ECVL

Generated by Doxygen 1.8.15

Chapter 1

Documentation

ECVL is the European Computer Vision Library, under development within the European project DeepHealth. Here you can find the provisional doxygen documentation. Checkout the GitHub project `here`.

Chapter 2

Bug List

Member ecvl::Threshold (p. **35**) **(const Image** (p. **57**) **&src, Image** (p. **57**) **&dst, double thresh, double maxval, ThresholdingType thresh_type=ThresholdingType::BINARY** (p. **18**)**)**

Input and output Images may have different color spaces.

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Namespace Documentation

7.1 ecvl Namespace Reference

Classes

- struct **ConstContiguousIterator**
- class **ConstContiguousView**
- struct **ConstIterator**
- class **ConstView**
- struct **ContiguousIterator**
- class **ContiguousView**
- class **ContiguousViewXYC**
- class **Image**

    Image *(*p. *57) class.*
- struct **Iterator**
- class **MetaData**
- class **ShowApp**

    ShowApp *(*p. *77) is a custom wxApp which allows you to visualize an ECVL* ***Image*** *(*p. *57).*
- struct **SignedTable1D**
- struct **StructAdd**
- struct **StructCopyImage**
- struct **StructDiv**
- struct **StructMul**
- struct **StructScalarAdd**
- struct **StructScalarDiv**
- struct **StructScalarDivInv**
- struct **StructScalarMul**
- struct **StructScalarNeg**
- struct **StructScalarSub**
- struct **StructScalarSubInv**
- struct **StructSub**
- struct **Table1D**
- struct **Table2D**
- struct **TypeInfo**
- struct **TypeInfo< ecvl::DataType::float32 >**
- struct **TypeInfo< ecvl::DataType::float64 >**
- struct **TypeInfo< ecvl::DataType::int16 >**
- struct **TypeInfo< ecvl::DataType::int32 >**

- struct **TypeInfo**< **ecvl::DataType::int64** >
- struct **TypeInfo**< **ecvl::DataType::int8** >
- struct **TypeInfo**< **ecvl::DataType::none** >
- struct **TypeInfo**< **ecvl::DataType::uint16** >
- struct **TypeInfo**< **ecvl::DataType::uint32** >
- struct **TypeInfo**< **ecvl::DataType::uint64** >
- struct **TypeInfo**< **ecvl::DataType::uint8** >
- class **View**
- class **wxImagePanel**

  wxImagePanel *(p. 103) creates a wxPanel to contain an* ***Image*** *(p. 57).*

Enumerations

- enum **DataType** {
  DataType::ECVL_TUPLE, **DataType::int8**, **DataType::int16**, **DataType::int32**,
  DataType::int64, **DataType::float32**, **DataType::float64**, **DataType::uint8**,
  DataType::uint16, **DataType::uint32**, **DataType::uint64**, **DataType::none** }

  DataType is an enum class which defines data types allowed for images.
- enum **ColorType** {
  ColorType::none, **ColorType::GRAY**, **ColorType::RGB**, **ColorType::BGR**,
  ColorType::HSV, **ColorType::YCbCr** }

  Enum class representing the ECVL supported color spaces.
- enum **ThresholdingType** { **ThresholdingType::BINARY**, **ThresholdingType::BINARY_INV** }

  Enum class representing the ECVL threhsolding types.
- enum **InterpolationType** {
  InterpolationType::nearest, **InterpolationType::linear**, **InterpolationType::area**, **InterpolationType**↩
  ::cubic,
  InterpolationType::lanczos4 }

  Enum class representing the ECVL interpolation types.

Functions

- template< DataType ODT, typename IDT >
  TypeInfo< ODT >::basetype **saturate_cast** (IDT v)

  Saturate a value (of any type) to the specified type.
- template< typename ODT , typename IDT >
  ODT **saturate_cast** (const IDT &v)

  Saturate a value (of any type) to the specified type.
- void **Add** (**Image** &src1_dst, const **Image** &src2)
- void **Sub** (**Image** &src1_dst, const **Image** &src2)
- void **Mul** (**Image** &src1_dst, const **Image** &src2)
- template< typename T >
  Image & **Mul** (**Image** &img, T value, bool saturate=true)

  In-place multiplication between an ***Image*** *(p. 57) and a scalar value, without type promotion.*
- template< typename T >
  Image & **Mul** (T value, **Image** &img, bool saturate=true)
- template< typename T >
  Image & **Add** (**Image** &img, T value, bool saturate=true)

  In-place addition between an ***Image*** *(p. 57) and a scalar value, without type promotion.*
- template< typename T >
  Image & **Add** (T value, **Image** &img, bool saturate=true)

- template<typename T >
  Image & **Sub** (**Image** &img, T value, bool saturate=true)

  In-place subtraction between an ***Image*** *(p. 57) and a scalar value, without type promotion.*
- template<typename T >
  Image & **Sub** (T value, **Image** &img, bool saturate=true)

  In-place subtraction between a scalar value and an ***Image*** *(p. 57), without type promotion.*
- template<typename T >
  Image & **Div** (**Image** &img, T value, bool saturate=true)

  In-place division between an ***Image*** *(p. 57) and a scalar value, without type promotion.*
- template<typename T , typename ET = double>
  Image & **Div** (T value, **Image** &img, bool saturate=true, ET epsilon=std::numeric_limits< double >::min())

  In-place divion between a scalar value and an ***Image*** *(p. 57), without type promotion.*
- **Image** & **Neg** (**Image** &img)

  In-place negation of an ***Image*** *(p. 57).*
- void **Mul** (const **Image** &src1, const **Image** &src2, **Image** &dst, **DataType** dst_type, bool saturate=true)

  Multiplies two Image(s) and stores the result in a third ***Image*** *(p. 57).*
- void **Sub** (const **Image** &src1, const **Image** &src2, **Image** &dst, **DataType** dst_type, bool saturate=true)

  Subtracts two Image(s) and stores the result in a third ***Image*** *(p. 57).*
- void **Add** (const **Image** &src1, const **Image** &src2, **Image** &dst, **DataType** dst_type, bool saturate=true)

  Adds two Image(s) and stores the result in a third ***Image*** *(p. 57).*
- template<typename ET = double>
  void **Div** (const **Image** &src1, const **Image** &src2, **Image** &dst, **DataType** dst_type, bool saturate=true, ET epsilon=std::numeric_limits< double >::min())

  Divides two Image(s) and stores the result in a third ***Image*** *(p. 57).*
- uint8_t **DataTypeSize** (**DataType** dt)

  Provides the size in bytes of a given DataType.
- constexpr size_t **DataTypeSize** ()

  Function to get the number of existing DataType at compile time.
- constexpr size_t **DataTypeSignedSize** ()

  Function to get the number of existing signed DataType at compile time.
- constexpr std::array< **DataType**, **DataTypeSize**()> **DataTypeArray** ()

  Function to get a std::array with all the DataType values at compile time.
- constexpr std::array< **DataType**, **DataTypeSignedSize**()> **DataTypeSignedArray** ()

  Function to get a std::array with all the signed DataType values at compile time.
- void **RearrangeChannels** (const **Image** &src, **Image** &dst, const std::string &channels)

  Changes the order of the ***Image*** *(p. 57) dimensions.*
- void **CopyImage** (const **Image** &src, **Image** &dst, **DataType** new_type= **DataType::none**)

  Copies the source ***Image*** *(p. 57) into the destination* ***Image*** *(p. 57).*
- bool **ImRead** (const std::string &filename, **Image** &dst)

  Loads an image from a file.
- bool **ImRead** (const **filesystem::path** &filename, **Image** &dst)
- bool **ImWrite** (const std::string &filename, const **Image** &src)

  Saves an image into a specified file.
- bool **ImWrite** (const **filesystem::path** &filename, const **Image** &src)
- void **ResizeDim** (const **ecvl::Image** &src, **ecvl::Image** &dst, const std::vector< int > &newdims, **InterpolationType** interp= **InterpolationType::linear**)

  Resizes an ***Image*** *(p. 57) to a new dimension.*
- void **ResizeScale** (const **ecvl::Image** &src, **ecvl::Image** &dst, const std::vector< double > &scales, **InterpolationType** interp= **InterpolationType::linear**)

  Resizes an ***Image*** *(p. 57) by scaling the dimentions to a given scale factor.*
- void **Flip2D** (const **ecvl::Image** &src, **ecvl::Image** &dst)

  Flips an ***Image*** *(p. 57).*

- void **Mirror2D** (const **ecvl::Image** &src, **ecvl::Image** &dst)

  *Mirrors an **Image** (p. 57).*
- void **Rotate2D** (const **ecvl::Image** &src, **ecvl::Image** &dst, double angle, const std::vector< double > &center={}, double scale=1.0, **InterpolationType** interp= **InterpolationType::linear**)

  *Rotates an **Image** (p. 57).*
- void **RotateFullImage2D** (const **ecvl::Image** &src, **ecvl::Image** &dst, double angle, double scale=1.0, **InterpolationType** interp= **InterpolationType::linear**)

  *Rotates an **Image** (p. 57) resizing the output accordingly.*
- void **ChangeColorSpace** (const **Image** &src, **Image** &dst, **ColorType** new_type)

  *Copies the source **Image** (p. 57) into destination **Image** (p. 57) changing the color space.*
- void **Threshold** (const **Image** &src, **Image** &dst, double thresh, double maxval, **ThresholdingType** thresh_type= **ThresholdingType::BINARY**)

  *Applies a fixed threshold to an input **Image** (p. 57).*
- double **OtsuThreshold** (const **Image** &src)

  Calculates the Otsu thresholding value.
- **ecvl::Image** **MatToImage** (const cv::Mat &m)

  *Convert a cv::Mat into an **ecvl::Image** (p. 57).*
- cv::Mat **ImageToMat** (const **Image** &img)

  *Convert an ECVL **Image** (p. 57) into OpenCV Mat.*
- void **ImShow** (const **Image** &img)

  *Displays an **Image** (p. 57).*
- wxImage **wx_from_mat** (**Image** &img)

7.1.1 Enumeration Type Documentation

7.1.1.1 ColorType

`enum **ecvl::ColorType** [strong]`

Enum class representing the ECVL supported color spaces.

Enumerator

Enumerator

none	Special ColorType for Images that contain only data and do not have any ColorType
GRAY	Gray-scale ColorType
RGB	RGB ColorType
BGR	BGR ColorType
HSV	HSV ColorType
YCbCr	YCbCr ColorType

Definition at line 27 of file image.h.

7.1.1.2 DataType

`enum ecvl::DataType [strong]`

DataType is an enum class which defines data types allowed for images.

Enumerator

Enumerator

ECVL_TUPLE	
int8	int8_t
int16	int16_t
int32	int32_t
int64	int64_t
float32	float
float64	double
uint8	uint8_t
uint16	uint16_t
uint32	uint32_t
uint64	uint64_t
none	none type

Definition at line 15 of file datatype.h.

7.1.1.3 InterpolationType

`enum ecvl::InterpolationType [strong]`

Enum class representing the ECVL interpolation types.

Enumerator

Enumerator

nearest	Nearest neighbor interpolation
linear	Bilinear interpolation
area	Resampling using pixel area relation. It may be a preferred method for image decimation, as it gives moire-free results. But when the image is zoomed, it is similar to the nearest method.
cubic	Bicubic interpolation
lanczos4	Lanczos interpolation over 8x8 neighborhood

Definition at line 23 of file imgproc.h.

7.1.1.4 ThresholdingType

```
enum ecvl::ThresholdingType [strong]
```

Enum class representing the ECVL threhsolding types.

Enumerator

Enumerator

BINARY	$$\mathtt{dst}(x,y) = \begin{cases} \mathtt{maxval} & \text{if } \mathtt{src}(x,y) > \mathtt{thresh} \\ 0 & \text{otherwise} \end{cases}$$
BINARY_INV	$$\mathtt{dst}(x,y) = \begin{cases} 0 & \text{if } \mathtt{src}(x,y) > \mathtt{thresh} \\ \mathtt{maxval} & \text{otherwise} \end{cases}$$

Definition at line 14 of file imgproc.h.

7.1.2 Function Documentation

7.1.2.1 Add() [1/4]

```
void ecvl::Add (
              Image & src1_dst,
             const  Image & src2 )
```

Definition at line 36 of file arithmetic.cpp.

7.1.2.2 Add() [2/4]

```
template<typename T >
Image& ecvl::Add (
              Image & img,
             T value,
             bool saturate = true )
```

In-place addition between an **Image** (p. 57) and a scalar value, without type promotion.

The **Add()** (p. 18) function sums a scalar value to the input **Image** (p. 57) and stores the result in the same image. The type of the image will not change. By default a saturation will be applied. If it is not the desired behavior change the "saturate" parameter to false.

Parameters

Parameters

`in,out`	*img*	**Image** (p. 57) to be summed (in-place) by a scalar value.
`in`	*value*	Scalar value to use for the sum.
`in`	*saturate*	Whether to apply saturation or not. Default is true.

Returns

Reference to the **Image** (p. 57) containing the result of the sum.

Definition at line 177 of file arithmetic.h.

7.1.2.3 Add() [3/4]

```
template<typename T >
Image& ecvl::Add (
             T value,
              Image & img,
             bool saturate = true )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Definition at line 187 of file arithmetic.h.

7.1.2.4 Add() [4/4]

```
void ecvl::Add (
             const  Image & src1,
             const  Image & src2,
              Image & dst,
              DataType dst_type,
             bool saturate = true )
```

Adds two Image(s) and stores the result in a third **Image** (p. 57).

This procedure adds src1 and src2 Image(s) (src1 + src2) and stores the result in the dst **Image** (p. 57) that will have the specified DataType. By default a saturation will be applied. If it is not the desired behavior change the "saturate" parameter to false.

Parameters

Parameters

in	*src1*	Augend (first addend) **Image** (p. 57).
in	*src2*	Addend (second addend) **Image** (p. 57).
out	*dst*	**Image** (p. 57) into which save the result of the division.
in	*dst_type*	DataType that destination **Image** (p. 57) must have at the end of the operation.
in	*saturate*	Whether to apply saturation or not. Default is true.

Returns

Definition at line 126 of file arithmetic.cpp.

7.1.2.5 ChangeColorSpace()

```
void ecvl::ChangeColorSpace (
            const  Image & src,
             Image & dst,
             ColorType new_type )
```

Copies the source **Image** (p. 57) into destination **Image** (p. 57) changing the color space.

The ChangeColorSpace procedure convert the color space of the source **Image** (p. 57) into the specified color space. New data are copied into destination **Image** (p. 57). Source and destination can be contiguous or not and can also be the same **Image** (p. 57).

Parameters

Parameters

in	*src*	The input **Image** (p. 57) to convert in the new color space.
out	*dst*	The output **Image** (p. 57) in the "new_type" color space.
in	*new_type*	The new color space in which the src **Image** (p. 57) must be converted.

Definition at line 168 of file imgproc.cpp.

7.1.2.6 CopyImage()

```
void ecvl::CopyImage (
            const  Image & src,
             Image & dst,
             DataType new_type =  DataType::none )
```

Copies the source **Image** (p. 57) into the destination **Image** (p. 57).

The **CopyImage()** (p. 20) procedure takes an **Image** (p. 57) and copies its data into the destination **Image** (p. 57). Source and destination cannot be the same **Image** (p. 57). Source cannot be a **Image** (p. 57) with **DataType::none** (p. 17). The optional new_type parameter can be used to change the DataType of the destination **Image** (p. 57). This function is mainly designed to change the DataType of an **Image** (p. 57), copying its data into a new **Image** (p. 57) or to copy an **Image** (p. 57) into a **View** (p. 101) as a patch. So if you just want to copy an **Image** (p. 57) as it is, use the copy constructor or = instead. Anyway, the procedure will handle all the possible situations that may happen trying to avoid unnecessary allocations. When the DataType is not specified the function will have the following behaviors:

- if the destination **Image** (p. 57) is empty the source will be directly copied into the destination.
- if source and destination have different size in memory or different channels and the destination is the owner of data, the procedure will overwrite the destination **Image** (p. 57) creating a new **Image** (p. 57) (channels and dimensions will be the same of the source **Image** (p. 57), pixels type (DataType) will be the same of the destination **Image** (p. 57) if they are not none or the same of the source otherwise).
- if source and destination have different size in memory or different channels and the destination is not the owner of data, the procedure will throw an exception.
- if source and destination have different color types and the destination is the owner of data, the procedure produces a destination **Image** (p. 57) with the same color type of the source.
- if source and destination have different color types and the destination is not the owner of data, the procedure will throw an exception. When the DataType is specified the function will have the same behavior, but the destination **Image** (p. 57) will have the specified DataType.

Parameters

Parameters

`in`	*src*	Source **Image** (p. 57) to be copied into destination **Image** (p. 57).
`out`	*dst*	Destination **Image** (p. 57) that will hold a copy of the source **Image** (p. 57). Cannot be the source **Image** (p. 57).
`in`	*new_type*	Desired type for the destination **Image** (p. 57) after the copy. If none (default) the destination **Image** (p. 57) will preserve its type if it is not empty, otherwise it will have the same type of the source **Image** (p. 57).

Definition at line 95 of file image.cpp.

7.1.2.7 DataTypeArray()

```
constexpr std::array< DataType, DataTypeSize()> ecvl::DataTypeArray ( )
```

Function to get a std::array with all the DataType values at compile time.

Returns

A std::array with all the DataType values.

Definition at line 71 of file datatype.h.

7.1.2.8 DataTypeSignedArray()

```
constexpr std::array< DataType, DataTypeSignedSize()> ecvl::DataTypeSignedArray ( )
```

Function to get a std::array with all the signed DataType values at compile time.

Returns

A std::array with all the signed DataType values.

Definition at line 88 of file datatype.h.

7.1.2.9 DataTypeSignedSize()

