AlgoCommon

Generated by Doxygen 1.8.11

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 cva Namespace Reference

Namespaces

- ac

4.2 cva::ac Namespace Reference

Namespaces

- ocv

Classes

- class ImageViewT

  A non-owning reference to an image.
- class PendingRequest

  A handle to an asynchronous operation that is either in progress or has completed.
- struct Point2

  A point in two-dimensional space whose coordinates are of type `T`*.*
- class Rectangle

  A rectangle of points with coordinates of type `T`*.*
- class Span

  A non-owning reference to a (possibly-empty) contiguous sequence of objects of type `T`*.*
- class Version

  Version number of a library which consists of major, minor and patch numbers.
- class View2d

  A non-owning reference to a two-dimensional block of pixels, each of which is a contiguous sequence of one or more channels of type T.

Typedefs

- using ConstImageView = ImageViewT< const void >
- using ImageView = ImageViewT< void >

Enumerations

- enum WaitResult { WaitResult::COMPLETED, WaitResult::TIMED_OUT }

  Result of a wait for an asynchronous operation's completion.
- enum ImageFormat { ImageFormat::LABEL_8, ImageFormat::INTENSITY_8, ImageFormat::RGB_8, ImageFormat::BGR_8 }

  An identifier of a specific number of channels and channel type of a pixel, as well as an intended interpretation of such a pixel.
- enum Precision {
  Precision::FP16, Precision::FP32, Precision::I8, Precision::I16,
  Precision::I32, Precision::Q78, Precision::U8, Precision::U16 }

  A number's precision/range.

Functions

- Version version ()

  Returns version number of the library.
- void outputToStream (std::ostream &)

  Does nothing.
- template<typename Arg0 , typename... Args>
  void outputToStream (std::ostream &os, const Arg0 &arg0, const Args &...args)

  Streams `arg0` and every element of `args` into `os`, in turn.
- template<typename E , typename... Args>
  void failRequirement (const char *class_name, const char *function_name, const Args &...args)

  Throws an exception of type `E` constructed with a string created from `class_name`, `function_name`, and `args`.
- template<typename E , typename... Args>
  void require (const char *class_name, const char *function_name, bool condition, const Args &...args)

  Calls failRequirement(`class_name`, `function_name`, `args`...) unless `condition` is true.
- std::size_t imageFormatNumChannels (ImageFormat format)

  Returns the number of channels identified by `format`.
- void imageFormatRequireMatchingNumChannels (const char *class_name, const char *function_name, ImageFormat format, const char *format_str, std::size_t nc, const char *nc_str)

  Throws an exception if `nc` is not imageFormatNumChannels(`format`).
- template<typename T >
  void imageFormatRequireMatchingType (const char *class_name, const char *function_name, ImageFormat format, const char *format_str, const char *type_str)

  Throws an exception if `T` does not match `format`.
- std::ostream & operator<< (std::ostream &os, Precision p)

  Streams a string representation of `p` into `os`.
- template<typename T >
  Span< T > toSpan (Span< T > span)

  Returns a copy of `span`.
- template<typename T , std::size_t N>
  Span< T > toSpan (T(&array)[N])

  Returns a span that references the sequence of all elements of `array`.
- template<typename T , typename Allocator >
  Span< T > toSpan (std::vector< T, Allocator > &vector)

  Returns a span that references the sequence of all elements of `vector`.
- template<typename T , typename Allocator >
  Span< const T > toSpan (const std::vector< T, Allocator > &vector)

4.2.1 Typedef Documentation

4.2.1.1 using cva::ac::ConstImageView = typedef ImageViewT<const void>

4.2.1.2 using cva::ac::ImageView = typedef ImageViewT<void>

4.2.2 Enumeration Type Documentation

4.2.2.1 enum cva::ac::ImageFormat [strong]

An identifier of a specific number of channels and channel type of a pixel, as well as an intended interpretation of such a pixel.

A type `T` is considered to match a format if `T` is the channel type identified by the format, or a cv-qualified version of it.

Enumerator

LABEL_8 One `std::uint8_t` channel representing an abstract label.

INTENSITY_8 One `std::uint8_t` channel representing an intensity of light.

RGB_8 Three `std::uint8_t` channels representing intensities of red, green and blue color components.

BGR_8 Three `std::uint8_t` channels representing intensities of blue, green and red color components.

4.2.2.2 enum cva::ac::Precision [strong]

A number's precision/range.

Enumerator

FP16 16-bit floating point number

FP32 32-bit floating point number

I8 8-bit signed integer

I16 16-bit signed integer

I32 32-bit signed integer

Q78 signed fixed-point number with 7 integer and 8 fractional bits

U8 8-bit unsigned integer

U16 16-bit unsigned integer

4.2.2.3 enum cva::ac::WaitResult [strong]

Result of a wait for an asynchronous operation's completion.

Enumerator

COMPLETED The operation has completed.

TIMED_OUT The wait has timed out.

