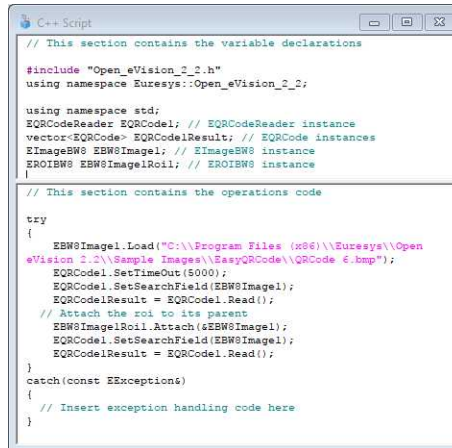
> ✅ **TIP**
>
> The execution time is the actual time that the processing took as measured on your computer. It depends your computer processor, memory, operating system... and, of course, on the processor load at the time of execution. Thus this execution time slightly varies from execution to execution.

5. To get a more representative execution time, click on the `Read`, `Detect`, `Results` or `Execute` button several times and calculate the mean execution time.

6. If your application requires that you reduce the execution time, try:
   □ To change the tool parameters,
   □ To add one or several ROIs on your image,
   □ To enhance your image.

The next step is .

Step 6: Using the Generated Code

By default, **Open eVision Studio** translates all the operations you perform in the interface into code in the language you selected as illustrated below.

Once your tool results suit you, you can save or copy this generated code to use it in your own application.

Copy and paste the code in your application

In the script window:

1. Select the code section you want to copy.

2. Right click on this code and click Copy in the menu.

3. Go to your development environment tool and paste the code in place.

Save the code

1. Go to the **Script** menu.

2. Click on Save Script As....

3. Enter a file name and path to save the code as a text file.

Manage the generated code

In the **Script** menu, you can:

☐ Select the programming language (please note that if you change the language, the script window content is automatically deleted).

☐ Activate or deactivate the Script Code Generation. Deactivate this option if you want to perform some operations without saving them as code.

5.4. Pre-Processing and Saving Images

When should you pre-process your images?

Of course, the best situation is to set up your image acquisition system to have good and easy to process images so the **Open eVision** tools run smoothly and efficiently.

If this is not possible or easy to achieve, you can pre-process your images or your ROIs to enhance and prepare them for the **Open eVision** tool you want to run.

Using the various available functions, you can adjust the gain and offset of your image, apply a convolution, threshold, scale, rotate and white balance your image, enhance contours... using EasyImage and EasyColor functions.

Pre-processing images

The difference between pre-processing an image and running tools is that the pre-processing generates a new image while the tools mainly extract and retrieve information from the image without changing it.

To pre-process an image or an ROI:

1. In the main menu bar, click on the library you want to use (EasyImage or EasyColor).

2. Click on the function you want to use.

Most function dialog boxes are similar to the one illustrated below with 2 image selection areas and a parameter setting area.
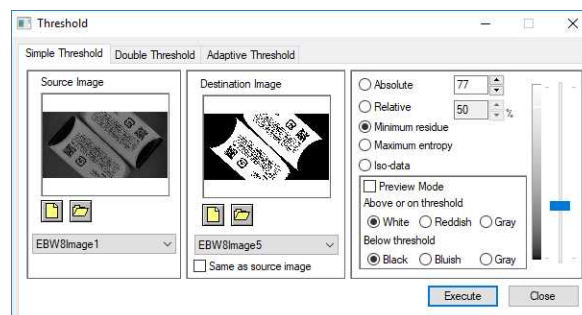
Example of a pre-processing dialog box (Threshold with EasyImage)