```
constexpr size_t ecvl::DataTypeSignedSize ( )
```

Function to get the number of existing signed DataType at compile time.

Returns

The number of existing signed DataType.

Definition at line 16 of file datatype.h.

7.1.2.10 DataTypeSize() [1/2]

```
uint8_t ecvl::DataTypeSize (
             DataType dt )
```

Provides the size in bytes of a given DataType.

Given one of the **DataType** (p. 17), the function returns its size in bytes.

Parameters

Parameters

in	*dt*	A DataType.

Returns

The DataType size in bytes

Definition at line 11 of file datatype.cpp.

7.1.2.11 DataTypeSize() [2/2]

```
constexpr size_t ecvl::DataTypeSize ( )
```

Function to get the number of existing DataType at compile time.

Returns

The number of existing DataType.

Definition at line 43 of file datatype.h.

7.1.2.12 Div() [1/3]

```
template<typename T >
Image& ecvl::Div (
             Image & img,
             T value,
             bool saturate = true )
```

In-place division between an **Image** (p. 57) and a scalar value, without type promotion.

The **Div()** (p. 22) function divides an input **Image** (p. 57) by a scalar value and stores the result in the same image. The type of the image will not change. By default a saturation will be applied. If it is not the desired behavior change the "saturate" parameter to false.

Parameters

Parameters

in,out	*img*	**Image** (p. 57) to be divided (in-place) by a scalar value.
in	*value*	Scalar value to use for the division.
in	*saturate*	Whether to apply saturation or not. Default is true.

Returns

Reference to the **Image** (p. 57) containing the result of the division.

Definition at line 312 of file arithmetic.h.

7.1.2.13 Div() [2/3]

```
template<typename T , typename ET = double>
Image& ecvl::Div (
             T value,
              Image & img,
             bool saturate = true,
             ET epsilon = std::numeric_limits<double>::min() )
```

In-place divion between a scalar value and an **Image** (p. 57), without type promotion.

The **Div()** (p. 22) function divides a scalar value by the input **Image** (p. 57) and stores the result in the same image. The type of the image will not change. By default a saturation will be applied. If it is not the desired behavior change the "saturate" parameter to false.

Parameters

Parameters

in	*value*	Scalar value to use for the division (Dividend).
in,out	*img*	Divisor of the operation. It will store the final result.
in	*saturate*	Whether to apply saturation or not. Default is true.
in	*epsilon*	Small value to be added to the **Image** (p. 57) values before performing the division. If not specified by default it is the minimum positive number representable in a double.

Returns

Reference to the **Image** (p. 57) containing the result of the division.

Definition at line 358 of file arithmetic.h.

7.1.2.14 Div() [3/3]

```
template<typename ET = double>
void ecvl::Div (
             const  Image & src1,
             const  Image & src2,
              Image & dst,
              DataType dst_type,
             bool saturate = true,
             ET epsilon = std::numeric_limits<double>::min() )
```

Divides two Image(s) and stores the result in a third **Image** (p. 57).

This procedure divides the src1 **Image** (p. 57) by the src2 **Image** (p. 57) (src1/src2) and stores the result into the dst **Image** (p. 57) that will have the specified DataType. By default a saturation will be applied. If it is not the desired behavior change the "saturate" parameter to false.

Parameters

Parameters

in	*src1*	Dividend (numerator) **Image** (p. 57).
in	*src2*	Divisor (denominator) **Image** (p. 57).
out	*dst*	**Image** (p. 57) into which save the result of the division.
in	*dst_type*	DataType that destination **Image** (p. 57) must have at the end of the operation.
in	*saturate*	Whether to apply saturation or not. Default is true.
in	*epsilon*	Small value to be added to the **Image** (p. 57) values before performing the division.If not specified by default it is the minimum positive number representable in a double.

Returns

Definition at line 455 of file arithmetic.h.

7.1.2.15 Flip2D()

```
void ecvl::Flip2D (
             const  ecvl::Image & src,
              ecvl::Image & dst )
```

Flips an **Image** (p. 57).

The Flip2D procedure vertically flips an **Image** (p. 57).

Parameters

Parameters

in	*src*	The input **Image** (p. 57).
out	*dst*	The output flipped **Image** (p. 57).

Definition at line 73 of file imgproc.cpp.

7.1.2.16 ImageToMat()

```
cv::Mat ecvl::ImageToMat (
             const  Image & img )
```

Convert an ECVL **Image** (p. 57) into OpenCV Mat.

Parameters

Parameters

in	*img*	Input ECVL **Image** (p. 57).

Returns

Output OpenCV Mat.

Definition at line 98 of file support_opencv.cpp.

7.1.2.17 ImRead() [1/2]

```
bool ecvl::ImRead (
            const std::string & filename,
             Image & dst )
```

Loads an image from a file.

The function ImRead loads an image from the specified file. If the image cannot be read for any reason, the function creates an empty **Image** (p. 57) and returns false.

Parameters

Parameters

in	*filename*	A std::string identifying the file name. In order to be platform independent consider to use **ImRead(const filesystem::path& filename, Image& dst)** (p. 25) .
out	*dst*	**Image** (p. 57) in which data will be stored.

Returns

true if the image is correctly read, false otherwise.

Definition at line 10 of file imgcodecs.cpp.

7.1.2.18 ImRead() [2/2]

```
bool ecvl::ImRead (
            const  filesystem::path & filename,
             Image & dst )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This variant of ImRead is platform independent.

Parameters

Parameters

in	*filename*	A **filesystem::path** (p. 73) identifying the file name.
out	*dst*	**Image** (p. 57) in which data will be stored.

Returns

true if the image is correctly read, false otherwise.

Definition at line 16 of file imgcodecs.cpp.

7.1.2.19 ImShow()

```
void ecvl::ImShow (
             const  Image & img )
```

Displays an **Image** (p. 57).

The ImShow function instantiates a **ShowApp** (p. 77) and starts it with a wxEntry() call. The image is shown with its original size.

Parameters

Parameters

in	*img*	**Image** (p. 57) to be shown.

Definition at line 87 of file gui.cpp.

7.1.2.20 ImWrite() [1/2]

```
bool ecvl::ImWrite (
             const std::string & filename,
             const  Image & src )
```

Saves an image into a specified file.

The function ImWrite saves the input image into a specified file. The image format is chosen based on the filename extension. The following sample shows how to create a BGR image and save it to the PNG file "test.png":

```
#include "ecvl/core.h"
using namespace std;
using namespace ecvl;
using namespace filesystem;
int main()
{
    // Create BGR Image
    Image img({ 500, 500, 3 }, DataType::uint8, "xyc", ColorType::BGR);

    // Populate Image with pseudo-random data
    for (int r = 0; r < img.dims_[1]; ++r) {
        for (int c = 0; c < img.dims_[0]; ++c) {
            *img.Ptr({ c, r, 0 }) = 255;
            *img.Ptr({ c, r, 1 }) = (r / 2) % 255;
            *img.Ptr({ c, r, 2 }) = (r / 2) % 255;
        }
    }
    ImWrite(path("./test.png"), img);
    return EXIT_SUCCESS;
}
```

Parameters

Parameters

in	*filename*	A std::string identifying the output file name. In order to be platform independent consider to use **ImWrite(const filesystem::path& filename, const Image& src)** (p. 27).
in	*src*	**Image** (p. 57) to be saved.

Returns

true if the image is correctly write, false otherwise.

Definition at line 21 of file imgcodecs.cpp.

7.1.2.21 ImWrite() [2/2]

```
bool ecvl::ImWrite (
            const  filesystem::path & filename,
            const  Image & src )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This variant of ImWrite is platform independent.

Parameters

Parameters

in	*filename*	A **filesystem::path** (p. 73) identifying the output file name.
in	*src*	**Image** (p. 57) to be saved.

Returns

true if the image is correctly write, false otherwise.

Definition at line 26 of file imgcodecs.cpp.

7.1.2.22 MatToImage()

```
 Image ecvl::MatToImage (
            const cv::Mat & m )
```

Convert a cv::Mat into an **ecvl::Image** (p. 57).

Parameters

Parameters

in	*m*	Input OpenCV Mat.

Returns

ECVL image.

Definition at line 7 of file support_opencv.cpp.

7.1.2.23 Mirror2D()

```
void ecvl::Mirror2D (
            const  ecvl::Image & src,
             ecvl::Image & dst )
```

Mirrors an **Image** (p. 57).

The Mirror2D procedure horizontally flips an **Image** (p. 57).

Parameters

Parameters

in	*src*	The input **Image** (p. 57).
out	*dst*	The output mirrored **Image** (p. 57).

Definition at line 89 of file imgproc.cpp.

7.1.2.24 Mul() [1/4]

```
void ecvl::Mul (
              Image & src1_dst,
             const  Image & src2 )
```

Definition at line 70 of file arithmetic.cpp.

7.1.2.25 Mul() [2/4]

```
template<typename T >
Image& ecvl::Mul (
              Image & img,
             T value,
             bool saturate = true )
```

In-place multiplication between an **Image** (p. 57) and a scalar value, without type promotion.

The **Mul()** (p. 27) function multiplies an input image by a scalar value and stores the result in the same image. The type of the image will not change. By default a saturation will be applied. If it is not the desired behavior change the "saturate" parameter to false.

Parameters

Parameters

`in,out`	*img*	**Image** (p. 57) to be multiplied (in-place) by a scalar value.
`in`	*value*	Scalar value to use for the multiplication.
`in`	*saturate*	Whether to apply saturation or not. Default is true.

Returns

Reference to the **Image** (p. 57) containing the result of the multiplication.

Definition at line 128 of file arithmetic.h.

7.1.2.26 Mul() [3/4]

```
template<typename T >
Image& ecvl::Mul (
             T value,
              Image & img,
             bool saturate = true )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Definition at line 138 of file arithmetic.h.

7.1.2.27 Mul() [4/4]

```
void ecvl::Mul (
            const  Image & src1,
            const  Image & src2,
             Image & dst,
             DataType dst_type,
            bool saturate = true )
```

Multiplies two Image(s) and stores the result in a third **Image** (p. 57).

This procedure multiplies two Image(s) together and stores the result in a third **Image** (p. 57) that will have the specified DataType. By default a saturation will be applied. If it is not the desired behavior change the "saturate" parameter to false.

Parameters

Parameters

in	*src1*	Multiplier (first factor) **Image** (p. 57).
in	*src2*	Multiplicand (second factor) **Image** (p. 57).
out	*dst*	**Image** (p. 57) into which save the result of the multiplication.
in	*dst_type*	DataType that destination **Image** (p. 57) must have at the end of the operation.
in	*saturate*	Whether to apply saturation or not. Default is true.

Returns

Definition at line 125 of file arithmetic.cpp.

7.1.2.28 Neg()

```
 Image & ecvl::Neg (
             Image & img )
```

In-place negation of an **Image** (p. 57).

The **Neg()** (p. 29) function negates every value of an **Image** (p. 57), and stores the the result in the same image. The type of the image will not change.

Parameters

Parameters

in,out	*img*	**Image** (p. 57) to be negated (in-place).

Returns

Reference to the **Image** (p. 57) containing the result of the negation.

Definition at line 93 of file arithmetic.cpp.

7.1.2.29 OtsuThreshold()

```
double ecvl::OtsuThreshold (
             const  Image & src )
```

Calculates the Otsu thresholding value.

The OtsuThreshold function calculates the Otsu threshold value over a given input **Image** (p. 57). the **Image** (p. 57) must by **ColorType::GRAY** (p. 16).

Parameters

Parameters

in	*src*	Input **Image** (p. 57) on which to calculato the Otsu threshold value.

Returns

Otsu threshold value.

Definition at line 274 of file imgproc.cpp.

7.1.2.30 RearrangeChannels()

```
void ecvl::RearrangeChannels (
             const  Image & src,
              Image & dst,
             const std::string & channels )
```

Changes the order of the **Image** (p. 57) dimensions.

The RearrangeChannels procedure changes the order of the input **Image** (p. 57) dimensions saving the result into the output **Image** (p. 57). The new order of dimensions can be specified as a string through the "channels" parameter. Input and output Images can be the same. The number of channels of the input **Image** (p. 57) must be the same of required channels.

Parameters

Parameters

in	*src*	Input **Image** (p. 57) on which to rearrange dimensions.
out	*dst*	The output rearranged **Image** (p. 57). Can be the src **Image** (p. 57).
in	*channels*	Desired order of **Image** (p. 57) channels.

Definition at line 49 of file image.cpp.

7.1.2.31 ResizeDim()

```
void ecvl::ResizeDim (
            const ecvl::Image & src,
            ecvl::Image & dst,
            const std::vector< int > & newdims,
            InterpolationType interp = InterpolationType::linear )
```

Resizes an **Image** (p. 57) to a new dimension.

The function resizes **Image** (p. 57) src and outputs the result in dst.

Parameters

Parameters

in	*src*	The input **Image** (p. 57).
out	*dst*	The output resized **Image** (p. 57).
in	*newdims*	std::vector<int> that specifies the new size of each dimension. The vector size must match the src **Image** (p. 57) dimentions, excluding the color channel
in	*interp*	InterpolationType to be used. See **InterpolationType** (p. 17).

Definition at line 30 of file imgproc.cpp.

7.1.2.32 ResizeScale()

```
void ecvl::ResizeScale (
            const ecvl::Image & src,
            ecvl::Image & dst,
            const std::vector< double > & scales,
            InterpolationType interp = InterpolationType::linear )
```

Resizes an **Image** (p. 57) by scaling the dimentions to a given scale factor.

The function resizes **Image** (p. 57) src and outputs the result in dst.

Parameters

Parameters

in	*src*	The input **Image** (p. 57).
out	*dst*	The output resized **Image** (p. 57).
in	*scales*	std::vector<double> that specifies the scale to apply to each dimension. The vector size must match the src **Image** (p. 57) dimentions, excluding the color channel.
in	*interp*	InterpolationType to be used. See **InterpolationType** (p. 17).

Definition at line 50 of file imgproc.cpp.

7.1.2.33 Rotate2D()

```
void ecvl::Rotate2D (
            const ecvl::Image & src,
            ecvl::Image & dst,
            double angle,
            const std::vector< double > & center = {},
            double scale = 1.0,
            InterpolationType interp = InterpolationType::linear )
```

Rotates an **Image** (p. 57).

The Rotate2D procedure rotates an **Image** (p. 57) of a given angle (expressed in degrees) in a clockwise manner, with respect to a given center. The value of unknown pixels in the output **Image** (p. 57) are set to 0. The output **Image** (p. 57) is guaranteed to have the same dimensions as the input one. An optional scale parameter can be provided: this won't change the output **Image** (p. 57) size, but the image is scaled during rotation. Different interpolation types are available, see **InterpolationType** (p. 17).

Parameters

Parameters

in	*src*	The input **Image** (p. 57).
out	*dst*	The output rotated **Image** (p. 57).
in	*angle*	The rotation angle in degrees.
in	*center*	A std::vector<double> representing the coordinates of the rotation center. If empty, the center of the image is used.
in	*scale*	Optional scaling factor.
in	*interp*	Interpolation type used. Default is **InterpolationType::linear** (p. 17).

Definition at line 105 of file imgproc.cpp.

7.1.2.34 RotateFullImage2D()

```
void ecvl::RotateFullImage2D (
            const ecvl::Image & src,
            ecvl::Image & dst,
            double angle,
            double scale = 1.0,
            InterpolationType interp = InterpolationType::linear )
```

Rotates an **Image** (p. 57) resizing the output accordingly.

The RotateFullImage2D procedure rotates an **Image** (p. 57) of a given angle (expressed in degrees) in a clockwise manner. The value of unknown pixels in the output **Image** (p. 57) are set to 0. The output **Image** (p. 57) is guaranteed to contain all the pixels of the rotated image. Thus, its dimensions can be different from those of the input. An optional scale parameter can be provided. Different interpolation types are available, see **InterpolationType** (p. 17).

Parameters

Parameters

in	*src*	The input **Image** (p. 57).
out	*dst*	The rotated output **Image** (p. 57).
in	*angle*	The rotation angle in degrees.
in	*scale*	Optional scaling factor.
in	*interp*	Interpolation type used. Default is **InterpolationType::linear** (p. 17).

Definition at line 134 of file imgproc.cpp.

7.1.2.35 saturate_cast() [1/2]

```
template<DataType ODT, typename IDT >
TypeInfo<ODT>::basetype ecvl::saturate_cast (
             IDT v )
```

Saturate a value (of any type) to the specified type.

Given an input of any type the saturate_cast function provide an output return value of the specified type applying saturation. When the input value in greater than the maximum possible value (max) for the output type, the max value is returned. When the input value in lower than the minimum possible value (min) for the output type, the min value is returned.

Parameters

Parameters

in	*v*	Input value (of any type).

Returns

Input value after cast and saturation.

Definition at line 25 of file arithmetic.h.

7.1.2.36 saturate_cast() [2/2]

```
template<typename ODT , typename IDT >
ODT ecvl::saturate_cast (
             const IDT & v )
```

Saturate a value (of any type) to the specified type.

Given an input of any type the saturate_cast function provide an output return value of the specified type applying saturation. When the input value in greater than the maximum possible value (max) for the output type, the max value is returned. When the input value in lower than the minimum possible value (min) for the output type, the min value is returned.

Parameters

Parameters

in	*v*	Input value (of any type).

Returns

Input value after cast and saturation.

Definition at line 52 of file arithmetic.h.

7.1.2.37 Sub() [1/4]

```
void ecvl::Sub (
             Image & src1_dst,
             const  Image & src2 )
```

Definition at line 53 of file arithmetic.cpp.

7.1.2.38 Sub() [2/4]

```
template<typename T >
Image& ecvl::Sub (
             Image & img,
             T value,
             bool saturate = true )
```

In-place subtraction between an **Image** (p. 57) and a scalar value, without type promotion.