4.2.3 Function Documentation

4.2.3.1 template<typename E , typename... Args> void cva::ac::failRequirement (const char * *class_name,* const char * *function_name,* const Args &... *args*) [inline]

Throws an exception of type `E` constructed with a string created from `class_name`, `function_name`, and `args`.

Precondition

`E` is constructible from an `std::string`.
`function_name` is not null.

The exception is constructed with an std::string that contains `class_name`, `function_name` and a message that consists of the string representations of `args` joined together, without delimiters. The string representations are obtained by streaming each element of `args` into an `std::ostream`.

`class_name` may be null, in which case it is omitted from the string.

4.2.3.2 std::size_t cva::ac::imageFormatNumChannels (ImageFormat *format*)

Returns the number of channels identified by `format`.

4.2.3.3 void cva::ac::imageFormatRequireMatchingNumChannels (const char * *class_name,* const char * *function_name,* ImageFormat *format,* const char * *format_str,* std::size_t *nc,* const char * *nc_str*)

Throws an exception if `nc` is not imageFormatNumChannels(`format`).

`format_str` and `nc_str` must be expressions, evaluating which yielded `format` and `nc`, respectively. They are used as part of the exception message.

`class_name` and `function_name` are used in the same way as in failRequirement().

4.2.3.4 template<typename T > void cva::ac::imageFormatRequireMatchingType (const char * *class_name,* const char * *function_name,* ImageFormat *format,* const char * *format_str,* const char * *type_str*) [inline]

Throws an exception if `T` does not match `format`.

`format_str` and `type_str` must be expressions/type specifications, evaluating which yielded `format` and `T`, respectively. They are used as part of the exception message.

`class_name` and `function_name` are used in the same way as in failRequirement().

4.2.3.5 std::ostream& cva::ac::operator<< (std::ostream & *os,* Precision *p*)

Streams a string representation of `p` into `os`.

4.2.3.6 void cva::ac::outputToStream (std::ostream &) [inline]

Does nothing.

This function only exists to help with generic programming.

4.2.3.7 template<typename Arg0 , typename... Args> void cva::ac::outputToStream (std::ostream & *os,* const Arg0 & *arg0,* const Args &... *args*) [inline]

Streams `arg0` and every element of `args` into `os`, in turn.

4.2.3.8 template<typename E , typename... Args> void cva::ac::require (const char * *class_name,* const char * *function_name,* bool *condition,* const Args &... *args*) [inline]

Calls failRequirement(`class_name`, `function_name`, `args`...) unless `condition` is true.

4.2.3.9 template<typename T > Span<T> cva::ac::toSpan (Span< T > *span*)

Returns a copy of `span`.

This function exists to simplify generic programming.

4.2.3.10 template<typename T , std::size_t N> Span<T> cva::ac::toSpan (T(&) *array[N]*)

Returns a span that references the sequence of all elements of `array`.

4.2.3.11 template<typename T , typename Allocator > Span<T> cva::ac::toSpan (std::vector< T, Allocator > & *vector*)

Returns a span that references the sequence of all elements of `vector`.

The span is invalidated whenever the return value of `vector.data` is invalidated.

4.2.3.12 template<typename T , typename Allocator > Span<const T> cva::ac::toSpan (const std::vector< T, Allocator > & *vector*)

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

4.2.3.13 Version cva::ac::version ( )

Returns version number of the library.

4.3 cva::ac::ocv Namespace Reference

Functions

- CVA_AC_SHARED_LIBRARY_IMPORT ImageView toImageView (ImageFormat format, const cv::Mat &mat)

4.3.1 Function Documentation

4.3.1.1 CVA_AC_SHARED_LIBRARY_IMPORT ImageView cva::ac::ocv::toImageView (ImageFormat *format,* const cv::Mat & *mat*)

Returns a view to the image referenced by `mat`, assuming its pixel format is `format`.

Precondition

`mat.channels()` is the number of channels identified by `format`.
`mat.depth()` and `format` identify the same channel type.

Chapter 5

Class Documentation

5.1 cva::ac::ImageViewT< T > Class Template Reference

A non-owning reference to an image.

`#include <image_view.hpp>`

Public Member Functions

- template<typename U , typename = typename std::enable_if< std::is_convertible<U*, T*>::value>::type>
  ImageViewT (ImageFormat format, const View2d< U > &typed)

  Constructs a view to the image referenced by `typed`, assuming its pixel format is `format`.
- template<typename U , typename = typename std::enable_if< std::is_convertible<U*, T*>::value>::type>
  ImageViewT (ImageViewT< U > that)

  Constructs a view to the image referenced by `that`.
- ImageFormat format () const

  Returns the pixel format of the referenced image.
- std::size_t width () const

  Returns the width of the referenced image.
- std::size_t height () const

  Returns the height of the referenced image.
- template<typename U >
  View2d< U > typed () const

  Returns a View2d that references the same image as `this`.