3. If there are multiple versions for your selected function, open the corresponding tab.

4. In the `Source Image` area, open the source image (as described in "Step 2: Opening an Image" on page 156).

5. In the `Destination Image` area, open or create a new destination image.

6. Set your parameters.

7. Click on the `Execute` button.

The pre-processed image is available in the destination image as illustrated below.
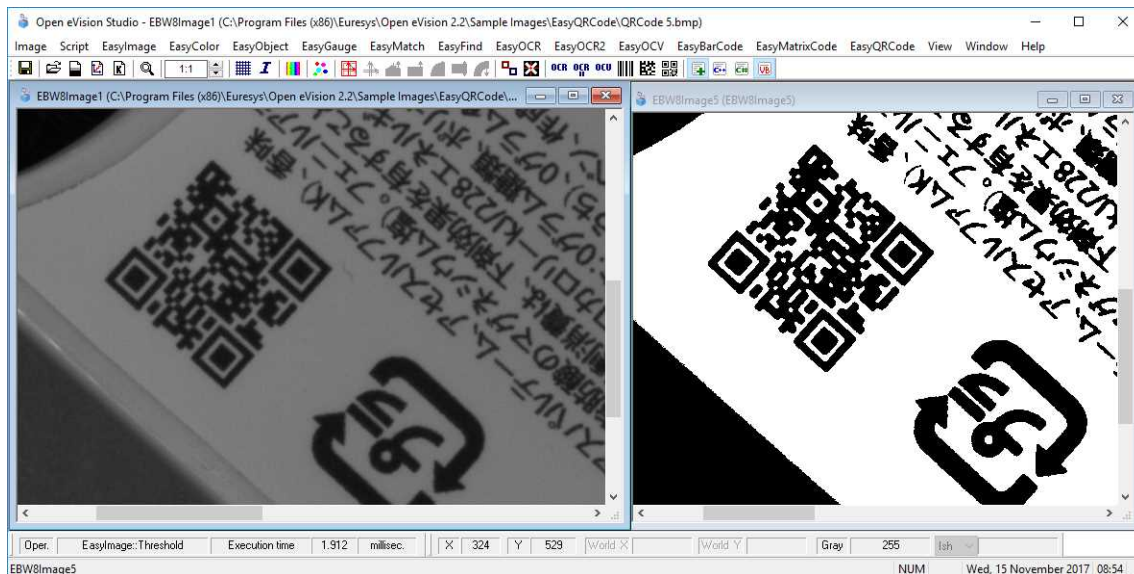
Source and destinations images (Threshold with EasyImage)

8. If you want to use the destination image outside of **Open eVision Studio**, save it as described below.

Saving an image

1. Click on the image you want to save to makes its window active.

2. To open the save menu either:
   - ☐ Right-click in the image
   - ☐ Or open the main menu > Image

3. Click on Save as....

4. Select the file format (JPEG, JPEG2000, PNG, TIFF or Bitmap).

5. Enter a name and select a path.

6. Click on the Save button.

6. Tutorials

6.1. EasyBarCode

Reading Bar Codes Automatically

"Reading a Bar Code" on page 178

Objective

Following this tutorial, you will learn how to perform automatic reading of multiple bar codes.

You'll need first to load multiple source images (step 1). The reading is then automatically performed on each image (step 2).

Each bar code is automatically detected and decoded

Step 1: Load the source images

1. From the main menu, click EasyBarCode, then New BarCode Tool.

2. Keep the default variable name, and click OK.

3. In the AutoRead tab, click the Open icon of the Source Image area, and load the image files
    EasyBarCode\Barcode 01.tif to Barcode 10.tif. Use the shift key to select multiple files.

4. Keep the default variable name, and click OK. The last image appears.

Step 2: Read the bar codes automatically

1. The bar code is automatically detected and decoded. The graphic result appears on the
    image, while the data content and the corresponding symbology are displayed in the
    Decoded Symbology area. It is not necessary to click Read once a new image appears.
    However, clicking Read will insert the corresponding code into the script window.

2. In the Results tab, find more information about the bar code. As a bar code might have a
    meaning under different symbologies, all possible contents are listed by decreasing
    likeliness.

3. In the AutoRead tab, click the Load Next and Load Previous icons to browse through images 01
    to 06.

The image files appear, and each bar code is automatically detected and decoded. The bar code
properties are updated.

4. To decode the remaining bar codes, we have to enable the additional symbologies.

5. In the Symbologies tab, click the Toggle All button of the Additional area.

6. In the AutoRead tab, click the Load Next and Load Previous icons and browse the remaining
    images.

6.2. EasyMatrixCode

Reading Data Matrix Codes Automatically

"Reading a Data Matrix Code" on page 180

Objective

Following this tutorial, you will learn how to use EasyMatrixCode to detect and decode
automatically Data Matrix codes in multiple files.

You'll need first to load multiple source images (step 1). The reading is then automatically
performed on each image (step 2). You can also grade the printing quality of each matrix code
(step 3).
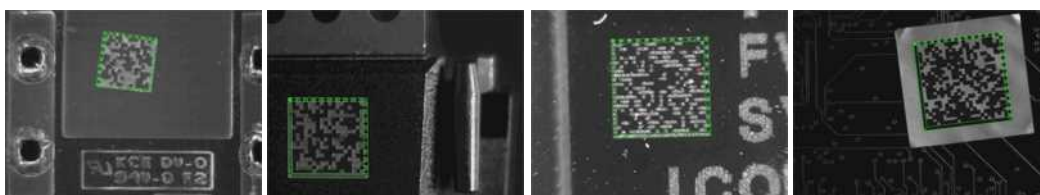
Each Data Matrix code is automatically detected and decoded

Step 1: Load the source images

1. From the main menu, click `EasyMatrixCode`, then `New MatrixCode Tool`.

2. Keep the default variable name for the new matrix code reader, and click `OK`.

3. In the `Read` tab, click the `Open` icon of the Source Image area, and load the image files `EasyMatrixCode\AutoRead\AutoRead 01.tif` to `AutoRead 04.tif`. Use the shift key to select multiple files.