The **Sub()** (p. 33) function subtracts a scalar value from the input **Image** (p. 57) and stores the result in the same image. The type of the image will not change. By default a saturation will be applied. If it is not the desired behavior change the "saturate" parameter to false.

Parameters

Parameters

in,out	*img*	**Image** (p. 57) to be subtracted (in-place) by a scalar value.
in	*value*	Scalar value to use for the subtraction.
in	*saturate*	Whether to apply saturation or not. Default is true.

Returns

Reference to the **Image** (p. 57) containing the result of the subtraction.

Definition at line 226 of file arithmetic.h.

7.1.2.39 Sub() [3/4]

```
template<typename T >
Image& ecvl::Sub (
             T value,
             Image & img,
             bool saturate = true )
```

In-place subtraction between a scalar value and an **Image** (p. 57), without type promotion.

The **Sub()** (p. 33) function subtracts the input **Image** (p. 57) from a scalar value and stores the result in the same image. The type of the image will not change. By default a saturation will be applied. If it is not the desired behavior change the "saturate" parameter to false.

Parameters

Parameters

in	*value*	Scalar value to use for the subtraction (Minuend).
in,out	*img*	Subtrahend of the operation. It will store the final result.
in	*saturate*	Whether to apply saturation or not. Default is true.

Returns

Reference to the **Image** (p. 57) containing the result of the subtraction.

Definition at line 269 of file arithmetic.h.

7.1.2.40 Sub() [4/4]

```
void ecvl::Sub (
            const  Image & src1,
            const  Image & src2,
             Image & dst,
             DataType dst_type,
            bool saturate = true )
```

Subtracts two Image(s) and stores the result in a third **Image** (p. 57).

This procedure subtracts the src2 **Image** (p. 57) from the src1 **Image** (p. 57) (src1 - src2) and stores the result in the dst **Image** (p. 57) that will have the specified DataType. By default a saturation will be applied. If it is not the desired behavior change the "saturate" parameter to false.

Parameters

Parameters

in	*src1*	Minuend **Image** (p. 57).
in	*src2*	Subtrahend **Image** (p. 57).
out	*dst*	**Image** (p. 57) into which save the result of the division.
in	*dst_type*	DataType that destination **Image** (p. 57) must have at the end of the operation.
in	*saturate*	Whether to apply saturation or not. Default is true.

Returns

Definition at line 127 of file arithmetic.cpp.

7.1.2.41 Threshold()

```
void ecvl::Threshold (
            const  Image & src,
             Image & dst,
            double thresh,
            double maxval,
             ThresholdingType thresh_type =  ThresholdingType::BINARY )
```

Applies a fixed threshold to an input **Image** (p. 57).

The Threshold function applies a fixed thresholding to an input **Image** (p. 57). The function is useful to get a binary image out of a grayscale (**ColorType::GRAY** (p. 16)) **Image** (p. 57) or to remove noise filtering out pixels with too small or too large values. Anyway, the function can be applied to any input **Image** (p. 57). The pixels up to "thresh" value will be set to 0, the pixels above this value will be set to "maxvalue" if "thresh_type" is **ThresholdingType::↩ BINARY** (p. 18) (default). The opposite will happen if "thresh_type" is **ThresholdingType::BINARY_INV** (p. 18).

Bug Input and output Images may have different color spaces.

Parameters

Parameters

in	*src*	Input **Image** (p. 57) on which to apply the threshold.
out	*dst*	The output thresholded **Image** (p. 57).
in	*thresh*	Threshold value.
in	*maxval*	The maximum values in the thresholded **Image** (p. 57).
in	*thresh_type*	Type of threshold to be applied, see **ThresholdingType** (p. 18). The default value is **ThresholdingType::BINARY** (p. 18).

Definition at line 258 of file imgproc.cpp.

7.1.2.42 wx_from_mat()

```
wxImage ecvl::wx_from_mat (
              Image & img )
```

Definition at line 45 of file gui.cpp.

7.2 filesystem Namespace Reference

Classes

- class **path**

Functions

- **path operator/** (const **path** &lhs, const **path** &rhs)
- bool **exists** (const **path** &p)
- bool **exists** (const **path** &p, std::error_code &ec)
- bool **create_directories** (const **path** &p)
- bool **create_directories** (const **path** &p, std::error_code &ec)
- void **copy** (const **path** &from, const **path** &to)
- void **copy** (const **path** &from, const **path** &to, std::error_code &ec)
- bool **exists** (const **path** &p, error_code &ec)
- bool **create_directories** (const **path** &p, error_code &ec)
- void **copy** (const **path** &from, const **path** &to, error_code &ec)

7.2.1 Function Documentation

7.2.1.1 copy() [1/3]

```
void filesystem::copy (
             const  path & from,
             const  path & to,
             error_code & ec )
```

Definition at line 93 of file filesystem.cc.

7.2.1.2 copy() [2/3]

```
void filesystem::copy (
             const  path & from,
             const  path & to )
```

Definition at line 77 of file filesystem.cc.

7.2.1.3 copy() [3/3]

```
void filesystem::copy (
             const  path & from,
             const  path & to,
             std::error_code & ec )
```

7.2.1.4 create_directories() [1/3]

```
bool filesystem::create_directories (
             const  path & p,
             error_code & ec )
```

Definition at line 61 of file filesystem.cc.

7.2.1.5 create_directories() [2/3]

```
bool filesystem::create_directories (
             const  path & p )
```

Definition at line 43 of file filesystem.cc.

7.2.1.6 create_directories() [3/3]

```
bool filesystem::create_directories (
            const  path & p,
            std::error_code & ec )
```

7.2.1.7 exists() [1/3]

```
bool filesystem::exists (
            const  path & p,
            error_code & ec )
```

Definition at line 38 of file filesystem.cc.

7.2.1.8 exists() [2/3]

```
bool filesystem::exists (
            const  path & p )
```

Definition at line 20 of file filesystem.cc.

7.2.1.9 exists() [3/3]

```
bool filesystem::exists (
            const  path & p,
            std::error_code & ec )
```

7.2.1.10 operator/()

```
 path filesystem::operator/ (
            const  path & lhs,
            const  path & rhs )  [inline]
```

Definition at line 109 of file filesystem.h.

Chapter 8

Class Documentation

8.1 ecvl::ConstContiguousIterator< T > Struct Template Reference

```
#include <iterators.h>
```

Public Member Functions

- **ConstContiguousIterator** (const **Image** &img, std::vector< int > pos={})
- **ConstContiguousIterator** & **operator++** ()
- const T & **operator** ∗ () const
- const T ∗ **operator->** () const
- bool **operator==** (const **ConstContiguousIterator** &rhs) const
- bool **operator!=** (const **ConstContiguousIterator** &rhs) const

Public Attributes

- uint8_t ∗ **ptr_**
- const **Image** ∗ **img_**

8.1.1 Detailed Description

template<typename T>
struct ecvl::ConstContiguousIterator< T >

Definition at line 68 of file iterators.h.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 ConstContiguousIterator()

```
template<typename T >
ConstContiguousIterator::ConstContiguousIterator (
            const  Image & img,
            std::vector< int > pos = {} )
```

Definition at line 78 of file image.h.

8.1.3 Member Function Documentation

8.1.3.1 operator ∗()

```
template<typename T >
const T&  ecvl::ConstContiguousIterator< T >::operator ∗ ( ) const  [inline]
```

Definition at line 74 of file iterators.h.

8.1.3.2 operator"!=()

```
template<typename T >
bool  ecvl::ConstContiguousIterator< T >::operator!= (
            const  ConstContiguousIterator< T > & rhs ) const  [inline]
```

Definition at line 77 of file iterators.h.

8.1.3.3 operator++()

```
template<typename T >
ConstContiguousIterator&  ecvl::ConstContiguousIterator< T >::operator++ ( )  [inline]
```

Definition at line 73 of file iterators.h.

8.1.3.4 operator->()

```
template<typename T >
const T∗  ecvl::ConstContiguousIterator< T >::operator-> ( ) const  [inline]
```

Definition at line 75 of file iterators.h.

8.1.3.5 operator==()

```
template<typename T >
bool ecvl::ConstContiguousIterator< T >::operator== (
            const ConstContiguousIterator< T > & rhs ) const [inline]
```

Definition at line 76 of file iterators.h.

8.1.4 Member Data Documentation

8.1.4.1 img_

```
template<typename T >
const Image* ecvl::ConstContiguousIterator< T >::img_
```

Definition at line 70 of file iterators.h.

8.1.4.2 ptr_

```
template<typename T >
uint8_t* ecvl::ConstContiguousIterator< T >::ptr_
```

Definition at line 69 of file iterators.h.

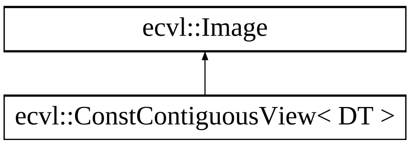
The documentation for this struct was generated from the following files:

- **iterators.h**
- **image.h**
- **iterators_impl.inc.h**

8.2 ecvl::ConstContiguousView< DT > Class Template Reference

```
#include <image.h>
```

Inheritance diagram for ecvl::ConstContiguousView< DT >:



Public Types

- using **basetype** = typename **TypeInfo**< DT >:: **basetype**

Public Member Functions

- **ConstContiguousView** (**Image** &img)
- const **basetype** & **operator()** (const std::vector< int > &coords)
- **ConstContiguousIterator**< **basetype** > **Begin** ()
- **ConstContiguousIterator**< **basetype** > **End** ()

Additional Inherited Members

8.2.1 Detailed Description

template<DataType DT>
class ecvl::ConstContiguousView< DT >

Definition at line 474 of file image.h.

8.2.2 Member Typedef Documentation

8.2.2.1 basetype

```
template<DataType DT>
using ecvl::ConstContiguousView< DT >:: basetype = typename TypeInfo<DT>:: basetype
```

Definition at line 476 of file image.h.

8.2.3 Constructor & Destructor Documentation

8.2.3.1 ConstContiguousView()

```
template<DataType DT>
ecvl::ConstContiguousView< DT >:: ConstContiguousView (
              Image & img )  [inline]
```

Definition at line 478 of file image.h.

8.2.4 Member Function Documentation

8.2.4.1 Begin()

```
template<DataType DT>
ConstContiguousIterator< basetype>  ecvl::ConstContiguousView< DT >::Begin ( )  [inline]
```

Definition at line 496 of file image.h.

8.2.4.2 End()

```
template<DataType DT>
ConstContiguousIterator< basetype>  ecvl::ConstContiguousView< DT >::End ( )  [inline]
```

Definition at line 497 of file image.h.

8.2.4.3 operator()()

```
template<DataType DT>
const  basetype&  ecvl::ConstContiguousView< DT >::operator() (
             const std::vector< int > & coords )  [inline]
```

Definition at line 492 of file image.h.

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

- **image.h**

8.3 ecvl::ConstIterator< T > Struct Template Reference

```
#include <iterators.h>
```

Public Types

- typedef **ConstIterator** &(ConstIterator::∗ **IncrementMemFn**) ()

Public Member Functions

- **ConstIterator** (const **Image** &img, std::vector< int > pos={})
- **ConstIterator** & **operator++** ()
- const T & **operator** ∗ () const
- const T ∗ **operator->** () const
- bool **operator==** (const **ConstIterator** &rhs) const
- bool **operator!=** (const **ConstIterator** &rhs) const

Public Attributes

- std::vector< int > **pos_**
- const uint8_t ∗ **ptr_**
- const **Image** ∗ **img_**
- **IncrementMemFn** **incrementor** = & **ConstIterator**<T>::IncrementPos

8.3.1 Detailed Description

template<typename T>
struct ecvl::ConstIterator< T >

Definition at line 32 of file iterators.h.

8.3.2 Member Typedef Documentation

8.3.2.1 IncrementMemFn

```
template<typename T >
typedef  ConstIterator&(ConstIterator::*  ecvl::ConstIterator< T >::IncrementMemFn) ()
```

Definition at line 37 of file iterators.h.

8.3.3 Constructor & Destructor Documentation

8.3.3.1 ConstIterator()

```
template<typename T >
ConstIterator::ConstIterator (
             const  Image & img,
             std::vector< int > pos = {} )
```

Definition at line 30 of file image.h.

8.3.4 Member Function Documentation

8.3.4.1 operator *()

```
template<typename T >
const T&  ecvl::ConstIterator< T >::operator * ( ) const  [inline]
```

Definition at line 42 of file iterators.h.

8.3.4.2 operator"!=()

```
template<typename T >
bool  ecvl::ConstIterator< T >::operator!= (
             const  ConstIterator< T > & rhs ) const  [inline]
```

Definition at line 45 of file iterators.h.

8.3.4.3 operator++()

```
template<typename T >
ConstIterator&  ecvl::ConstIterator< T >::operator++ ( )  [inline]
```

Definition at line 41 of file iterators.h.

8.3.4.4 operator->()

```
template<typename T >
const T*  ecvl::ConstIterator< T >::operator-> ( ) const  [inline]
```

Definition at line 43 of file iterators.h.

8.3.4.5 operator==()

```
template<typename T >
bool  ecvl::ConstIterator< T >::operator== (
             const  ConstIterator< T > & rhs ) const  [inline]
```

Definition at line 44 of file iterators.h.

8.3.5 Member Data Documentation

8.3.5.1 img_

```
template<typename T >
const Image* ecvl::ConstIterator< T >::img_
```

Definition at line 35 of file iterators.h.

8.3.5.2 incrementor

```
template<typename T >
IncrementMemFn ecvl::ConstIterator< T >::incrementor = & ConstIterator<T>::IncrementPos
```

Definition at line 38 of file iterators.h.

8.3.5.3 pos_

```
template<typename T >
std::vector<int> ecvl::ConstIterator< T >::pos_
```

Definition at line 33 of file iterators.h.

8.3.5.4 ptr_

```
template<typename T >
const uint8_t* ecvl::ConstIterator< T >::ptr_
```

Definition at line 34 of file iterators.h.

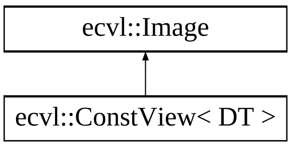
The documentation for this struct was generated from the following files:

- **iterators.h**
- **image.h**
- **iterators_impl.inc.h**

8.4 ecvl::ConstView< DT > Class Template Reference

```
#include <image.h>
```

Inheritance diagram for ecvl::ConstView< DT >:



Public Types

- using **basetype** = typename **TypeInfo**< DT >:: **basetype**

Public Member Functions

- **ConstView** (const **Image** &img)
- const **basetype** & **operator()** (const std::vector< int > &coords)
- **ConstIterator**< **basetype** > **Begin** ()
- **ConstIterator**< **basetype** > **End** ()

Additional Inherited Members

8.4.1 Detailed Description

template<DataType DT>
class ecvl::ConstView< DT >

Definition at line 420 of file image.h.

8.4.2 Member Typedef Documentation

8.4.2.1 basetype

```
template<DataType DT>
using ecvl::ConstView< DT >:: basetype = typename TypeInfo<DT>:: basetype
```

Definition at line 422 of file image.h.

8.4.3 Constructor & Destructor Documentation

8.4.3.1 ConstView()

```
template<DataType DT>
ecvl::ConstView< DT >::  ConstView (
             const  Image & img )  [inline]
```

Definition at line 424 of file image.h.

8.4.4 Member Function Documentation

8.4.4.1 Begin()

```
template<DataType DT>
ConstIterator< basetype>  ecvl::ConstView< DT >::Begin ( )  [inline]
```

Definition at line 442 of file image.h.

8.4.4.2 End()

```
template<DataType DT>
ConstIterator< basetype>  ecvl::ConstView< DT >::End ( )  [inline]
```

Definition at line 443 of file image.h.

8.4.4.3 operator()()

```
template<DataType DT>
const  basetype&  ecvl::ConstView< DT >::operator() (
             const std::vector< int > & coords )  [inline]
```

Definition at line 438 of file image.h.

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

- **image.h**

8.5 ecvl::ContiguousIterator< T > Struct Template Reference

```
#include <iterators.h>
```

Public Member Functions

- **ContiguousIterator** (**Image** &img, std::vector< int > pos={})
- **ContiguousIterator** & **operator++** ()
- T & **operator** ∗ () const
- T ∗ **operator->** () const
- bool **operator==** (const **ContiguousIterator** &rhs) const
- bool **operator!=** (const **ContiguousIterator** &rhs) const

Public Attributes

- uint8_t ∗ **ptr_**
- **Image** ∗ **img_**

8.5.1 Detailed Description

template<typename T>
struct ecvl::ContiguousIterator< T >

Definition at line 53 of file iterators.h.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 ContiguousIterator()

```
template<typename T >
ContiguousIterator::ContiguousIterator (
             Image & img,
            std::vector< int > pos = {} )
```

Definition at line 56 of file image.h.

8.5.3 Member Function Documentation

8.5.3.1 operator ∗()

```
template<typename T >
T&  ecvl::ContiguousIterator< T >::operator * ( ) const  [inline]
```

Definition at line 59 of file iterators.h.

8.5.3.2 operator"!=()

```
template<typename T >
bool  ecvl::ContiguousIterator< T >::operator!= (
            const  ContiguousIterator< T > & rhs ) const  [inline]
```

Definition at line 62 of file iterators.h.

8.5.3.3 operator++()

```
template<typename T >
ContiguousIterator&  ecvl::ContiguousIterator< T >::operator++ ( )  [inline]
```

Definition at line 58 of file iterators.h.

8.5.3.4 operator->()

```
template<typename T >
T*  ecvl::ContiguousIterator< T >::operator-> ( ) const  [inline]
```

Definition at line 60 of file iterators.h.

8.5.3.5 operator==()

```
template<typename T >
bool  ecvl::ContiguousIterator< T >::operator== (
            const  ContiguousIterator< T > & rhs ) const  [inline]
```

Definition at line 61 of file iterators.h.