4. Keep the default variable name for the new image, and click OK. The last image appears.

Step 2: Read the Data Matrix codes automatically

1. The Data Matrix code is automatically detected and decoded. The matrix code reference corner is highlighted with a bold cross mark. It is not necessary to click `Read` once a new image appears. However, clicking `Read` will insert the corresponding code into the script window.

2. In the `Results` area, find more information about the matrix code, such as the decoded string.

3. In the `Read` tab, click the `Load Next` and `Load Previous` icons. The image files appear, and each Data Matrix code is automatically detected and decoded. The matrix code properties are updated. If no Data Matrix code could been located in the image, an error message is displayed in the Message field.

Step 3: Grade Data Matrix code printing quality

1. In the `Print Quality` tab, select the `Compute Grading` check-box.

2. Click `Apply`.

For each printing quality parameter, the corresponding value and its grade equivalent appear. An A grade means a good quality, while an F grade indicates a poor one.

Learning a Data Matrix Code and Creating an EasyMatrixCode Model File

"Learning a Data Matrix Code" on page 180

Objective

Following this tutorial, you will learn how to use EasyMatrixCode to learn a Data Matrix code, and save it as an EasyMatrixCode model file.

You'll need first to load a source image (step 1), and learn the matrix code (step 2). Then you'll save the learned matrix code as a new model file (step 3). You can also add new learned matrix codes to an existing model if needed (step 4).
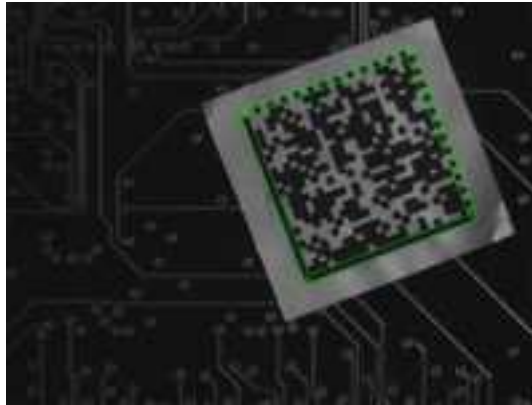
The Data Matrix code has been learned

Step 1: Load the source image

1. From the main menu, click `EasyMatrixCode`, then `New MatrixCode Tool`.

2. Keep the default variable name for the new matrix code reader object, and click `OK`.

3. In the `Learn` tab, click the `Open` icon of the Source Image area, and load the image file `EasyMatrixCode\Label\Label 4.tif`.

4. Keep the default variable name for the new image object, and click `OK`.

Step 2: Learn the Data Matrix code

- In the `Learn` tab, click `Learn`.

The Data Matrix code is detected and decoded without error.

The graphical result appears on the image.

The properties of the learned matrix code are updated in the dialog box.

Step 3: Save the model file

1. In the `Learn` tab, click the `Save As...` button.

2. Type a file name for the new EasyMatrixCode model file. Its extension will be .mx2.

3. Click `Save`.

Step 4: Learning more Data Matrix codes

1. In the `Read` tab, click the `Open` icon of the source image area, and load the image file `EasyMatrixCode\PCB Code\PCB Code 3.jpg`.

2. Keep the default variable name for the new image object, and click `OK`.

3. An error message is displayed in the Message area of the `Read` tab. The matrix code can not be read, since the reader uses the model from the "Label 4" image. You need to learn the "PCB Code 3" matrix code, and add it to the model.

4. In the `Learn` tab, click `Learn More`. The Data Matrix code is detected and decoded without error. The graphical result appears on the image. The properties of the learned matrix code are updated in the dialog box.

5. Using `Learn More` rather than Learn involves that the "Label 4" model is not replaced by the "PCB Code 3" model, but both are now included in the same model. In the Read tab, the "PCB Code 3" matrix code is correctly read. Select the "Label 4" image in the drop-down list of the source image area. The "Label 4" matrix code is still read without error, which means that both learned matrix codes have been kept.

6. Finally, save the model again (refer to step 3). The new matrix code has been added.

6.3. EasyOCR

Learning Characters and Creating an EasyOCR Font

Objective

Following this tutorial, you will learn how to use EasyOCR to learn new characters and save them in an EasyOCR font.

You'll need first to load a source image (step 1). Then you'll set the segmentation parameters to isolate each character (step 2). Each character will have to be learnt (step 3), and finally you'll save all the learnt characters as a font file (step 4). You can also add new characters to an existing font if needed (step 5).
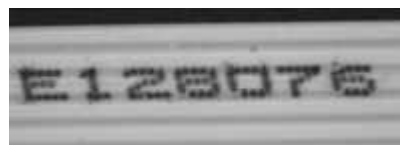
Source image

The image is segmented so that all the characters are detected
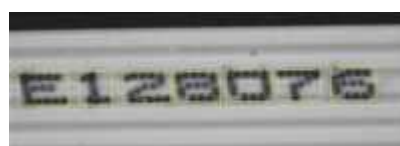
All the characters have been learn

Step 1: Load the source image

1. From the main menu, click `EasyOCR`, then `New OCR Tool`.

2. Keep the default variable name for the new OCR object, and click `OK`.

3. In the `Source Image` tab, click the `Open` icon of the Source Image area, and load the image file
    EasyOCR\FlatCable\FlatCable1.tif.

4. Keep the default variable name for the new image object, and click `OK`.

Step 2: Set segmentation parameters

1. Select the `Segmentation Parameters` tab, and move the red frame in the image above a
    character.

2. Tune each property to get a green bounding box around each character:
    □ `threshold value` = 113
    □ `characters` color = Black on White
    □ `min width` = 36
    □ `min height` = 31
    □ `spacing` = 4
    □ `max width` = 98
    □ `max height` = 72
    □ `noise area` = 9

Step 3: Learn new characters

1. Select the `Learn` tab, and click the character E in the image. You are then prompted to
    identify the character along with its class. Enter E in the character field, and select the
    '`EOcrClass_Uppercase`' class. Click `OK`. Whenever a character has been added to the current
    font, its bounding box turns yellow.

2. Click the character 1 in the image. Enter 1 in the character field, and select the '`EOcrClass_
    Digit`' class. Click `OK`.

3. Proceed with remaining characters.

Step 4: Save the EasyOCR font

● In the `Font File` tab, click the `Save As...` button. Type a file name for the new EasyOCR font
   file. Its extension will be .ocr. Finally, click `Save`.

Step 5: Add characters to an existing font

1. In the Source Image tab, click the `Open` icon of the Source Image area, and load the image file
    EasyOCR\FlatCable\FlatCable2.tif.

2. Keep the default variable name for the new image object, and click `OK`.

3. In the `Recognition` tab, click `Execute`. Characters 2 and 8 are read correctly, but A, W and G
    are not (`low confidence score`). They don't belong to the font.

4. Select the `Learn` tab, and learn the characters A, W, and G (refer to step 3).

5. Then save the font again (refer to step 4). The new characters have been added.

Recognizing Characters According to a Font

"Recognizing Characters According to a Font" on page 171

Objective

Following this tutorial, you will learn how to use EasyOCR to recognize characters, regarding to a specific font.

You'll need first to load a source image (step 1), and an EasyOCR font file (step 2). Then you'll perform the characters recognition (step 3).
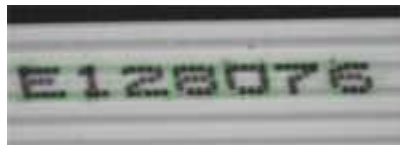
Characters matching the font are automatically detected

Results after explicit recognition

Step 1: Load the source image

1. From the main menu, click `EasyOCR`, then `New OCR Tool`.

2. Keep the default variable name for the new OCR object, and click `OK`.

3. In the Source Image tab, click the `Open` icon of the Source Image area, and load the image file `EasyOCR\FlatCable\FlatCable1.tif`.

4. Keep the default variable name for the new image object, and click `OK`.

Step 2: Load the font file

- In the `Font File` tab, click `Load`, and select the font file `EasyOCR\FlatCable\FlatCable.ocr`. In the image, the detected characters are highlighted in green.

euresys

Step 3: Recognize the characters

- In the `Recognition` tab, click `Execute` to trigger the recognition of the detected characters. The recognized characters appear in the Recognition results area. Further information about each character can be found in the table.