8.5.4 Member Data Documentation

8.5.4.1 img_

```
template<typename T >
Image*  ecvl::ContiguousIterator< T >::img_
```

Definition at line 55 of file iterators.h.

8.5.4.2 ptr_

```
template<typename T >
uint8_t* ecvl::ContiguousIterator< T >::ptr_
```

Definition at line 54 of file iterators.h.

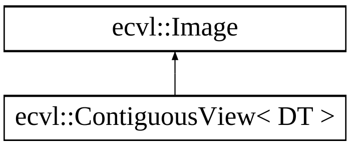
The documentation for this struct was generated from the following files:

- **iterators.h**
- **image.h**
- **iterators_impl.inc.h**

8.6 ecvl::ContiguousView< DT > Class Template Reference

```
#include <image.h>
```

Inheritance diagram for ecvl::ContiguousView< DT >:



Public Types

- using **basetype** = typename **TypeInfo**< DT >:: **basetype**

Public Member Functions

- **ContiguousView** (**Image** &img)
- **basetype** & **operator()** (const std::vector< int > &coords)
- **ContiguousIterator**< **basetype** > **Begin** ()
- **ContiguousIterator**< **basetype** > **End** ()

Additional Inherited Members

8.6.1 Detailed Description

template<DataType DT>
class ecvl::ContiguousView< DT >

Definition at line 447 of file image.h.

8.6.2 Member Typedef Documentation

8.6.2.1 basetype

```
template<DataType DT>
using ecvl::ContiguousView< DT >:: basetype = typename TypeInfo<DT>:: basetype
```

Definition at line 449 of file image.h.

8.6.3 Constructor & Destructor Documentation

8.6.3.1 ContiguousView()

```
template<DataType DT>
ecvl::ContiguousView< DT >:: ContiguousView (
              Image & img ) [inline]
```

Definition at line 451 of file image.h.

8.6.4 Member Function Documentation

8.6.4.1 Begin()

```
template<DataType DT>
ContiguousIterator< basetype> ecvl::ContiguousView< DT >::Begin ( ) [inline]
```

Definition at line 469 of file image.h.

8.6.4.2 End()

```
template<DataType DT>
ContiguousIterator< basetype> ecvl::ContiguousView< DT >::End ( ) [inline]
```

Definition at line 470 of file image.h.

8.6.4.3 operator()()

```
template<DataType DT>
basetype& ecvl::ContiguousView< DT >::operator() (
            const std::vector< int > & coords ) [inline]
```

Definition at line 465 of file image.h.

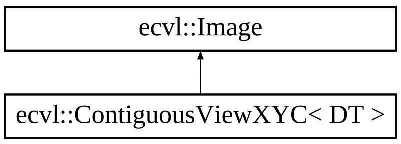
The documentation for this class was generated from the following file:

- **image.h**

8.7 ecvl::ContiguousViewXYC< DT > Class Template Reference

```
#include <image.h>
```

Inheritance diagram for ecvl::ContiguousViewXYC< DT >:



Public Types

- using **basetype** = typename **TypeInfo**< DT >:: **basetype**

Public Member Functions

- **ContiguousViewXYC** (**Image** &img)
- int **width** () const
- int **height** () const
- int **channels** () const
- **basetype** & **operator()** (int x, int y, int c)
- **ContiguousIterator**< **basetype** > **Begin** ()
- **ContiguousIterator**< **basetype** > **End** ()

Additional Inherited Members

8.7.1 Detailed Description

template<DataType DT>
class ecvl::ContiguousViewXYC< DT >

Definition at line 501 of file image.h.

8.7.2 Member Typedef Documentation

8.7.2.1 basetype

```
template<DataType DT>
using ecvl::ContiguousViewXYC< DT >:: basetype = typename TypeInfo<DT>:: basetype
```

Definition at line 503 of file image.h.

8.7.3 Constructor & Destructor Documentation

8.7.3.1 ContiguousViewXYC()

```
template<DataType DT>
ecvl::ContiguousViewXYC< DT >:: ContiguousViewXYC (
              Image & img ) [inline]
```

Definition at line 505 of file image.h.

8.7.4 Member Function Documentation

8.7.4.1 Begin()

```
template<DataType DT>
ContiguousIterator< basetype> ecvl::ContiguousViewXYC< DT >::Begin ( ) [inline]
```

Definition at line 531 of file image.h.

8.7.4.2 channels()

```
template<DataType DT>
int ecvl::ContiguousViewXYC< DT >::channels ( ) const [inline]
```

Definition at line 525 of file image.h.

8.7.4.3 End()

```
template<DataType DT>
ContiguousIterator< basetype>  ecvl::ContiguousViewXYC< DT >::End ( )  [inline]
```

Definition at line 532 of file image.h.

8.7.4.4 height()

```
template<DataType DT>
int  ecvl::ContiguousViewXYC< DT >::height ( ) const  [inline]
```

Definition at line 524 of file image.h.

8.7.4.5 operator()()

```
template<DataType DT>
basetype&  ecvl::ContiguousViewXYC< DT >::operator() (
             int x,
             int y,
             int c )  [inline]
```

Definition at line 527 of file image.h.

8.7.4.6 width()

```
template<DataType DT>
int  ecvl::ContiguousViewXYC< DT >::width ( ) const  [inline]
```

Definition at line 523 of file image.h.

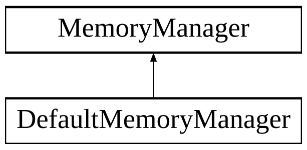
The documentation for this class was generated from the following file:

- **image.h**

8.8 DefaultMemoryManager Class Reference

```
#include <memorymanager.h>
```

Inheritance diagram for DefaultMemoryManager:



Public Member Functions

- virtual uint8_t ∗ **Allocate** (size_t nbytes) override
- virtual void **Deallocate** (uint8_t ∗data) override
- virtual uint8_t ∗ **AllocateAndCopy** (size_t nbytes, uint8_t ∗src) override

Static Public Member Functions

- static **DefaultMemoryManager** ∗ **GetInstance** ()

8.8.1 Detailed Description

Definition at line 16 of file memorymanager.h.

8.8.2 Member Function Documentation

8.8.2.1 Allocate()

```
virtual uint8_t* DefaultMemoryManager::Allocate (
             size_t nbytes )  [inline], [override], [virtual]
```

Implements **MemoryManager** (p. 71).

Definition at line 18 of file memorymanager.h.

8.8.2.2 AllocateAndCopy()

```
virtual uint8_t* DefaultMemoryManager::AllocateAndCopy (
             size_t nbytes,
             uint8_t * src )  [inline], [override], [virtual]
```

Implements **MemoryManager** (p. 71).

Definition at line 24 of file memorymanager.h.

8.8.2.3 Deallocate()

```
virtual void DefaultMemoryManager::Deallocate (
             uint8_t * data )  [inline], [override], [virtual]
```

Implements **MemoryManager** (p. 72).

Definition at line 21 of file memorymanager.h.

8.8.2.4 GetInstance()

```
DefaultMemoryManager * DefaultMemoryManager::GetInstance ( )  [static]
```

Definition at line 3 of file memorymanager.cpp.

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

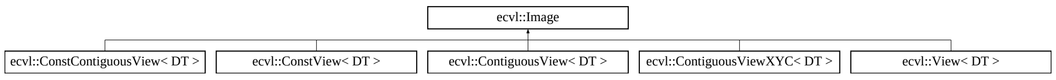
- **memorymanager.h**
- **memorymanager.cpp**

8.9 ecvl::Image Class Reference

Image (p. 57) class.

`#include <image.h>`

Inheritance diagram for ecvl::Image:


ecvl::Image
ecvl::ConstContiguousView< DT >
ecvl::ConstView< DT >
ecvl::ContiguousView< DT >
ecvl::ContiguousViewXYC< DT >
ecvl::View< DT >

Public Member Functions

- template<typename T >
  Iterator< T > **Begin** ()

  *Generic non-const Begin **Iterator** (p. 68).*
- template<typename T >
  Iterator< T > **End** ()

  *Generic non-const End **Iterator** (p. 68).*
- template<typename T >
  ConstIterator< T > **Begin** () const

  *Generic const Begin **Iterator** (p. 68).*
- template<typename T >
  ConstIterator< T > **End** () const

  *Generic const End **Iterator** (p. 68).*
- template<typename T >
  ContiguousIterator< T > **ContiguousBegin** ()

  *Contiguous non-const Begin **Iterator** (p. 68).*
- template<typename T >
  ContiguousIterator< T > **ContiguousEnd** ()

  *Contiguous non-const End **Iterator** (p. 68).*
- template<typename T >
  ConstContiguousIterator< T > **ContiguousBegin** () const

  *Contiguous const Begin **Iterator** (p. 68).*
- template<typename T >
  ConstContiguousIterator< T > **ContiguousEnd** () const

  *Contiguous const End **Iterator** (p. 68).*
- **Image** ()

Default constructor.

- **Image** (const std::vector< int > &dims, **DataType** elemtype, std::string channels, **ColorType** colortype)

  Initializing constructor.
- **Image** (const **Image** &img)

  Copy constructor.
- **Image** (**Image** &&img)

  Move constructor.
- **Image** & **operator=** (**Image** rhs)
- void **Create** (const std::vector< int > &dims, **DataType** elemtype, std::string channels, **ColorType** colortype)

  Allocates new contiguous data if needed.
- ~**Image** ()

  Destructor.
- bool **IsEmpty** () const

  *To check whether the **Image** (p. 57) contains or not data, regardless the owning status.*
- bool **IsOwner** () const

  *To check whether the **Image** (p. 57) is owner of the data.*
- uint8_t ∗ **Ptr** (const std::vector< int > &coords)

  Returns a non-const pointer to data at given coordinates.
- const uint8_t ∗ **Ptr** (const std::vector< int > &coords) const

  Returns a const pointer to data at given coordinates.

Public Attributes

- **DataType** **elemtype_**

  *Type of **Image** (p. 57) pixels, must be one of the values available in **DataType** (p. 17).*
- uint8_t **elemsize_**

  *Size (in bytes) of **Image** (p. 57) pixels.*
- std::vector< int > **dims_**

  *Vector of **Image** (p. 57) dimensions. Each dimension is given in pixels/voxels.*
- std::vector< int > **strides_**

  *Vector of **Image** (p. 57) strides.*
- std::string **channels_**

  *String which describes how **Image** (p. 57) planes are organized.*
- **ColorType** **colortype_**

  ***Image** (p. 57) ColorType.*
- uint8_t ∗ **data_**

  *Pointer to **Image** (p. 57) data.*
- size_t **datasize_**

  *Size of **Image** (p. 57) data in bytes.*
- bool **contiguous_**

  Whether the image is stored contiguously or not in memory.
- **MetaData** ∗ **meta_**

  *Pointer to **Image** (p. 57) **MetaData** (p. 72).*
- **MemoryManager** ∗ **mem_**

  *Pointer to the **MemoryManager** (p. 71) employed by the **Image** (p. 57).*

Friends

- void **swap** (**Image** &lhs, **Image** &rhs)

8.9.1 Detailed Description

Image (p. 57) class.

Definition at line 39 of file image.h.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 Image() [1/4]

```
ecvl::Image::Image ( )  [inline]
```

Default constructor.

The default constructor creates an empty image without any data.

Definition at line 172 of file image.h.

8.9.2.2 Image() [2/4]

```
ecvl::Image::Image (
             const std::vector< int > & dims,
              DataType elemtype,
             std::string channels,
              ColorType colortype )  [inline]
```

Initializing constructor.

The initializing constructor creates a proper image and allocates the data.

Definition at line 191 of file image.h.

8.9.2.3 Image() [3/4]

```
ecvl::Image::Image (
             const  Image & img )  [inline]
```

Copy constructor.

The copy constructor creates an new **Image** (p. 57) copying (Deep Copy) the input one. The new **Image** (p. 57) will be contiguous regardless of the contiguity of the to be copied **Image** (p. 57).

Definition at line 222 of file image.h.

8.9.2.4 Image() [4/4]

```
ecvl::Image::Image (
              Image && img )  [inline]
```

Move constructor.

Move constructor

Definition at line 272 of file image.h.

8.9.2.5 ∼Image()

```
ecvl::Image::∼Image ( )  [inline]
```

Destructor.

If the **Image** (p. 57) is the owner of data they will be deallocate. Otherwise nothing will happen.

Definition at line 327 of file image.h.

8.9.3 Member Function Documentation

8.9.3.1 Begin() [1/2]

```
template<typename T >
Iterator<T> ecvl::Image::Begin ( )  [inline]
```

Generic non-const Begin **Iterator** (p. 68).

This function gives you a non-const generic Begin **Iterator** (p. 68) that can be used both for contiguous and non-contiguous non-const Images. It is useful to iterate over a non-const **Image** (p. 57). If the **Image** (p. 57) is contiguous prefer the use of ContiguousIterato which in most cases improve the performance.

Definition at line 108 of file image.h.

8.9.3.2 Begin() [2/2]

```
template<typename T >
ConstIterator<T> ecvl::Image::Begin ( ) const  [inline]
```

Generic const Begin **Iterator** (p. 68).

This function gives you a const generic Begin **Iterator** (p. 68) that can be used both for contiguous and non-contiguous const Images. It is useful to iterate over a const **Image** (p. 57). If the **Image** (p. 57) is contiguous prefer the use of **ConstContiguousIterator** (p. 39) which in most cases improve the performance.

Definition at line 125 of file image.h.

8.9.3.3 ContiguousBegin() [1/2]

```
template<typename T >
ContiguousIterator<T> ecvl::Image::ContiguousBegin ( )  [inline]
```

Contiguous non-const Begin **Iterator** (p. 68).

This function gives you a contiguous non-const Begin **Iterator** (p. 68) that can be used only for contiguous Images. If the **Image** (p. 57) is contiguous it is preferable to the non-contiguous iterator since it has usually better performance.

Definition at line 142 of file image.h.

8.9.3.4 ContiguousBegin() [2/2]

```
template<typename T >
ConstContiguousIterator<T> ecvl::Image::ContiguousBegin ( ) const  [inline]
```

Contiguous const Begin **Iterator** (p. 68).

This function gives you a contiguous const Begin **Iterator** (p. 68) that can be used only for contiguous Images. If the **Image** (p. 57) is contiguous it is preferable to the non-contiguous iterator since it has usually better performance.

Definition at line 159 of file image.h.

8.9.3.5 ContiguousEnd() [1/2]

```
template<typename T >
ContiguousIterator<T> ecvl::Image::ContiguousEnd ( )  [inline]
```

Contiguous non-const End **Iterator** (p. 68).

This function gives you a contiguous non-const End **Iterator** (p. 68) that can be used only for contiguous Images.

Definition at line 150 of file image.h.

8.9.3.6 ContiguousEnd() [2/2]

```
template<typename T >
ConstContiguousIterator<T> ecvl::Image::ContiguousEnd ( ) const  [inline]
```

Contiguous const End **Iterator** (p. 68).

This function gives you a contiguous const End **Iterator** (p. 68) that can be used only for contiguous Images.

Definition at line 166 of file image.h.

8.9.3.7 Create()

```
void ecvl::Image::Create (
            const std::vector< int > & dims,
             DataType elemtype,
            std::string channels,
             ColorType colortype )
```

Allocates new contiguous data if needed.

The Create method allocates **Image** (p. 57) data as specified by the input parameters. The procedures tries to avoid the allocation of new memory when possible. The resulting image will be contiguous in any case. Calling this method on an **Image** (p. 57) that does not own data will always cause a new allocation, and the **Image** (p. 57) will become the owner of the data.

Parameters

Parameters

in	*dims*	New **Image** (p. 57) dimensions.
in	*elemtype*	New **Image** (p. 57) DataType.
in	*channels*	New **Image** (p. 57) channels.
in	*colortype*	New **Image** (p. 57) colortype.

Definition at line 8 of file image.cpp.

8.9.3.8 End() [1/2]

```
template<typename T >
Iterator<T> ecvl::Image::End ( )  [inline]
```

Generic non-const End **Iterator** (p. 68).

This function gives you a non-const generic End **Iterator** (p. 68) that can be used both for contiguous and non-contiguous non-const Images. It is useful to iterate over over a non-const **Image** (p. 57).

Definition at line 116 of file image.h.

8.9.3.9 End() [2/2]

```
template<typename T >
ConstIterator<T> ecvl::Image::End ( ) const  [inline]
```

Generic const End **Iterator** (p. 68).

This function gives you a const generic End **Iterator** (p. 68) that can be used both for contiguous and non-contiguous const Images. It is useful to iterate over a const **Image** (p. 57).

Definition at line 133 of file image.h.

8.9.3.10 IsEmpty()

```
bool ecvl::Image::IsEmpty ( ) const  [inline]
```

To check whether the **Image** (p. 57) contains or not data, regardless the owning status.

Definition at line 333 of file image.h.

8.9.3.11 IsOwner()

```
bool ecvl::Image::IsOwner ( ) const  [inline]
```

To check whether the **Image** (p. 57) is owner of the data.

Definition at line 336 of file image.h.

8.9.3.12 operator=()

```
 Image& ecvl::Image::operator= (
             Image rhs )  [inline]
```

Definition at line 303 of file image.h.

8.9.3.13 Ptr() [1/2]

```
uint8_t* ecvl::Image::Ptr (
             const std::vector< int > & coords )  [inline]
```

Returns a non-const pointer to data at given coordinates.

Definition at line 339 of file image.h.

8.9.3.14 Ptr() [2/2]

```
const uint8_t* ecvl::Image::Ptr (
             const std::vector< int > & coords ) const  [inline]
```

Returns a const pointer to data at given coordinates.

Definition at line 344 of file image.h.

8.9.4 Friends And Related Function Documentation

8.9.4.1 swap

```
void swap (
              Image & lhs,
              Image & rhs )  [friend]
```

Definition at line 288 of file image.h.