7. Code Snippets

7.1. Basic Types

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= new EImageBW8();
EImageBW8 dstImage= new EImageBW8();

// 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);
```

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= new EImageBW8();

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

//Pointer to the third-party image buffer
IntPtr imgPtr = bufferPointer;

// ...

// Link the Open eVision image to the third-party image
// Assuming the corresponding  buffer is aligned on 4 bytes
srcImage.SetImagePtr(sizeX, sizeY, imgPtr);
```

Retrieving Pixel Values

Functional Guide | Reference: GetImagePtr

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

using System.RunTime.InteropServices;

IntPtr pixAddr;
byte pix;

//...

for(int y = 0; y < height; ++y)
{
    pixAddr = bw8Image.GetImagePtr(0,y);
    for(int x = 0; x < width; ++x)
        pix = Marshal.ReadByte(pixAddr,x);
}
```

Importing Bitmap from the Resources

Functional Guide | Reference: SetImagePtr

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

// Use this function to fasten the copy of the bitmap
[DllImport("kernel32.dll", EntryPoint = "CopyMemory", SetLastError = false)]
public static extern void CopyMemory(IntPtr dest, IntPtr src, uint count);

// Get the bitmap
Bitmap bitmap = new Bitmap(WindowsFormsApp1.Properties.Resources.Image1);

int width = bitmap.Width;
int height = bitmap.Height;

BitmapData bmd = bitmap.LockBits(new Rectangle(0, 0, width, height), ImageLockMode.ReadOnly,
bitmap.PixelFormat);

EImageC24 image = new EImageC24(width, height);


// Number of bytes per row to copy
uint strideWidth = (uint)(3 * width);

for (int y = 0; y < height; ++y)
{
  CopyMemory(image.GetImagePtr(0, y), bmd.Scan0 + y * bmd.Stride, strideWidth);
}

bitmap.UnlockBits(bmd);
bitmap.Dispose();
```

```
// Process the image

image.Dispose();
```

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= new EImageBW8();

// ROI constructor
EROIBW8 myROI= new EROIBW8();

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

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

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= new EBW8Vector();
EBW8 bw8 = new EBW8();

// Clear the vector
ramp.Empty();

// Fill the vector with increasing values
for(int i= 0; i < 128; i++)
{
    bw8.Value = (byte)i;
    ramp.AddElement(bw8);
}

// Retrieve the 10th element value
EBW8 value = ramp.GetElement(9);
```

Exception Management

Functional Guide | Reference: GetPixel, What

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

try
{
    // Image constructor
    EImageC24 srcImage= new EImageC24();

    // ...

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

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

7.2. EasyBarCode

Reading a Bar Code

Functional Guide | Reference: Read

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

// Image constructor
EImageBW8 srcImage= new EImageBW8();

// Bar code reader constructor
EBarCode reader= new EBarCode();

// String for the decoded bar code
string result;

// ...

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

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= new EImageBW8();

// Bar code reader constructor
EBarCode reader= new EBarCode();

// String for the decoded bar code
string result;

// ...

// Disable all standard symbologies
reader.StandardSymbologies= 0;

// Enable the Code32 symbology only
reader.AdditionalSymbologies= (int)ESymbologies.Code32;

// Enable checksum verification
reader.VerifyChecksum= true;

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

```
// Retrieve the number of symbologies for
// which the decoding process was successful
uint numDecoded = reader.NumDecodedSymbologies;

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

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= new EImageBW8();

// Bar code reader constructor
EBarCode reader= new EBarCode();

// String for the decoded bar code
string result;

// ...

// Disable automatic bar code detection
reader.KnownLocation = 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);
```

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 = new EImageBW8();
// Mail bar code reader constructor
EMailBarcodeReader reader = new EMailBarcodeReader();
// Select expected symbologies and orientations (optional)
reader.ExpectedSymbologies = ...;
reader.ExpectedOrientations = ...;
```

```
// ...
// Read
EMailBarcode [] codes = reader.Read(srcImage);
// Retrieve the data included in found mail barcodes
for (int index= 0; index < codes.Length; index++)
{
    string text = codes[index].Text;
    EStringPair [] components = codes[index]. ComponentStrings;
}
```

7.3. EasyMatrixCode

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= new EImageBW8();

// Matrix code reader constructor
EMatrixCodeReader reader= new EMatrixCodeReader();

// Matrix code constructor
EMatrixCode mxCode= new EMatrixCode();

// String for the decoded information
string result;

// ...

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

// Retrieve the decoded string
result = mxCode.DecodedString;
```

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= new EImageBW8();
EImageBW8 srcImage= new EImageBW8();
```

```
// Matrix code reader constructor
EMatrixCodeReader reader= new EMatrixCodeReader();

// Matrix code constructor
EMatrixCode mxCode= new EMatrixCode();

// String for the decoded information
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.DecodedString;
```

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= new EImageBW8();

// Matrix code reader constructor
EMatrixCodeReader reader= new EMatrixCodeReader();

// Matrix code constructor
EMatrixCode mxCode= new EMatrixCode();

// String for the decoded information
string result;

// ...

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

// Add the 15x15 and 17x17 logical sizes
reader.SearchParams.AddLogicalSize(ELogicalSize._15x15);
reader.SearchParams.AddLogicalSize(ELogicalSize._17x17);

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

// Add the ECC050 family
reader.SearchParams.AddFamily(EFamily.ECC050);
```

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

// Retrieve the decoded string
result = mxCode.DecodedString;
```

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= new EImageBW8();

// Matrix code reader constructor
EMatrixCodeReader reader= new EMatrixCodeReader();

// Matrix code constructor
EMatrixCode mxCode= new EMatrixCode();

// ...

// Enable grading computation
reader.ComputeGrading= true;

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

// Retrieve the print quality grading
int axialNonUniformityGrade= mxCode.AxialNonUniformityGrade;
int contrastGrade= mxCode.ContrastGrade;
int printGrowthGrade= mxCode.PrintGrowthGrade;
int unusedErrorCorrectionGrade= mxCode.UnusedErrorCorrectionGrade;
```

7.4. EasyMatrixCode2

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.                      //
/////////////////////////////////////////////////////////
using EMC2 = Euresys.Open_eVision_x_x.EasyMatrixCode2;
// Load an image
EImageBW8 image = new EImageBW8();
image.Load("image.bmp");
// Prepare a matrix code reader
```

```
EMC2.EMatrixCodeReader reader = new EMC2.EMatrixCodeReader();
// Let the reader know that there are no more than 3 codes in the image
reader.MaxNumCodes = 3;
// Read the source image
reader.Read(image);
// Retrieve the detected codes
EMC2.EMatrixCode[] codes = reader.ReadResults;
// Retrieve the decoded string for the first code
string result = codes[0].DecodedString;
```

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.     //
/////////////////////////////////////////////////////////

using EMC2 = Euresys.Open_eVision_x_x.EasyMatrixCode2;

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

// Prepare a matrix code reader
EMC2.EMatrixCodeReader reader = new EMC2.EMatrixCodeReader();

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

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

// Retrieve the detected codes
EMC2.EMatrixCode[] codes = reader.ReadResults;

// Retrieve the decoded string of the first code
string result = codes[0].DecodedString;
```

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 = new EImageBW8();
ERectangleRegion gridRegion = new ERectangleRegion();
EMC2.EMatrixCodeReader reader = new EMC2.EMatrixCodeReader();

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

```
EMC2.EMatrixCode[] codesMiddleCell = grid.GetResults(1, 2);

// cleanup
foreach (EMC2.EMatrixCode code in codesMiddleCell)
{
   code.Dispose();
}
grid.Dispose();
reader.Dispose();
gridRegion.Dispose();
srcImage.Dispose();
```

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.                 //
////////////////////////////////////////////////////////
using EMC2 = Euresys.Open_eVision_x_x.EasyMatrixCode2;
// Load an image
EImageBW8 image = new EImageBW8();
image.Load("image.bmp");
// Prepare a matrix code reader
EMC2.EMatrixCodeReader reader = new EMC2.EMatrixCodeReader();
// Tell the reader to compute grades for the read codes
reader.ComputeGrading = true;
// Read the codes in this image
reader.Read(image);
// Retrieve the detected codes
EMC2.EMatrixCode[] codes = reader.ReadResults;
// Retrieve the SemiT10 grades of the first code
EMatrixCodeSemiT10GradingParameters semiT10Grades = codes[0].SemiT10GradingParameters;
// Retrieve specific grade values
float cellDefects = semiT10Grades.CellDefects;
float symbolContrast = semiT10Grades.SymbolContrast;
float unusedErrorCorrection = semiT10Grades.UnusedErrorCorrection;
```

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.               //
//////////////////////////////////////////////////////////////////////
using System.Threading;
using EMC2 = Euresys.Open_eVision_x_x.EasyMatrixCode2;
// create a subroutine that reads the codes from an image
void Read(ref EImageBW8 image, ref EMC2.EMatrixCodeReader reader, ref EMC2.EMatrixCode[] codes, ref 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;
}
void main()
{
  // Prepare three images
  EImageBW8 img1 = new EImageBW8();
  EImageBW8 img2 = new EImageBW8();
  EImageBW8 img3 = new EImageBW8();

  // Prepare three matrix code readers
  EMC2.EMatrixCodeReader reader1 = new EMC2.EMatrixCodeReader();
  EMC2.EMatrixCodeReader reader2 = new EMC2.EMatrixCodeReader();
  EMC2.EMatrixCodeReader reader3 = new EMC2.EMatrixCodeReader();

  // Prepare three vectors of matrix code instances
  EMC2.EMatrixCode[] codes1 = null;
  EMC2.EMatrixCode[] codes2 = null;
  EMC2.EMatrixCode[] codes3 = null;