8.9.5 Member Data Documentation

8.9.5.1 channels_

```
std::string ecvl::Image::channels_
```

String which describes how **Image** (p. 57) planes are organized.

A single character provides the information related to the corresponding channel. The possible values are:

- 'x': horizontal spatial dimension
- 'y': vertical spatial dimension
- 'z': depth spatial dimension
- 'c': color dimension
- 't': temporal dimension
- 'o': any other dimension For example, "xyc" describes a 2-dimensional **Image** (p. 57) structured in color planes. This could be for example a **ColorType::GRAY** (p. 16) **Image** (p. 57) with dims_[2] = 1 or a **Color↩ Type::RGB** (p. 16) **Image** (p. 57) with dims_[2] = 3 an so on. The ColorType constrains the value of the dimension corresponding to the color channel. Another example is "cxy" with dims_[0] = 3 and **ColorType↩ ::BGR** (p. 16). In this case the color dimension is the one which changes faster as it is done in other libraries such as OpenCV.

Definition at line 52 of file image.h.

8.9.5.2 colortype_

```
ColorType ecvl::Image::colortype_
```

Image (p. 57) ColorType.

If this is different from **ColorType::none** (p. 17) the channels_ string must contain a 'c' and the corresponding dimension must have the appropriate value. See **ColorType** (p. 16) for the possible values.

Definition at line 75 of file image.h.

8.9.5.3 contiguous_

```
bool ecvl::Image::contiguous_
```

Whether the image is stored contiguously or not in memory.

Definition at line 89 of file image.h.

8.9.5.4 data_

```
uint8_t* ecvl::Image::data_
```

Pointer to **Image** (p. 57) data.

If the **Image** (p. 57) is not the owner of data, for example when using **Image** (p. 57) views, this attribute will point to the data of another **Image** (p. 57). The possession or not of the data depends on the **MemoryManager** (p. 71).

Definition at line 81 of file image.h.

8.9.5.5 datasize_

```
size_t ecvl::Image::datasize_
```

Size of **Image** (p. 57) data in bytes.

Definition at line 88 of file image.h.

8.9.5.6 dims_

```
std::vector<int> ecvl::Image::dims_
```

Vector of **Image** (p. 57) dimensions. Each dimension is given in pixels/voxels.

Definition at line 44 of file image.h.

8.9.5.7 elemsize_

```
uint8_t ecvl::Image::elemsize_
```

Size (in bytes) of **Image** (p. 57) pixels.

Definition at line 43 of file image.h.

8.9.5.8 elemtype_

```
DataType ecvl::Image::elemtype_
```

Type of **Image** (p. 57) pixels, must be one of the values available in **DataType** (p. 17).

Definition at line 41 of file image.h.

8.9.5.9 mem_

```
MemoryManager* ecvl::Image::mem_
```

Pointer to the **MemoryManager** (p. 71) employed by the **Image** (p. 57).

It can be **DefaultMemoryManager** (p. 55) or **ShallowMemoryManager** (p. 75). The former is responsible for allocating and deallocating data, when using the **DefaultMemoryManager** (p. 55) the **Image** (p. 57) is the owner of data. When **ShallowMemoryManager** (p. 75) is employed the **Image** (p. 57) does not own data and operations on memory are not allowed or does not produce any effect.

Definition at line 92 of file image.h.

8.9.5.10 meta_

```
MetaData* ecvl::Image::meta_
```

Pointer to **Image** (p. 57) **MetaData** (p. 72).

Definition at line 91 of file image.h.

8.9.5.11 strides_

```
std::vector<int> ecvl::Image::strides_
```

Vector of **Image** (p. 57) strides.

```
Strides represent
the number of bytes the pointer on data
has to move to reach the next pixel/voxel
on the correspondent size.
```

Definition at line 46 of file image.h.

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

- **image.h**
- **image.cpp**

8.10 ecvl::SignedTable1D< _StructFun, Args >::integer< i > Struct Template Reference

```
#include <datatype_matrix.h>
```

8.10.1 Detailed Description

template<template< DataType, typename ... >class _StructFun, typename ... Args>
template<int i>
struct ecvl::SignedTable1D< _StructFun, Args >::integer< i >

Definition at line 44 of file datatype_matrix.h.

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

- **datatype_matrix.h**

8.11 ecvl::Table1D< _StructFun, Args >::integer< i > Struct Template Reference

```
#include <datatype_matrix.h>
```

8.11.1 Detailed Description

template<template< DataType DT, typename ... >class _StructFun, typename ... Args>
template<int i>
struct ecvl::Table1D< _StructFun, Args >::integer< i >

Definition at line 15 of file datatype_matrix.h.

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

- **datatype_matrix.h**

8.12 ecvl::Table2D< _StructFun, Args >::integer< i > Struct Template Reference

```
#include <datatype_matrix.h>
```

8.12.1 Detailed Description

template<template< DataType, DataType, typename ... >class _StructFun, typename ... Args>
template<int i>
struct ecvl::Table2D< _StructFun, Args >::integer< i >

Definition at line 73 of file datatype_matrix.h.

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

- **datatype_matrix.h**

8.13 ecvl::Iterator< T > Struct Template Reference

```
#include <iterators.h>
```

Public Types

- typedef **Iterator** &(Iterator::∗ **IncrementMemFn**) ()

Public Member Functions

- **Iterator** (**Image** &img, std::vector< int > pos={})
- **Iterator** & **operator++** ()
- T & **operator** ∗ () const
- T ∗ **operator->** () const
- bool **operator==** (const **Iterator** &rhs) const
- bool **operator!=** (const **Iterator** &rhs) const

Public Attributes

- std::vector< int > **pos_**
- uint8_t ∗ **ptr_**
- **Image** ∗ **img_**
- **IncrementMemFn** **incrementor** = & **Iterator**<T>::IncrementPos

8.13.1 Detailed Description

template<typename T>
struct ecvl::Iterator< T >

Definition at line 12 of file iterators.h.

8.13.2 Member Typedef Documentation

8.13.2.1 IncrementMemFn

```
template<typename T >
typedef  Iterator&(Iterator::*  ecvl::Iterator< T >::IncrementMemFn) ()
```

Definition at line 17 of file iterators.h.

8.13.3 Constructor & Destructor Documentation

8.13.3.1 Iterator()

```
template<typename T >
Iterator::Iterator (
              Image & img,
             std::vector< int > pos = {} )
```

Definition at line 4 of file image.h.

8.13.4 Member Function Documentation

8.13.4.1 operator *()

```
template<typename T >
T&  ecvl::Iterator< T >::operator * ( ) const  [inline]
```

Definition at line 22 of file iterators.h.

8.13.4.2 operator"!=()

```
template<typename T >
bool  ecvl::Iterator< T >::operator!= (
             const  Iterator< T > & rhs ) const  [inline]
```

Definition at line 25 of file iterators.h.

8.13.4.3 operator++()

```
template<typename T >
Iterator&  ecvl::Iterator< T >::operator++ ( )  [inline]
```

Definition at line 21 of file iterators.h.

8.13.4.4 operator->()

```
template<typename T >
T*  ecvl::Iterator< T >::operator-> ( ) const  [inline]
```

Definition at line 23 of file iterators.h.

8.13.4.5 operator==()

```
template<typename T >
bool ecvl::Iterator< T >::operator== (
            const Iterator< T > & rhs ) const [inline]
```

Definition at line 24 of file iterators.h.

8.13.5 Member Data Documentation

8.13.5.1 img_

```
template<typename T >
Image* ecvl::Iterator< T >::img_
```

Definition at line 15 of file iterators.h.

8.13.5.2 incrementor

```
template<typename T >
IncrementMemFn ecvl::Iterator< T >::incrementor = & Iterator<T>::IncrementPos
```

Definition at line 18 of file iterators.h.

8.13.5.3 pos_

```
template<typename T >
std::vector<int> ecvl::Iterator< T >::pos_
```

Definition at line 13 of file iterators.h.

8.13.5.4 ptr_

```
template<typename T >
uint8_t* ecvl::Iterator< T >::ptr_
```

Definition at line 14 of file iterators.h.

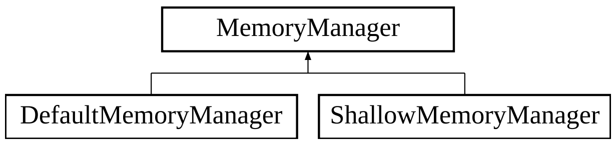
The documentation for this struct was generated from the following files:

- **iterators.h**
- **image.h**
- **iterators_impl.inc.h**

8.14 MemoryManager Class Reference

```
#include <memorymanager.h>
```

Inheritance diagram for MemoryManager:



Public Member Functions

- virtual uint8_t ∗ **Allocate** (size_t nbytes)=0
- virtual void **Deallocate** (uint8_t ∗data)=0
- virtual uint8_t ∗ **AllocateAndCopy** (size_t nbytes, uint8_t ∗src)=0
- virtual ∼**MemoryManager** ()

8.14.1 Detailed Description

Definition at line 8 of file memorymanager.h.

8.14.2 Constructor & Destructor Documentation

8.14.2.1 ∼MemoryManager()

```
virtual MemoryManager::∼MemoryManager ( )  [inline], [virtual]
```

Definition at line 13 of file memorymanager.h.

8.14.3 Member Function Documentation

8.14.3.1 Allocate()

```
virtual uint8_t* MemoryManager::Allocate (
            size_t nbytes )  [pure virtual]
```

Implemented in **ShallowMemoryManager** (p. 76), and **DefaultMemoryManager** (p. 56).

8.14.3.2 AllocateAndCopy()

```
virtual uint8_t* MemoryManager::AllocateAndCopy (
             size_t nbytes,
             uint8_t * src )  [pure virtual]
```

Implemented in **ShallowMemoryManager** (p. 76), and **DefaultMemoryManager** (p. 56).

8.14.3.3 Deallocate()

```
virtual void MemoryManager::Deallocate (
             uint8_t * data )  [pure virtual]
```

Implemented in **ShallowMemoryManager** (p. 76), and **DefaultMemoryManager** (p. 56).

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

- **memorymanager.h**

8.15 ecvl::MetaData Class Reference

```
#include <image.h>
```

Public Member Functions

- virtual bool **Query** (const std::string &name, std::string &value) const =0
- virtual ∼**MetaData** ()

8.15.1 Detailed Description

Definition at line 17 of file image.h.

8.15.2 Constructor & Destructor Documentation

8.15.2.1 ∼MetaData()

```
virtual ecvl::MetaData::∼MetaData ( )  [inline], [virtual]
```

Definition at line 20 of file image.h.

8.15.3 Member Function Documentation

8.15.3.1 Query()

```
virtual bool ecvl::MetaData::Query (
            const std::string & name,
            std::string & value ) const  [pure virtual]
```

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

- **image.h**

8.16 filesystem::path Class Reference

```
#include <filesystem.h>
```

Public Member Functions

- **path** ()
- **path** (const std::string &p)
- **path** & **operator/=** (const **path** &p)
- **path** & **operator=** (const std::string &s)
- **path** & **operator=** (const **path** &p)
- std::string **string** () const
- **path** **parent_path** () const
- **path** **stem** () const

8.16.1 Detailed Description

Definition at line 9 of file filesystem.h.

8.16.2 Constructor & Destructor Documentation

8.16.2.1 path() [1/2]

```
filesystem::path::path ( )  [inline]
```

Definition at line 12 of file filesystem.h.

8.16.2.2 path() [2/2]

```
filesystem::path::path (
            const std::string & p )  [inline], [explicit]
```

Definition at line 14 of file filesystem.h.

8.16.3 Member Function Documentation

8.16.3.1 operator/=()

```
 path& filesystem::path::operator/= (
            const  path & p )  [inline]
```

Definition at line 20 of file filesystem.h.

8.16.3.2 operator=() [1/2]

```
 path& filesystem::path::operator= (
            const std::string & s )  [inline]
```

Definition at line 48 of file filesystem.h.

8.16.3.3 operator=() [2/2]

```
 path& filesystem::path::operator= (
            const  path & p )  [inline]
```

Definition at line 55 of file filesystem.h.

8.16.3.4 parent_path()

```
 path filesystem::path::parent_path ( ) const  [inline]
```

Definition at line 66 of file filesystem.h.

8.16.3.5 stem()

```
path filesystem::path::stem ( ) const  [inline]
```

Definition at line 80 of file filesystem.h.

8.16.3.6 string()

```
std::string filesystem::path::string ( ) const  [inline]
```

Definition at line 61 of file filesystem.h.

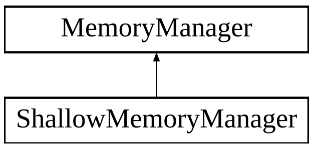
The documentation for this class was generated from the following files:

- **filesystem.h**
- **filesystem.cc**

8.17 ShallowMemoryManager Class Reference

```
#include <memorymanager.h>
```

Inheritance diagram for ShallowMemoryManager:



Public Member Functions

- virtual uint8_t ∗ **Allocate** (size_t nbytes) override
- virtual void **Deallocate** (uint8_t ∗data) override
- virtual uint8_t ∗ **AllocateAndCopy** (size_t nbytes, uint8_t ∗src) override

Static Public Member Functions

- static **ShallowMemoryManager** ∗ **GetInstance** ()

8.17.1 Detailed Description

Definition at line 31 of file memorymanager.h.

8.17.2 Member Function Documentation

8.17.2.1 Allocate()

```
virtual uint8_t* ShallowMemoryManager::Allocate (
             size_t nbytes )  [inline], [override], [virtual]
```

Implements **MemoryManager** (p. 71).

Definition at line 33 of file memorymanager.h.

8.17.2.2 AllocateAndCopy()

```
virtual uint8_t* ShallowMemoryManager::AllocateAndCopy (
             size_t nbytes,
             uint8_t * src )  [inline], [override], [virtual]
```

Implements **MemoryManager** (p. 71).

Definition at line 37 of file memorymanager.h.

8.17.2.3 Deallocate()

```
virtual void ShallowMemoryManager::Deallocate (
             uint8_t * data )  [inline], [override], [virtual]
```

Implements **MemoryManager** (p. 72).

Definition at line 36 of file memorymanager.h.

8.17.2.4 GetInstance()

```
ShallowMemoryManager * ShallowMemoryManager::GetInstance ( )  [static]
```

Definition at line 10 of file memorymanager.cpp.

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

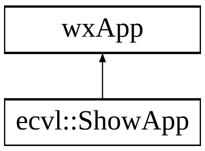
- **memorymanager.h**
- **memorymanager.cpp**

8.18 ecvl::ShowApp Class Reference

ShowApp (p. 77) is a custom wxApp which allows you to visualize an ECVL **Image** (p. 57).

```
#include <gui.h>
```

Inheritance diagram for ecvl::ShowApp:



Public Member Functions

- bool **OnInit** ()
  Initialization function. Starts the main loop of the application.
- **ShowApp** (const **Image** &img)
  Constructor.

8.18.1 Detailed Description

ShowApp (p. 77) is a custom wxApp which allows you to visualize an ECVL **Image** (p. 57).

Definition at line 32 of file gui.h.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 ShowApp()

```
ecvl::ShowApp::ShowApp (
            const  Image & img )  [inline]
```

Constructor.

The constructor creates a **ShowApp** (p. 77) initializing its **Image** (p. 57) with the given input **Image** (p. 57).

Definition at line 50 of file gui.h.

8.18.3 Member Function Documentation

8.18.3.1 OnInit()

```
bool ecvl::ShowApp::OnInit ( )
```

Initialization function. Starts the main loop of the application.

The **OnInit()** (p. 77) function creates a wxFrame which has the width and the height of the **Image** (p. 57) that has to be shown. It also creates the **wxImagePanel** (p. 103) which contains the frame and employs the conversion from **Image** (p. 57) to wxImage. It set the wxImage in the frame and starts the main loop of the **ShowApp** (p. 77).

Definition at line 72 of file gui.cpp.

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

- **gui.h**
- **gui.cpp**

8.19 ecvl::SignedTable1D< _StructFun, Args > Struct Template Reference

```
#include <datatype_matrix.h>
```

Classes

- struct **integer**

Public Types

- using **fun_type** = decltype(&_StructFun< static_cast< **DataType** >(0), Args... >::ActualFunction)

Public Member Functions

- template<int i>
  constexpr void **FillData** (**integer**< i >)
- constexpr void **FillData** (**integer**< **DataTypeSignedSize**()>)
- constexpr **SignedTable1D** ()
- **fun_type** **operator()** (**DataType** dt) const

Public Attributes

- **fun_type** **data** [**DataTypeSignedSize**()]

8.19.1 Detailed Description

template<template< DataType, typename ... >class _StructFun, typename ... Args>
struct ecvl::SignedTable1D< _StructFun, Args >

Definition at line 39 of file datatype_matrix.h.

8.19.2 Member Typedef Documentation

8.19.2.1 fun_type

```
template<template< DataType, typename ...  >class _StructFun, typename ...  Args>
using  ecvl::SignedTable1D< _StructFun, Args >::  fun_type = decltype(&_StructFun<static_↩
cast< DataType>(0), Args...>::ActualFunction)
```

Definition at line 41 of file datatype_matrix.h.

8.19.3 Constructor & Destructor Documentation

8.19.3.1 SignedTable1D()

```
template<template< DataType, typename ...  >class _StructFun, typename ...  Args>
constexpr  ecvl::SignedTable1D< _StructFun, Args >::  SignedTable1D ( )  [inline]
```

Definition at line 55 of file datatype_matrix.h.

8.19.4 Member Function Documentation

8.19.4.1 FillData() [1/2]

```
template<template< DataType, typename ...  >class _StructFun, typename ...  Args>
template<int i>
constexpr void  ecvl::SignedTable1D< _StructFun, Args >::FillData (
              integer< i >  )  [inline]
```

Definition at line 47 of file datatype_matrix.h.