   // Prepare three Booleans to track the thread progress
  bool finished1 = false;
  bool finished2 = false;
  bool finished3 = false;

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

  // Launch three threads to read the codes in each image
  Thread thr1 = new Thread(() => Read(ref img1, ref reader1, ref codes1, ref finished1));
  Thread thr2 = new Thread(() => Read(ref img2, ref reader2, ref codes2, ref finished2));
  Thread thr3 = new Thread(() => Read(ref img3, ref reader3, ref codes3, ref finished3));

  // Start the threads, they will run in the background.
  thr1.Join();
  thr1.Join();
  thr1.Join();

  // Wait until one of the threads has finished
  while (!(finished1 || finished2 || finished3))
    Thread.Sleep(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();
}
```

7.5. EasyQRCode

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= new EImageBW8();

// QR code reader constructor
EQRCodeReader reader= new EQRCodeReader ();

// ...

// Read
EQRCode [] qrCodes = reader.Read(srcImage);
```

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 = new EImageBW8();
ERectangleRegion gridRegion = new ERectangleRegion();
EQRCodeReader reader = new EQRCodeReader();

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

EQRCode[] codesMiddleCell = grid.GetResults(1, 2);

// cleanup
foreach (EQRCode code in codesMiddleCell)
{
  code.Dispose();
}
grid.Dispose();
reader.Dispose();
gridRegion.Dispose();
srcImage.Dispose();
```

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 = new EQRCodeReader();;

// Tell the reader to compute grades for the codes read
reader.ComputeGrading = true;

// Read the codes in this image
EQRCode[] codes = reader.Read(image);

// Retrieve the detected codes
// Retrieve the ISO15415 grades of the first code
EQRCodeIso15415GradingParameters grades = codes[0].Iso15415GradingParameters;

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

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= new EImageBW8();

// QR code reader constructor
EQRCodeReader reader= new EQRCodeReader ();

// ...

// Read
EQRCode [] qrCodes = reader.Read(srcImage);

// Retrieve version, model and position information
// of the first QR code found, if one was found
if (qrCodes.Length > 0)
{
    uint version = qrCodes[0].Version;
    EQRCodeModel model = qrCodes[0].Model;
    EQRCodeGeometry geometry = qrCodes[0].Geometry;
}
```

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= new EImageBW8();

// QR code reader constructor
EQRCodeReader reader= new EQRCodeReader ();

// ...

// Set the search parameters
reader.MaximumVersion = 7;
reader.MinimumIsotropy = 0.9f;

// Set the searched models
reader.SearchedModels = new EQRCodeModel[] {EQRCodeModel.Model2};

// Read
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);
```

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 = new EImageBW8();

// QR code reader constructor
EQRCodeReader reader = new EQRCodeReader();

// ...

// Read
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);
```

Retrieving the Decoded String (Safe)

Functional Guide | Reference: Read, GetDecodedString

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

```
// Image constructor
EImageBW8 srcImage = new EImageBW8();

// QR code reader constructor
EQRCodeReader reader = new EQRCodeReader();

// ...

// Read
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);
}
```

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 = new EImageBW8();

// QR code reader constructor
EQRCodeReader reader = new EQRCodeReader();

// ...

// Read
EQRCode [] qrCodes = reader.Read(srcImage);

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

// Retrieve the coding mode and the raw bit stream of the first QR code found
EQRCodeCodingMode codingMode = stream.CodingMode;
byte [] bitstream = stream.RawBitstream;

// Retrieve the encoding and the decoded data of each part of the first QR code found
EQRCodeDecodedStreamPart [] parts = stream.DecodedStreamParts;
for (uint i = 0; i < parts.Length; ++i)
{
```

```
  // Retrieve encoding
  EQRCodeEncoding encoding = parts[i].Encoding;

  // Retrieve the decoded data
  byte [] decodedData = parts[i].DecodedData;

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

7.6. EasyOCR

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= new EImageBW8();

// EOCR constructor
EOCR ocr= new EOCR();

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

// ...

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

// Adjust the segmentation parameters
ocr.TextColor= EOCRColor.BlackOnWhite;
ocr.MinCharWidth= 15;
ocr.MaxCharWidth= 50;
ocr.MinCharHeight= 15;
ocr.MaxCharHeight= 75;
ocr.NoiseArea= 15;

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

// Learn the characters
ocr.LearnPatterns(srcImage, text, (int)EOCRClass.Digit);

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

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= new EImageBW8();

// EOCR constructor
EOCR ocr= new EOCR();

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

// ...