8.19.4.2 FillData() [2/2]

```
template<template< DataType, typename ...  >class _StructFun, typename ...  Args>
constexpr void  ecvl::SignedTable1D< _StructFun, Args >::FillData (
              integer<  DataTypeSignedSize()>  )  [inline]
```

Definition at line 53 of file datatype_matrix.h.

8.19.4.3 operator()()

```
template<template< DataType, typename ...  >class _StructFun, typename ...  Args>
fun_type  ecvl::SignedTable1D< _StructFun, Args >::operator() (
             DataType dt ) const  [inline]
```

Definition at line 59 of file datatype_matrix.h.

8.19.5 Member Data Documentation

8.19.5.1 data

```
template<template< DataType, typename ...  >class _StructFun, typename ...  Args>
fun_type  ecvl::SignedTable1D< _StructFun, Args >::data[ DataTypeSignedSize()]
```

Definition at line 63 of file datatype_matrix.h.

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

- **datatype_matrix.h**

8.20 ecvl::StructAdd< a, b > Struct Template Reference

Static Public Member Functions

- static void **ActualFunction** (**Image** &src1_dst, const **Image** &src2)

8.20.1 Detailed Description

template<DataType a, DataType b>
struct ecvl::StructAdd< a, b >

Definition at line 23 of file arithmetic.cpp.

8.20.2 Member Function Documentation

8.20.2.1 ActualFunction()

```
template<DataType a, DataType b>
static void  ecvl::StructAdd< a, b >::ActualFunction (
              Image & src1_dst,
             const  Image & src2 )  [inline], [static]
```

Definition at line 24 of file arithmetic.cpp.

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

- **arithmetic.cpp**

8.21 ecvl::StructCopyImage< SDT, DDT > Struct Template Reference

```
#include <image.h>
```

Static Public Member Functions

- static void **ActualFunction** (const **Image** &src, **Image** &dst)

8.21.1 Detailed Description

template<DataType SDT, DataType DDT>
struct ecvl::StructCopyImage< SDT, DDT >

Definition at line 552 of file image.h.

8.21.2 Member Function Documentation

8.21.2.1 ActualFunction()

```
template<DataType SDT, DataType DDT>
static void  ecvl::StructCopyImage< SDT, DDT >::ActualFunction (
             const  Image & src,
              Image & dst )  [inline], [static]
```

Definition at line 553 of file image.h.

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

- **image.h**

8.22 ecvl::StructDiv< a, b, ET > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void **ActualFunction** (**Image** &src1_dst, const **Image** &src2, bool saturate, ET epsilon)

8.22.1 Detailed Description

template<DataType a, DataType b, typename ET>
struct ecvl::StructDiv< a, b, ET >

Definition at line 73 of file arithmetic.h.

8.22.2 Member Function Documentation

8.22.2.1 ActualFunction()

```
template<DataType a, DataType b, typename ET >
static void  ecvl::StructDiv< a, b, ET >::ActualFunction (
              Image & src1_dst,
             const  Image & src2,
             bool saturate,
             ET epsilon )  [inline], [static]
```

Definition at line 74 of file arithmetic.h.

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

- **arithmetic.h**

8.23 ecvl::StructMul< a, b > Struct Template Reference

Static Public Member Functions

- static void **ActualFunction** (**Image** &src1_dst, const **Image** &src2)

8.23.1 Detailed Description

template<DataType a, DataType b>
struct ecvl::StructMul< a, b >

Definition at line 57 of file arithmetic.cpp.

8.23.2 Member Function Documentation

8.23.2.1 ActualFunction()

```
template<DataType a, DataType b>
static void  ecvl::StructMul< a, b >::ActualFunction (
              Image & src1_dst,
             const  Image & src2 )  [inline], [static]
```

Definition at line 58 of file arithmetic.cpp.

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

- **arithmetic.cpp**

8.24 ecvl::StructScalarAdd< DT, T > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static **Image** & **ActualFunction** (**Image** &img, T value, bool saturate)

8.24.1 Detailed Description

template<DataType DT, typename T>
struct ecvl::StructScalarAdd< DT, T >

Definition at line 144 of file arithmetic.h.

8.24.2 Member Function Documentation

8.24.2.1 ActualFunction()

```
template<DataType DT, typename T >
static  Image&  ecvl::StructScalarAdd< DT, T >::ActualFunction (
              Image & img,
             T value,
             bool saturate )  [inline], [static]
```

Definition at line 145 of file arithmetic.h.

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

- **arithmetic.h**

8.25 ecvl::StructScalarDiv< DT, T > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static **Image** & **ActualFunction** (**Image** &img, T value, bool saturate)

8.25.1 Detailed Description

template<DataType DT, typename T>
struct ecvl::StructScalarDiv< DT, T >

Definition at line 279 of file arithmetic.h.

8.25.2 Member Function Documentation

8.25.2.1 ActualFunction()

```
template<DataType DT, typename T >
static  Image&  ecvl::StructScalarDiv< DT, T >::ActualFunction (
               Image & img,
              T value,
              bool saturate )  [inline], [static]
```

Definition at line 280 of file arithmetic.h.

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

- **arithmetic.h**

8.26 ecvl::StructScalarDivInv< DT, T, ET > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static **Image** & **ActualFunction** (T value, **Image** &img, bool saturate, ET epsilon)

8.26.1 Detailed Description

template<DataType DT, typename T, typename ET>
struct ecvl::StructScalarDivInv< DT, T, ET >

Definition at line 322 of file arithmetic.h.

8.26.2 Member Function Documentation

8.26.2.1 ActualFunction()

```
template<DataType DT, typename T , typename ET >
static  Image&  ecvl::StructScalarDivInv< DT, T, ET >::ActualFunction (
             T value,
              Image & img,
             bool saturate,
             ET epsilon )  [inline], [static]
```

Definition at line 323 of file arithmetic.h.

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

- **arithmetic.h**

8.27 ecvl::StructScalarMul< DT, T > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static **Image** & **ActualFunction** (**Image** &img, T d, bool saturate)

8.27.1 Detailed Description

template<DataType DT, typename T>
struct ecvl::StructScalarMul< DT, T >

Definition at line 95 of file arithmetic.h.

8.27.2 Member Function Documentation

8.27.2.1 ActualFunction()

```
template<DataType DT, typename T >
static  Image&  ecvl::StructScalarMul< DT, T >::ActualFunction (
              Image & img,
             T d,
             bool saturate )  [inline], [static]
```

Definition at line 96 of file arithmetic.h.

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

- **arithmetic.h**

8.28 ecvl::StructScalarNeg< DT > Struct Template Reference

Static Public Member Functions

- static **Image** & **ActualFunction** (**Image** &img)

8.28.1 Detailed Description

template<DataType DT>
struct ecvl::StructScalarNeg< DT >

Definition at line 79 of file arithmetic.cpp.

8.28.2 Member Function Documentation

8.28.2.1 ActualFunction()

```
template<DataType DT>
static  Image&  ecvl::StructScalarNeg< DT >::ActualFunction (
              Image & img )  [inline], [static]
```

Definition at line 80 of file arithmetic.cpp.

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

- **arithmetic.cpp**

8.29 ecvl::StructScalarSub< DT, T > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static **Image** & **ActualFunction** (**Image** &img, T value, bool saturate)

8.29.1 Detailed Description

template<DataType DT, typename T>
struct ecvl::StructScalarSub< DT, T >

Definition at line 193 of file arithmetic.h.

8.29.2 Member Function Documentation

8.29.2.1 ActualFunction()

```
template<DataType DT, typename T >
static  Image&  ecvl::StructScalarSub< DT, T >::ActualFunction (
              Image & img,
             T value,
             bool saturate )  [inline], [static]
```

Definition at line 194 of file arithmetic.h.

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

- **arithmetic.h**

8.30 ecvl::StructScalarSubInv< DT, T > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static **Image** & **ActualFunction** (T value, **Image** &img, bool saturate)

8.30.1 Detailed Description

template<DataType DT, typename T>
struct ecvl::StructScalarSubInv< DT, T >

Definition at line 236 of file arithmetic.h.

8.30.2 Member Function Documentation

8.30.2.1 ActualFunction()

```
template<DataType DT, typename T >
static  Image&  ecvl::StructScalarSubInv< DT, T >::ActualFunction (
             T value,
              Image & img,
             bool saturate )  [inline], [static]
```

Definition at line 237 of file arithmetic.h.

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

- **arithmetic.h**

8.31 ecvl::StructSub< a, b > Struct Template Reference

Static Public Member Functions

- static void **ActualFunction** (**Image** &src1_dst, const **Image** &src2)

8.31.1 Detailed Description

template<DataType a, DataType b>
struct ecvl::StructSub< a, b >

Definition at line 40 of file arithmetic.cpp.

8.31.2 Member Function Documentation

8.31.2.1 ActualFunction()

```
template<DataType a, DataType b>
static void  ecvl::StructSub< a, b >::ActualFunction (
              Image & src1_dst,
             const  Image & src2 )  [inline], [static]
```

Definition at line 41 of file arithmetic.cpp.

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

- **arithmetic.cpp**

8.32 ecvl::Table1D< _StructFun, Args > Struct Template Reference

```
#include <datatype_matrix.h>
```

Classes

- struct **integer**

Public Types

- using **fun_type** = decltype(&_StructFun< static_cast< **DataType** >(0), Args... >::ActualFunction)

Public Member Functions

- template<int i>
  constexpr void **FillData** (**integer**< i >)
- constexpr void **FillData** (**integer**< **DataTypeSize**()>)
- constexpr **Table1D** ()
- **fun_type** **operator()** (**DataType** dt) const

Public Attributes

- **fun_type** **data** [**DataTypeSize**()]

8.32.1 Detailed Description

template<template< DataType DT, typename ... >class _StructFun, typename ... Args>
struct ecvl::Table1D< _StructFun, Args >

Definition at line 10 of file datatype_matrix.h.

8.32.2 Member Typedef Documentation

8.32.2.1 fun_type

```
template<template< DataType DT, typename ...  >class _StructFun, typename ...  Args>
using  ecvl::Table1D< _StructFun, Args >::  fun_type = decltype(&_StructFun<static_cast<
DataType>(0), Args...>::ActualFunction)
```

Definition at line 12 of file datatype_matrix.h.

8.32.3 Constructor & Destructor Documentation

8.32.3.1 Table1D()

```
template<template< DataType DT, typename ...  >class _StructFun, typename ...  Args>
constexpr  ecvl::Table1D< _StructFun, Args >::  Table1D ( )  [inline]
```

Definition at line 26 of file datatype_matrix.h.

8.32.4 Member Function Documentation

8.32.4.1 FillData() [1/2]

```
template<template< DataType DT, typename ...  >class _StructFun, typename ...  Args>
template<int i>
constexpr void  ecvl::Table1D< _StructFun, Args >::FillData (
              integer< i >  )  [inline]
```

Definition at line 18 of file datatype_matrix.h.

8.32.4.2 FillData() [2/2]

```
template<template< DataType DT, typename ...  >class _StructFun, typename ...  Args>
constexpr void  ecvl::Table1D< _StructFun, Args >::FillData (
              integer<  DataTypeSize()>  )  [inline]
```

Definition at line 24 of file datatype_matrix.h.

8.32.4.3 operator()()

```
template<template< DataType DT, typename ...  >class _StructFun, typename ...  Args>
fun_type  ecvl::Table1D< _StructFun, Args >::operator() (
              DataType dt ) const  [inline]
```

Definition at line 30 of file datatype_matrix.h.

8.32.5 Member Data Documentation

8.32.5.1 data

```
template<template< DataType DT, typename ...  >class _StructFun, typename ...  Args>
fun_type  ecvl::Table1D< _StructFun, Args >::data[ DataTypeSize()]
```

Definition at line 34 of file datatype_matrix.h.

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

- **datatype_matrix.h**

8.33 ecvl::Table2D< _StructFun, Args > Struct Template Reference

```
#include <datatype_matrix.h>
```

Classes

- struct **integer**

Public Types

- using **fun_type** = decltype(&_StructFun< static_cast< **DataType** >(0), static_cast< **DataType** >(0), Args... >::ActualFunction)

Public Member Functions

- template<int i>
  constexpr void **FillData** (**integer**< i >)
- constexpr void **FillData** (**integer**< **DataTypeSize**() ∗ **DataTypeSize**() >)
- constexpr **Table2D** ()
- **fun_type** **operator()** (**DataType** src, **DataType** dst) const

Public Attributes

- **fun_type** **data** [**DataTypeSize**() ∗ **DataTypeSize**()]

8.33.1 Detailed Description

template<template< DataType, DataType, typename ... >class _StructFun, typename ... Args>
struct ecvl::Table2D< _StructFun, Args >

Definition at line 68 of file datatype_matrix.h.

8.33.2 Member Typedef Documentation

8.33.2.1 fun_type

```
template<template< DataType, DataType, typename ...  >class _StructFun, typename ...  Args>
using  ecvl::Table2D< _StructFun, Args >::  fun_type = decltype(&_StructFun<static_cast<
DataType>(0), static_cast< DataType>(0), Args...>::ActualFunction)
```

Definition at line 70 of file datatype_matrix.h.

8.33.3 Constructor & Destructor Documentation

8.33.3.1 Table2D()

```
template<template< DataType, DataType, typename ...  >class _StructFun, typename ...  Args>
constexpr  ecvl::Table2D< _StructFun, Args >::  Table2D ( )  [inline]
```

Definition at line 86 of file datatype_matrix.h.

8.33.4 Member Function Documentation

8.33.4.1 FillData() [1/2]

```
template<template< DataType, DataType, typename ...  >class _StructFun, typename ...  Args>
template<int i>
constexpr void  ecvl::Table2D< _StructFun, Args >::FillData (
              integer< i >  )  [inline]
```

Definition at line 76 of file datatype_matrix.h.

8.33.4.2 FillData() [2/2]

```
template<template< DataType, DataType, typename ...  >class _StructFun, typename ...  Args>
constexpr void  ecvl::Table2D< _StructFun, Args >::FillData (
              integer<  DataTypeSize() * DataTypeSize() >  )  [inline]
```

Definition at line 84 of file datatype_matrix.h.

8.33.4.3 operator()()

```
template<template< DataType, DataType, typename ...  >class _StructFun, typename ...  Args>
fun_type  ecvl::Table2D< _StructFun, Args >::operator() (
              DataType src,
              DataType dst ) const  [inline]
```

Definition at line 90 of file datatype_matrix.h.

8.33.5 Member Data Documentation

8.33.5.1 data

```
template<template< DataType, DataType, typename ...  >class _StructFun, typename ...  Args>
fun_type  ecvl::Table2D< _StructFun, Args >::data[ DataTypeSize() * DataTypeSize()]
```

Definition at line 96 of file datatype_matrix.h.

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

- **datatype_matrix.h**

8.34 ecvl::TypeInfo< DataType > Struct Template Reference

```
#include <datatype.h>
```

Public Types

- using **basetype** = void

8.34.1 Detailed Description

template<ecvl::DataType>
struct ecvl::TypeInfo< DataType >

Definition at line 33 of file datatype.h.

8.34.2 Member Typedef Documentation

8.34.2.1 basetype

```
template<ecvl::DataType >
using ecvl::TypeInfo< DataType >:: basetype = void
```

Definition at line 33 of file datatype.h.

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

- **datatype.h**

8.35 ecvl::TypeInfo< ecvl::DataType::float32 > Struct Template Reference

```
#include <datatype.h>
```

Public Types

- using **basetype** = float

8.35.1 Detailed Description

template<>
struct ecvl::TypeInfo< ecvl::DataType::float32 >

Definition at line 6 of file datatype.h.

8.35.2 Member Typedef Documentation

8.35.2.1 basetype

```
using ecvl::TypeInfo< ecvl::DataType::float32 >:: basetype = float
```

Definition at line 6 of file datatype.h.

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

- **datatype.h**

8.36 ecvl::TypeInfo< ecvl::DataType::float64 > Struct Template Reference

```
#include <datatype.h>
```

Public Types

- using **basetype** = double

8.36.1 Detailed Description

template<>
struct ecvl::TypeInfo< ecvl::DataType::float64 >

Definition at line 7 of file datatype.h.

8.36.2 Member Typedef Documentation

8.36.2.1 basetype

```
using ecvl::TypeInfo< ecvl::DataType::float64 >:: basetype = double
```

Definition at line 7 of file datatype.h.

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

- **datatype.h**

8.37 ecvl::TypeInfo< ecvl::DataType::int16 > Struct Template Reference

```
#include <datatype.h>
```

Public Types

- using **basetype** = int16_t

8.37.1 Detailed Description

template<>
struct ecvl::TypeInfo< ecvl::DataType::int16 >

Definition at line 3 of file datatype.h.

8.37.2 Member Typedef Documentation

8.37.2.1 basetype

```
using ecvl::TypeInfo< ecvl::DataType::int16 >:: basetype = int16_t
```

Definition at line 3 of file datatype.h.

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

- **datatype.h**

8.38 ecvl::TypeInfo< ecvl::DataType::int32 > Struct Template Reference

```
#include <datatype.h>
```

Public Types

- using **basetype** = int32_t

8.38.1 Detailed Description

template<>
struct ecvl::TypeInfo< ecvl::DataType::int32 >

Definition at line 4 of file datatype.h.

8.38.2 Member Typedef Documentation

8.38.2.1 basetype

```
using ecvl::TypeInfo< ecvl::DataType::int32 >:: basetype = int32_t
```

Definition at line 4 of file datatype.h.

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

- **datatype.h**

8.39 ecvl::TypeInfo< ecvl::DataType::int64 > Struct Template Reference

```
#include <datatype.h>
```

Public Types

- using **basetype** = int64_t

8.39.1 Detailed Description

template<>
struct ecvl::TypeInfo< ecvl::DataType::int64 >

Definition at line 5 of file datatype.h.

8.39.2 Member Typedef Documentation

8.39.2.1 basetype

```
using ecvl::TypeInfo< ecvl::DataType::int64 >:: basetype = int64_t
```

Definition at line 5 of file datatype.h.

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

- **datatype.h**

8.40 ecvl::TypeInfo< ecvl::DataType::int8 > Struct Template Reference

```
#include <datatype.h>
```

Public Types

- using **basetype** = int8_t

8.40.1 Detailed Description

template<>
struct ecvl::TypeInfo< ecvl::DataType::int8 >

Definition at line 2 of file datatype.h.

8.40.2 Member Typedef Documentation

8.40.2.1 basetype

```
using ecvl::TypeInfo< ecvl::DataType::int8 >:: basetype = int8_t
```

Definition at line 2 of file datatype.h.

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

- **datatype.h**

8.41 ecvl::TypeInfo< ecvl::DataType::none > Struct Template Reference

```
#include <datatype.h>
```

Public Types

- using **basetype** = void

8.41.1 Detailed Description

template<>
struct ecvl::TypeInfo< ecvl::DataType::none >

Definition at line 7 of file datatype.h.

8.41.2 Member Typedef Documentation

8.41.2.1 basetype

```
using ecvl::TypeInfo< ecvl::DataType::none >:: basetype = void
```

Definition at line 7 of file datatype.h.

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

- **datatype.h**

8.42 ecvl::TypeInfo< ecvl::DataType::uint16 > Struct Template Reference

```
#include <datatype.h>
```

Public Types

- using **basetype** = uint16_t

8.42.1 Detailed Description

template<>
struct ecvl::TypeInfo< ecvl::DataType::uint16 >

Definition at line 3 of file datatype.h.

8.42.2 Member Typedef Documentation

8.42.2.1 basetype

```
using ecvl::TypeInfo< ecvl::DataType::uint16 >:: basetype = uint16_t
```

Definition at line 3 of file datatype.h.

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

- **datatype.h**

8.43 ecvl::TypeInfo< ecvl::DataType::uint32 > Struct Template Reference

```
#include <datatype.h>
```

Public Types

- using **basetype** = uint32_t

8.43.1 Detailed Description

template<>
struct ecvl::TypeInfo< ecvl::DataType::uint32 >

Definition at line 4 of file datatype.h.

8.43.2 Member Typedef Documentation

8.43.2.1 basetype

```
using ecvl::TypeInfo< ecvl::DataType::uint32 >:: basetype = uint32_t
```

Definition at line 4 of file datatype.h.

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

- **datatype.h**

8.44 ecvl::TypeInfo< ecvl::DataType::uint64 > Struct Template Reference

```
#include <datatype.h>
```

Public Types

- using **basetype** = uint64_t

8.44.1 Detailed Description

template<>
struct ecvl::TypeInfo< ecvl::DataType::uint64 >

Definition at line 5 of file datatype.h.

8.44.2 Member Typedef Documentation

8.44.2.1 basetype

```
using ecvl::TypeInfo< ecvl::DataType::uint64 >:: basetype = uint64_t
```

Definition at line 5 of file datatype.h.

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

- **datatype.h**

8.45 ecvl::TypeInfo< ecvl::DataType::uint8 > Struct Template Reference

```
#include <datatype.h>
```

Public Types

- using **basetype** = uint8_t

8.45.1 Detailed Description

template<>
struct ecvl::TypeInfo< ecvl::DataType::uint8 >

Definition at line 2 of file datatype.h.

8.45.2 Member Typedef Documentation

8.45.2.1 basetype

```
using ecvl::TypeInfo< ecvl::DataType::uint8 >:: basetype = uint8_t
```

Definition at line 2 of file datatype.h.

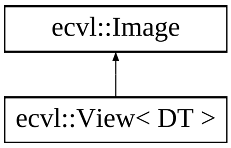
The documentation for this struct was generated from the following file:

- **datatype.h**

8.46 ecvl::View< DT > Class Template Reference

```
#include <image.h>
```

Inheritance diagram for ecvl::View< DT >:



Public Types

- using **basetype** = typename **TypeInfo**< DT >:: **basetype**

Public Member Functions

- **View** (**Image** &img)
- **View** (**Image** &img, const std::vector< int > &start, const std::vector< int > &size)
- **basetype** & **operator()** (const std::vector< int > &coords)
- **Iterator**< **basetype** > **Begin** ()
- **Iterator**< **basetype** > **End** ()

Additional Inherited Members

8.46.1 Detailed Description

template<DataType DT>
class ecvl::View< DT >

Definition at line 353 of file image.h.

8.46.2 Member Typedef Documentation

8.46.2.1 basetype

```
template<DataType DT>
using  ecvl::View< DT >::  basetype = typename  TypeInfo<DT>::  basetype
```

Definition at line 355 of file image.h.

8.46.3 Constructor & Destructor Documentation

8.46.3.1 View() [1/2]

```
template<DataType DT>
ecvl::View< DT >::  View (
              Image & img )  [inline]
```

Definition at line 357 of file image.h.

8.46.3.2 View() [2/2]

```
template<DataType DT>
ecvl::View< DT >::  View (
              Image & img,
             const std::vector< int > & start,
             const std::vector< int > & size )  [inline]
```

Definition at line 375 of file image.h.

8.46.4 Member Function Documentation

8.46.4.1 Begin()

```
template<DataType DT>
Iterator< basetype>  ecvl::View< DT >::Begin ( )  [inline]
```

Definition at line 415 of file image.h.

8.46.4.2 End()

```
template<DataType DT>
Iterator< basetype>  ecvl::View< DT >::End ( )  [inline]
```

Definition at line 416 of file image.h.

8.46.4.3 operator()()

```
template<DataType DT>
basetype&  ecvl::View< DT >::operator() (
             const std::vector< int > & coords )  [inline]
```

Definition at line 411 of file image.h.

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

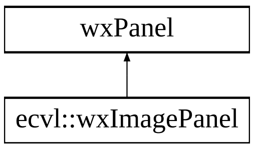
- **image.h**

8.47 ecvl::wxImagePanel Class Reference

wxImagePanel (p. 103) creates a wxPanel to contain an **Image** (p. 57).

`#include <gui.h>`

Inheritance diagram for ecvl::wxImagePanel:



Public Member Functions

- **wxImagePanel** (wxFrame *parent)
- void **SetImage** (const wxImage &img)

8.47.1 Detailed Description

wxImagePanel (p. 103) creates a wxPanel to contain an **Image** (p. 57).

Definition at line 13 of file gui.h.

8.47.2 Constructor & Destructor Documentation

8.47.2.1 wxImagePanel()

```
ecvl::wxImagePanel::wxImagePanel (
             wxFrame * parent )  [inline]
```

Definition at line 23 of file gui.h.

8.47.3 Member Function Documentation

8.47.3.1 SetImage()

```
void ecvl::wxImagePanel::SetImage (
             const wxImage & img )
```

Definition at line 10 of file gui.cpp.

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

- **gui.h**
- **gui.cpp**

Chapter 9

File Documentation

9.1 arithmetic.cpp File Reference

```
#include "ecvl/core/arithmetic.h"
```

Classes

- struct **ecvl::StructAdd**< **a, b** >
- struct **ecvl::StructSub**< **a, b** >
- struct **ecvl::StructMul**< **a, b** >
- struct **ecvl::StructScalarNeg**< **DT** >

Namespaces

- **ecvl**

Macros

- #define **STANDARD_INPLACE_OPERATION**(Function, TemplateImplementation)
- #define **STANDARD_NON_INPLACE_OPERATION**(Function)

Functions

- void **ecvl::Add** (Image &src1_dst, const Image &src2)
- void **ecvl::Sub** (Image &src1_dst, const Image &src2)
- void **ecvl::Mul** (Image &src1_dst, const Image &src2)
- Image & **ecvl::Neg** (Image &img)

  In-place negation of an ***Image*** *(p. 57).*
- void **ecvl::Mul** (const Image &src1, const Image &src2, Image &dst, DataType dst_type, bool saturate=true)

  Multiplies two Image(s) and stores the result in a third ***Image*** *(p. 57).*
- void **ecvl::Add** (const Image &src1, const Image &src2, Image &dst, DataType dst_type, bool saturate=true)

  Adds two Image(s) and stores the result in a third ***Image*** *(p. 57).*
- void **ecvl::Sub** (const Image &src1, const Image &src2, Image &dst, DataType dst_type, bool saturate=true)

  Subtracts two Image(s) and stores the result in a third ***Image*** *(p. 57).*

9.1.1 Macro Definition Documentation

9.1.1.1 STANDARD_INPLACE_OPERATION

```
#define STANDARD_INPLACE_OPERATION(
                Function,
                TemplateImplementation )
```

Value:

```
void Function(Image& src1_dst, const Image& src2)                    \
{                                                                    \
    static constexpr Table2D<TemplateImplementation> table;          \
    table(src1_dst.elemtype_, src2.elemtype_)(src1_dst, src2);       \
}
```

Definition at line 14 of file arithmetic.cpp.

9.1.1.2 STANDARD_NON_INPLACE_OPERATION

```
#define STANDARD_NON_INPLACE_OPERATION(
                Function )
```

Value:

```
void Function(const Image& src1, const Image& src2, Image& dst, DataType dst_type, bool saturate) \
{                                                                                                 \
    if (src1.dims_ != src2.dims_ || src1.channels_ != src2.channels_) {                           \
        throw std::runtime_error("Source images must have the same dimensions and channels.");    \
    }                                                                                             \
                                                                                                  \
    if (!dst.IsOwner()) {                                                                         \
        if (src1.dims_ != dst.dims_ || src1.channels_ != dst.channels_) {                         \
            throw std::runtime_error("Non-owning data destination image must have the             \
                 same dimensions and channels as the sources.");                               \
        }                                                                                         \
    }                                                                                             \
                                                                                                  \
    CopyImage(src1, dst, dst_type);                                                               \
    Function(dst, src2);                                                                          \
}
```

Definition at line 108 of file arithmetic.cpp.

9.2 arithmetic.h File Reference

```
#include "ecvl/core/datatype_matrix.h"
#include "ecvl/core/image.h"
#include "ecvl/core/standard_errors.h"
```

Classes

- struct **ecvl::StructDiv< a, b, ET >**
- struct **ecvl::StructScalarMul< DT, T >**
- struct **ecvl::StructScalarAdd< DT, T >**
- struct **ecvl::StructScalarSub< DT, T >**
- struct **ecvl::StructScalarSubInv< DT, T >**
- struct **ecvl::StructScalarDiv< DT, T >**
- struct **ecvl::StructScalarDivInv< DT, T, ET >**

Namespaces

- **ecvl**

Functions

- template<DataType ODT, typename IDT >
  TypeInfo< ODT >::basetype **ecvl::saturate_cast** (IDT v)

  Saturate a value (of any type) to the specified type.
- template<typename ODT , typename IDT >
  ODT **ecvl::saturate_cast** (const IDT &v)

  Saturate a value (of any type) to the specified type.
- void **ecvl::Add** (Image &src1_dst, const Image &src2)
- void **ecvl::Sub** (Image &src1_dst, const Image &src2)
- void **ecvl::Mul** (Image &src1_dst, const Image &src2)
- template<typename T >
  Image & **ecvl::Mul** (Image &img, T value, bool saturate=true)

  *In-place multiplication between an **Image** (p. 57) and a scalar value, without type promotion.*
- template<typename T >
  Image & **ecvl::Mul** (T value, Image &img, bool saturate=true)
- template<typename T >
  Image & **ecvl::Add** (Image &img, T value, bool saturate=true)

  *In-place addition between an **Image** (p. 57) and a scalar value, without type promotion.*
- template<typename T >
  Image & **ecvl::Add** (T value, Image &img, bool saturate=true)
- template<typename T >
  Image & **ecvl::Sub** (Image &img, T value, bool saturate=true)

  *In-place subtraction between an **Image** (p. 57) and a scalar value, without type promotion.*
- template<typename T >
  Image & **ecvl::Sub** (T value, Image &img, bool saturate=true)

  *In-place subtraction between a scalar value and an **Image** (p. 57), without type promotion.*
- template<typename T >
  Image & **ecvl::Div** (Image &img, T value, bool saturate=true)

  *In-place division between an **Image** (p. 57) and a scalar value, without type promotion.*
- template<typename T , typename ET = double>
  Image & **ecvl::Div** (T value, Image &img, bool saturate=true, ET epsilon=std::numeric_limits< double >↩
  ::min())

  *In-place divion between a scalar value and an **Image** (p. 57), without type promotion.*
- Image & **ecvl::Neg** (Image &img)

  *In-place negation of an **Image** (p. 57).*
- void **ecvl::Mul** (const Image &src1, const Image &src2, Image &dst, DataType dst_type, bool saturate=true)

  *Multiplies two Image(s) and stores the result in a third **Image** (p. 57).*

- void **ecvl::Sub** (const Image &src1, const Image &src2, Image &dst, DataType dst_type, bool saturate=true)

  *Subtracts two Image(s) and stores the result in a third **Image** (p. 57).*
- void **ecvl::Add** (const Image &src1, const Image &src2, Image &dst, DataType dst_type, bool saturate=true)

  *Adds two Image(s) and stores the result in a third **Image** (p. 57).*
- template<typename ET = double>
  void **ecvl::Div** (const Image &src1, const Image &src2, Image &dst, DataType dst_type, bool saturate=true, ET epsilon=std::numeric_limits< double >::min())

  *Divides two Image(s) and stores the result in a third **Image** (p. 57).*

9.3 core.cpp File Reference

```
#include "ecvl/core/image.h"
```

9.4 core.h File Reference

```
#include "core/arithmetic.h"
#include "core/datatype.h"
#include "core/filesystem.h"
#include "core/image.h"
#include "core/imgcodecs.h"
#include "core/imgproc.h"
#include "core/iterators.h"
#include "core/memorymanager.h"
#include "core/support_opencv.h"
```

9.5 datatype.cpp File Reference

```
#include "ecvl/core/datatype.h"
#include "ecvl/core/datatype_tuples.inc.h"
```

Namespaces

- **ecvl**

Macros

- #define **ECVL_TUPLE**(name, size, ...) size,

Functions

- uint8_t **ecvl::DataTypeSize** (DataType dt)

  Provides the size in bytes of a given DataType.

9.5.1 Macro Definition Documentation

9.5.1.1 ECVL_TUPLE

```
#define ECVL_TUPLE(
              name,
              size,
              ...  ) size,
```

9.6 datatype.h File Reference

```
#include <cstdint>
#include <limits>
#include <array>
#include "datatype_tuples.inc.h"
#include "datatype_existing_tuples.inc.h"
#include "datatype_existing_tuples_signed.inc.h"
```

Classes

- struct **ecvl::TypeInfo**< **DataType** >
- struct **ecvl::TypeInfo**< **ecvl::DataType::int8** >
- struct **ecvl::TypeInfo**< **ecvl::DataType::int16** >
- struct **ecvl::TypeInfo**< **ecvl::DataType::int32** >
- struct **ecvl::TypeInfo**< **ecvl::DataType::int64** >
- struct **ecvl::TypeInfo**< **ecvl::DataType::float32** >
- struct **ecvl::TypeInfo**< **ecvl::DataType::float64** >
- struct **ecvl::TypeInfo**< **ecvl::DataType::uint8** >
- struct **ecvl::TypeInfo**< **ecvl::DataType::uint16** >
- struct **ecvl::TypeInfo**< **ecvl::DataType::uint32** >
- struct **ecvl::TypeInfo**< **ecvl::DataType::uint64** >
- struct **ecvl::TypeInfo**< **ecvl::DataType::none** >

Namespaces

- **ecvl**

Macros

- #define **ECVL_TUPLE**(name, ...) name,
- #define **ECVL_TUPLE**(name, size, type, ...) template<> struct TypeInfo<ecvl::DataType::name> { using basetype = type; };
- #define **ECVL_TUPLE**(name, ...) + 1
- #define **ECVL_TUPLE**(name, ...) + 1

Enumerations

- enum **ecvl::DataType** {
  ecvl::DataType::ECVL_TUPLE, **ecvl::DataType::int8**, **ecvl::DataType::int16**, **ecvl::DataType::int32**, **ecvl::DataType::int64**, **ecvl::DataType::float32**, **ecvl::DataType::float64**, **ecvl::DataType::uint8**, **ecvl::DataType::uint16**, **ecvl::DataType::uint32**, **ecvl::DataType::uint64**, **ecvl::DataType::none** }

    DataType is an enum class which defines data types allowed for images.

Functions

- uint8_t **ecvl::DataTypeSize** (DataType dt)

    Provides the size in bytes of a given DataType.
- constexpr size_t **ecvl::DataTypeSize** ()

    Function to get the number of existing DataType at compile time.
- constexpr size_t **ecvl::DataTypeSignedSize** ()

    Function to get the number of existing signed DataType at compile time.
- constexpr std::array< DataType, DataTypeSize()> **ecvl::DataTypeArray** ()

    Function to get a std::array with all the DataType values at compile time.
- constexpr std::array< DataType, DataTypeSignedSize()> **ecvl::DataTypeSignedArray** ()

    Function to get a std::array with all the signed DataType values at compile time.

9.6.1 Macro Definition Documentation

9.6.1.1 ECVL_TUPLE [1/4]

```
#define ECVL_TUPLE(
              name,
              ...  ) name,
```

Definition at line 34 of file datatype.h.

9.6.1.2 ECVL_TUPLE [2/4]

```
#define ECVL_TUPLE(
              name,
              size,
              type,
              ...  ) template<> struct TypeInfo<ecvl::DataType::name> { using basetype =
type; };
```

Definition at line 34 of file datatype.h.

9.6.1.3 ECVL_TUPLE [3/4]

```
#define ECVL_TUPLE(
              name,
              ...  ) + 1
```

Definition at line 34 of file datatype.h.