// Recognize the characters
string text= ocr.Recognize(srcImage, 10, (int)EOCRClass.AllClasses);

// Alternatively
// Define the character filter (2 letters and 3 digits)
uint[] charFilter = new uint[5];
charFilter[0] = (uint)EOCRClass.UpperCase;
charFilter[1] = (uint)EOCRClass.UpperCase;
charFilter[2] = (uint)EOCRClass.Digit;
charFilter[3] = (uint)EOCRClass.Digit;
charFilter[4] = (uint)EOCRClass.Digit;

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

7.7. EasyOCR2

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 = new EImageBW8();
image.Load("image.tif");

// Attach a ROI to the image
EROIBW8 roi = new EROIBW8();
roi.Attach(image, 50, 224, 340, 96);

// Create an EOCR2 instance
EOCR2 ocr2 = new EOCR2();

// Set the expected character sizes
ocr2.CharsWidthRange = new EIntegerRange(25,25);
ocr2.CharsHeight = 37;

// Set the text polarity, in this case WhiteOnBlack
ocr2.TextPolarity = EasyOCR2TextPolarity.WhiteOnBlack;

// Set the topology
ocr2.Topology = ".{10}\n.{3} .{4}";

// Detect the text in the image. The output Text structure contains:
```
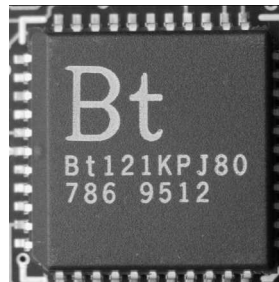
```
// - an individual textbox for each character
// - an individual bitmap image for each character
// - a threshold value to binarize the bitmap image for each character
// All structured in a hierarchy with Lines -> Words -> Characters
EOCR2Text text = ocr2.Detect(roi);

// Cleanup
text.Dispose();
ocr2.CharsWidthRange.Dispose();
ocr2.Dispose();
roi.Dispose();
image.Dispose();
```


The image used in this code snippet

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 = new EImageBW8();
image.Load("image.tif");

// Attach a ROI to the image
EROIBW8 roi = new EROIBW8();
roi.Attach(image, 50, 224, 340, 96);

// Create an EOCR2 instance
EOCR2 ocr2 = new EOCR2();

// Set the required parameters
ocr2.CharsWidthRange = new EIntegerRange(25,25);
ocr2.CharsHeight = 37;
ocr2.TextPolarity = EasyOCR2TextPolarity.WhiteOnBlack;
ocr2.Topology = ".{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.Text = "Bt121KPJ80\n786 9512";
//    3) Add the characters to the character database
ocr2.Learn(text);

// Save the character database
ocr2.SaveCharacterDatabase("myDB.o2d");
```
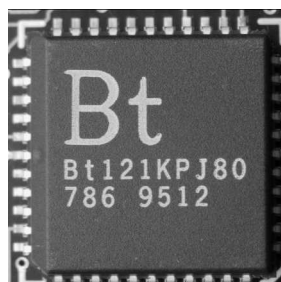
```
// Alternatively, save the model file.
// This will store the character database and the parameter settings
ocr2.Save("myModel.o2m");

// Cleanup
text.Dispose();
ocr2.CharsWidthRange.Dispose();
ocr2.Dispose();
roi.Dispose();
image.Dispose();
```


The image used in this code snippet

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 = new EImageBW8();
image.Load("image.tif");

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

// Create an EOCR2 instance
EOCR2 ocr2 = new EOCR2();

// Set the required parameters
ocr2.CharsWidthRange = new EIntegerRange(25,25);
ocr2.CharsHeight = 37;
ocr2.Topology = "[LN]{10}\nN{3} N{4}";
ocr2.TextPolarity = 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
string result = ocr2.Read(roi);
```

```
// Cleanup
ocr2.CharsWidthRange.Dispose();
ocr2.Dispose();
roi.Dispose();
image.Dispose();
```


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 = new EImageBW8();
image.Load("image.tif");

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

// Create an EOCR2 instance
EOCR2 ocr2 = new EOCR2();

// Set the required parameters
ocr2.CharsWidthRange = new EIntegerRange(25,25);
ocr2.CharsHeight = 37;
ocr2.Topology = "[LN]{10}\nN{3} N{4}";
ocr2.TextPolarity = EasyOCR2TextPolarity.WhiteOnBlack;

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

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

// Cleanup
ocr2.CharsWidthRange.Dispose();
ocr2.Dispose();
roi.Dispose();
image.Dispose();
```

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 = new EImageBW8();
image.Load("image.tif");

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

// Create an EOCR2 instance
EOCR2 ocr2 = new EOCR2();

// 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
string result = ocr2.Read(roi);

// Cleanup
ocr2.Dispose();
roi.Dispose();
image.Dispose();
```

View Bitmap

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

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