9.6.1.4 ECVL_TUPLE [4/4]

```
#define ECVL_TUPLE(
              name,
              ...  ) + 1
```

Definition at line 34 of file datatype.h.

9.7 datatype_existing_tuples.inc.h File Reference

```
#include "datatype_existing_tuples_signed.inc.h"
#include "datatype_existing_tuples_unsigned.inc.h"
```

9.8 datatype_existing_tuples_signed.inc.h File Reference

9.9 datatype_existing_tuples_unsigned.inc.h File Reference

9.10 datatype_matrix.h File Reference

```
#include "datatype.h"
```

Classes

- struct **ecvl::Table1D< _StructFun, Args >**
- struct **ecvl::Table1D< _StructFun, Args >::integer< i >**
- struct **ecvl::SignedTable1D< _StructFun, Args >**
- struct **ecvl::SignedTable1D< _StructFun, Args >::integer< i >**
- struct **ecvl::Table2D< _StructFun, Args >**
- struct **ecvl::Table2D< _StructFun, Args >::integer< i >**

Namespaces

- **ecvl**

9.11 datatype_tuples.inc.h File Reference

```
#include "datatype_existing_tuples.inc.h"
```

9.12 filesystem.cc File Reference

```
#include "ecvl/core/filesystem.h"
#include <algorithm>
#include <fstream>
#include <string>
#include <sys/stat.h>
#include <sys/types.h>
```

Namespaces

- **filesystem**

Functions

- bool **filesystem::exists** (const path &p)
- bool **filesystem::exists** (const path &p, error_code &ec)
- bool **filesystem::create_directories** (const path &p)
- bool **filesystem::create_directories** (const path &p, error_code &ec)
- void **filesystem::copy** (const path &from, const path &to)
- void **filesystem::copy** (const path &from, const path &to, error_code &ec)

9.13 filesystem.h File Reference

```
#include <string>
#include <system_error>
```

Classes

- class **filesystem::path**

Namespaces

- **filesystem**

Functions

- path **filesystem::operator/** (const path &lhs, const path &rhs)
- bool **filesystem::exists** (const path &p)
- bool **filesystem::exists** (const path &p, std::error_code &ec)
- bool **filesystem::create_directories** (const path &p)
- bool **filesystem::create_directories** (const path &p, std::error_code &ec)
- void **filesystem::copy** (const path &from, const path &to)
- void **filesystem::copy** (const path &from, const path &to, std::error_code &ec)

9.14 gui.cpp File Reference

```
#include "ecvl/gui.h"
```

Namespaces

- **ecvl**

Functions

- wxImage **ecvl::wx_from_mat** (Image &img)
- void **ecvl::ImShow** (const Image &img)

  *Displays an **Image** (p. 57).*

9.15 gui.h File Reference

```
#include <wx/wx.h>
#include "ecvl/core/image.h"
```

Classes

- class **ecvl::wxImagePanel**

  ***wxImagePanel** (p. 103) creates a wxPanel to contain an **Image** (p. 57).*
- class **ecvl::ShowApp**

  ***ShowApp** (p. 77) is a custom wxApp which allows you to visualize an ECVL **Image** (p. 57).*

Namespaces

- **ecvl**

Functions

- void **ecvl::ImShow** (const Image &img)

  *Displays an **Image** (p. 57).*

9.16 home.h File Reference

9.17 image.cpp File Reference

```
#include "ecvl/core/image.h"
#include "ecvl/core/datatype_matrix.h"
#include "ecvl/core/standard_errors.h"
```

Namespaces

- **ecvl**

Functions

- void **ecvl::RearrangeChannels** (const Image &src, Image &dst, const std::string &channels)

    Changes the order of the ***Image*** *(p. 57) dimensions.*
- void **ecvl::CopyImage** (const Image &src, Image &dst, DataType new_type=DataType::none)

    Copies the source ***Image*** *(p. 57) into the destination* ***Image*** *(p. 57).*

9.18 image.h File Reference

```
#include <algorithm>
#include <numeric>
#include <stdexcept>
#include <vector>
#include <opencv2/core.hpp>
#include "datatype.h"
#include "memorymanager.h"
#include "iterators.h"
#include "iterators_impl.inc.h"
```

Classes

- class **ecvl::MetaData**
- class **ecvl::Image**

    Image *(p. 57) class.*
- class **ecvl::View**< **DT** >
- class **ecvl::ConstView**< **DT** >
- class **ecvl::ContiguousView**< **DT** >
- class **ecvl::ConstContiguousView**< **DT** >
- class **ecvl::ContiguousViewXYC**< **DT** >
- struct **ecvl::StructCopyImage**< **SDT, DDT** >

Namespaces

- **ecvl**

Enumerations

- enum **ecvl::ColorType** {
  ecvl::ColorType::none, **ecvl::ColorType::GRAY**, **ecvl::ColorType::RGB**, **ecvl::ColorType::BGR**, **ecvl::ColorType::HSV**, **ecvl::ColorType::YCbCr** }

  Enum class representing the ECVL supported color spaces.

Functions

- void **ecvl::RearrangeChannels** (const Image &src, Image &dst, const std::string &channels)

  *Changes the order of the **Image** (p. 57) dimensions.*
- void **ecvl::CopyImage** (const Image &src, Image &dst, DataType new_type=DataType::none)

  *Copies the source **Image** (p. 57) into the destination **Image** (p. 57).*

9.19 imgcodecs.cpp File Reference

```
#include "ecvl/core/imgcodecs.h"
#include <opencv2/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include "ecvl/core/support_opencv.h"
```

Namespaces

- **ecvl**

Functions

- bool **ecvl::ImRead** (const std::string &filename, Image &dst)

  Loads an image from a file.
- bool **ecvl::ImRead** (const **filesystem::path** &filename, Image &dst)
- bool **ecvl::ImWrite** (const std::string &filename, const Image &src)

  Saves an image into a specified file.
- bool **ecvl::ImWrite** (const **filesystem::path** &filename, const Image &src)

9.20 imgcodecs.h File Reference

```
#include <string>
#include "image.h"
#include "filesystem.h"
```

Namespaces

- **ecvl**

Functions

- bool **ecvl::ImRead** (const std::string &filename, Image &dst)

  Loads an image from a file.
- bool **ecvl::ImRead** (const **filesystem::path** &filename, Image &dst)
- bool **ecvl::ImWrite** (const std::string &filename, const Image &src)

  Saves an image into a specified file.
- bool **ecvl::ImWrite** (const **filesystem::path** &filename, const Image &src)

9.21 imgproc.cpp File Reference

```
#include "ecvl/core/imgproc.h"
#include <stdexcept>
#include <opencv2/imgproc.hpp>
#include "ecvl/core/datatype_matrix.h"
#include "ecvl/core/standard_errors.h"
```

Namespaces

- **ecvl**

Functions

- void **ecvl::ResizeDim** (const **ecvl::Image** &src, **ecvl::Image** &dst, const std::vector< int > &newdims, InterpolationType interp=InterpolationType::linear)

  *Resizes an **Image** (p. 57) to a new dimension.*
- void **ecvl::ResizeScale** (const **ecvl::Image** &src, **ecvl::Image** &dst, const std::vector< double > &scales, InterpolationType interp=InterpolationType::linear)

  *Resizes an **Image** (p. 57) by scaling the dimentions to a given scale factor.*
- void **ecvl::Flip2D** (const **ecvl::Image** &src, **ecvl::Image** &dst)

  *Flips an **Image** (p. 57).*
- void **ecvl::Mirror2D** (const **ecvl::Image** &src, **ecvl::Image** &dst)

  *Mirrors an **Image** (p. 57).*
- void **ecvl::Rotate2D** (const **ecvl::Image** &src, **ecvl::Image** &dst, double angle, const std::vector< double > &center={}, double scale=1.0, InterpolationType interp=InterpolationType::linear)

  *Rotates an **Image** (p. 57).*
- void **ecvl::RotateFullImage2D** (const **ecvl::Image** &src, **ecvl::Image** &dst, double angle, double scale=1.0, InterpolationType interp=InterpolationType::linear)

  *Rotates an **Image** (p. 57) resizing the output accordingly.*
- void **ecvl::ChangeColorSpace** (const Image &src, Image &dst, ColorType new_type)

  *Copies the source **Image** (p. 57) into destination **Image** (p. 57) changing the color space.*
- void **ecvl::Threshold** (const Image &src, Image &dst, double thresh, double maxval, ThresholdingType thresh_type=ThresholdingType::BINARY)

  *Applies a fixed threshold to an input **Image** (p. 57).*
- double **ecvl::OtsuThreshold** (const Image &src)

  Calculates the Otsu thresholding value.

9.22 imgproc.h File Reference

```
#include "image.h"
#include "support_opencv.h"
```

Namespaces

- **ecvl**

Enumerations

- enum **ecvl::ThresholdingType** { **ecvl::ThresholdingType::BINARY**, **ecvl::ThresholdingType::BINAR↩ Y_INV** }

  Enum class representing the ECVL threhsolding types.
- enum **ecvl::InterpolationType** {
  ecvl::InterpolationType::nearest, **ecvl::InterpolationType::linear**, **ecvl::InterpolationType::area**,
  ecvl::InterpolationType::cubic,
  ecvl::InterpolationType::lanczos4 }

  Enum class representing the ECVL interpolation types.

Functions

- void **ecvl::ResizeDim** (const **ecvl::Image** &src, **ecvl::Image** &dst, const std::vector< int > &newdims, InterpolationType interp=InterpolationType::linear)

  Resizes an ***Image*** *(p. 57) to a new dimension.*
- void **ecvl::ResizeScale** (const **ecvl::Image** &src, **ecvl::Image** &dst, const std::vector< double > &scales, InterpolationType interp=InterpolationType::linear)

  Resizes an ***Image*** *(p. 57) by scaling the dimentions to a given scale factor.*
- void **ecvl::Flip2D** (const **ecvl::Image** &src, **ecvl::Image** &dst)

  Flips an ***Image*** *(p. 57).*
- void **ecvl::Mirror2D** (const **ecvl::Image** &src, **ecvl::Image** &dst)

  Mirrors an ***Image*** *(p. 57).*
- void **ecvl::Rotate2D** (const **ecvl::Image** &src, **ecvl::Image** &dst, double angle, const std::vector< double > &center={}, double scale=1.0, InterpolationType interp=InterpolationType::linear)

  Rotates an ***Image*** *(p. 57).*
- void **ecvl::RotateFullImage2D** (const **ecvl::Image** &src, **ecvl::Image** &dst, double angle, double scale=1.0, InterpolationType interp=InterpolationType::linear)

  Rotates an ***Image*** *(p. 57) resizing the output accordingly.*
- void **ecvl::ChangeColorSpace** (const Image &src, Image &dst, ColorType new_type)

  Copies the source ***Image*** *(p. 57) into destination* ***Image*** *(p. 57) changing the color space.*
- void **ecvl::Threshold** (const Image &src, Image &dst, double thresh, double maxval, ThresholdingType thresh_type=ThresholdingType::BINARY)

  Applies a fixed threshold to an input ***Image*** *(p. 57).*
- double **ecvl::OtsuThreshold** (const Image &src)

  Calculates the Otsu thresholding value.

9.23 iterators.h File Reference

```
#include <vector>
#include <cstdint>
```

Classes

- struct **ecvl::Iterator**< **T** >
- struct **ecvl::ConstIterator**< **T** >
- struct **ecvl::ContiguousIterator**< **T** >
- struct **ecvl::ConstContiguousIterator**< **T** >

Namespaces

- **ecvl**

9.24 iterators_impl.inc.h File Reference

9.25 memorymanager.cpp File Reference

```
#include "ecvl/core/memorymanager.h"
```

9.26 memorymanager.h File Reference

```
#include <cstdint>
#include <cstring>
#include <stdexcept>
```

Classes

- class **MemoryManager**
- class **DefaultMemoryManager**
- class **ShallowMemoryManager**

9.27 standard_errors.h File Reference

Macros

- #define **ECVL_ERROR_MSG** "[Error]: "
- #define **ECVL_WARNING_MSG** "[Warning]: "
- #define **ECVL_ERROR_NOT_IMPLEMENTED** throw std::runtime_error(**ECVL_ERROR_MSG** "Not implemented");
- #define **ECVL_ERROR_NOT_REACHABLE_CODE** throw std::runtime_error(**ECVL_ERROR_MSG** "How did you get here?");
- #define **ECVL_ERROR_WRONG_PARAMS**(...) throw std::runtime_error(**ECVL_ERROR_MSG** "Wrong parameters - " __VA_ARGS__);
- #define **ECVL_ERROR_NOT_ALLOWED_ON_NON_OWING_IMAGE**(...) throw std::runtime_error(**ECV↩ L_ERROR_MSG** "Operation not allowed on non-owning Image" __VA_ARGS__);
- #define **ECVL_ERROR_UNSUPPORTED_OPENCV_DEPTH** throw std::runtime_error(**ECVL_ERROR_↩ MSG** "Unsupported OpenCV depth");
- #define **ECVL_ERROR_UNSUPPORTED_OPENCV_DIMS** throw std::runtime_error(**ECVL_ERROR_MSG** "Unsupported OpenCV dimensions");
- #define **ECVL_ERROR_EMPTY_IMAGE** throw std::runtime_error(**ECVL_ERROR_MSG** "Empty image provided");
- #define **ECVL_ERROR_NOT_ALLOWED_ON_UNSIGNED_IMG** throw std::runtime_error(**ECVL_ERRO↩ R_MSG** "Operation not allowed on unsigned Image");

9.27.1 Macro Definition Documentation

9.27.1.1 ECVL_ERROR_EMPTY_IMAGE

```
#define ECVL_ERROR_EMPTY_IMAGE throw std::runtime_error( ECVL_ERROR_MSG "Empty image provided");
```

Definition at line 13 of file standard_errors.h.

9.27.1.2 ECVL_ERROR_MSG

```
#define ECVL_ERROR_MSG "[Error]:  "
```

Definition at line 4 of file standard_errors.h.

9.27.1.3 ECVL_ERROR_NOT_ALLOWED_ON_NON_OWING_IMAGE

```
#define ECVL_ERROR_NOT_ALLOWED_ON_NON_OWING_IMAGE(
              ...  ) throw std::runtime_error( ECVL_ERROR_MSG "Operation not allowed on non-owning
Image" __VA_ARGS__);
```

Definition at line 10 of file standard_errors.h.

9.27.1.4 ECVL_ERROR_NOT_ALLOWED_ON_UNSIGNED_IMG

```
#define ECVL_ERROR_NOT_ALLOWED_ON_UNSIGNED_IMG throw std::runtime_error( ECVL_ERROR_MSG "Operation
not allowed on unsigned Image");
```

Definition at line 14 of file standard_errors.h.

9.27.1.5 ECVL_ERROR_NOT_IMPLEMENTED

```
#define ECVL_ERROR_NOT_IMPLEMENTED throw std::runtime_error( ECVL_ERROR_MSG "Not implemented");
```

Definition at line 7 of file standard_errors.h.

9.27.1.6 ECVL_ERROR_NOT_REACHABLE_CODE

```
#define ECVL_ERROR_NOT_REACHABLE_CODE throw std::runtime_error( ECVL_ERROR_MSG "How did you
get here?");
```

Definition at line 8 of file standard_errors.h.

9.27.1.7 ECVL_ERROR_UNSUPPORTED_OPENCV_DEPTH

```
#define ECVL_ERROR_UNSUPPORTED_OPENCV_DEPTH throw std::runtime_error( ECVL_ERROR_MSG "Unsupported
OpenCV depth");
```

Definition at line 11 of file standard_errors.h.

9.27.1.8 ECVL_ERROR_UNSUPPORTED_OPENCV_DIMS

```
#define ECVL_ERROR_UNSUPPORTED_OPENCV_DIMS throw std::runtime_error( ECVL_ERROR_MSG "Unsupported
OpenCV dimensions");
```

Definition at line 12 of file standard_errors.h.

9.27.1.9 ECVL_ERROR_WRONG_PARAMS

```
#define ECVL_ERROR_WRONG_PARAMS(
              ...  ) throw std::runtime_error( ECVL_ERROR_MSG "Wrong parameters - " __VA_ARGS←
__);
```

Definition at line 9 of file standard_errors.h.

9.27.1.10 ECVL_WARNING_MSG

```
#define ECVL_WARNING_MSG "[Warning]:  "
```

Definition at line 5 of file standard_errors.h.

9.28 support_opencv.cpp File Reference

```
#include "ecvl/core/support_opencv.h"
#include "ecvl/core/standard_errors.h"
```

Namespaces

- **ecvl**

Functions

- **ecvl::Image ecvl::MatToImage** (const cv::Mat &m)
  *Convert a cv::Mat into an **ecvl::Image** (p. 57).*
- cv::Mat **ecvl::ImageToMat** (const Image &img)
  *Convert an ECVL **Image** (p. 57) into OpenCV Mat.*

9.29 support_opencv.h File Reference

```
#include "image.h"
```

Namespaces

- **ecvl**

Functions

- **ecvl::Image ecvl::MatToImage** (const cv::Mat &m)
  *Convert a cv::Mat into an **ecvl::Image** (p. 57).*
- cv::Mat **ecvl::ImageToMat** (const Image &img)
  *Convert an ECVL **Image** (p. 57) into OpenCV Mat.*

9.30 test_core.cpp File Reference

```
#include <gtest/gtest.h>
#include "ecvl/core.h"
```

Functions

- **TEST** (Core, CreateEmptyImage)
- **TEST** (Core, CreateImageWithFiveDims)

9.30.1 Function Documentation

9.30.1.1 TEST() [1/2]

```
TEST (
            Core ,
            CreateEmptyImage  )
```

Definition at line 7 of file test_core.cpp.

9.30.1.2 TEST() [2/2]

```
TEST (
            Core ,
            CreateImageWithFiveDims  )
```

Definition at line 14 of file test_core.cpp.

Index