Friends

- template<typename U >
  class ImageViewT

5.1.1 Detailed Description

template<typename T>
class cva::ac::ImageViewT< T >

A non-owning reference to an image.

Precondition

`T` is `void` or cv-qualified `void`.

"Image" is defined as a block of pixels of a particular format, laid out in a way compatible with a View2d of an appropriate type.

The cv-qualifiers of `T` are used to restrict access to the image. For instance, if `T` is `const void`, then modifying the pixels will not be allowed.

Instead of using this template directly, you may want to use the helper aliases ImageView and ConstImageView.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 template<typename T> template<typename U , typename = typename std::enable_if< std::is_convertible<U*, T*>::value>::type> cva::ac::ImageViewT< T >::ImageViewT (ImageFormat *format,* const View2d< U > & *typed*) [inline]

Constructs a view to the image referenced by `typed`, assuming its pixel format is `format`.

Precondition

`T` has the same cv-qualifiers as `U`, or more.
`U` matches `format`.
`typed.nc()` is the number of channels identified by `format`.

5.1.2.2 template<typename T> template<typename U , typename = typename std::enable_if< std::is_convertible<U*, T*>::value>::type> cva::ac::ImageViewT< T >::ImageViewT (ImageViewT< U > *that*) [inline]

Constructs a view to the image referenced by `that`.

Precondition

`T` is either `U` or `U` with additional cv-qualifiers.

5.1.3 Member Function Documentation

5.1.3.1 template<typename T> ImageFormat cva::ac::ImageViewT< T >::format ( ) const [inline]

Returns the pixel format of the referenced image.

5.1.3.2 template<typename T> std::size_t cva::ac::ImageViewT< T >::height ( ) const [inline]

Returns the height of the referenced image.

5.1.3.3 template<typename T> template<typename U > View2d<U> cva::ac::ImageViewT< T >::typed ( ) const [inline]

Returns a View2d that references the same image as `this`.

Precondition

`U` matches format().

5.1.3.4 template<typename T> std::size_t cva::ac::ImageViewT< T >::width ( ) const [inline]

Returns the width of the referenced image.

5.1.4 Friends And Related Function Documentation

5.1.4.1 template<typename T> template<typename U > friend class ImageViewT [friend]

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

- inc/cva/ac/image_view.hpp

5.2 cva::ac::PendingRequest Class Reference

A handle to an asynchronous operation that is either in progress or has completed.

`#include <async.hpp>`

Public Member Functions

- virtual ~PendingRequest ()

  Releases resources associated with the asynchronous operation.
- virtual void wait ()=0

  Waits for the asynchronous operation to finish and marks it as completed.
- virtual WaitResult timedWait (std::uint64_t timeout_ms)=0

  Waits for the asynchronous operation to finish for up to `timeout_ms` milliseconds.
- virtual bool successful () const =0

  Returns whether the asynchronous operation has completed successfully.

5.2.1 Detailed Description

A handle to an asynchronous operation that is either in progress or has completed.

When a PendingRequest is created, the operation is considered to be in progress. While it may finish asynchronously, it will not be considered completed until one of the wait methods is called.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 virtual cva::ac::PendingRequest::~PendingRequest ( ) [inline], [virtual]

Releases resources associated with the asynchronous operation.

If the operation is still in progress, it is canceled. In this case the operation's outputs will be left with unspecified contents.

5.2.3 Member Function Documentation

5.2.3.1 virtual bool cva::ac::PendingRequest::successful ( ) const [pure virtual]

Returns whether the asynchronous operation has completed successfully.

Precondition

The operation has completed.

If this returns false, than the operation's outputs have unspecified contents.

5.2.3.2 virtual WaitResult cva::ac::PendingRequest::timedWait (std::uint64_t *timeout_ms*) [pure virtual]

Waits for the asynchronous operation to finish for up to `timeout_ms` milliseconds.

If it finishes within that time, marks it as completed and returns WaitResult::COMPLETED. Otherwise, returns WaitResult::TIMED_OUT.

If it has already completed, does nothing and returns WaitResult::COMPLETED.

5.2.3.3 virtual void cva::ac::PendingRequest::wait ( ) [pure virtual]

Waits for the asynchronous operation to finish and marks it as completed.

It it has already completed, does nothing and returns immediately.

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

- async.hpp

5.3 cva::ac::Point2< T > Struct Template Reference

A point in two-dimensional space whose coordinates are of type `T`.

`#include <geometry.hpp>`

Public Member Functions

- constexpr Point2 ()

  Constructs a point whose coordinates are both `T()`.
- constexpr Point2 (T x, T y)

  Constructs a point with given coordinates `x` and `y`.

Public Attributes

- T x
- T y

Friends

- bool operator== (const Point2 &left, const Point2 &right)

  Returns whether both coordinates of `left` compare equal to those of `right`.
- bool operator!= (const Point2 &left, const Point2 &right)

  Returns the opposite of operator==().

5.3.1 Detailed Description

template<typename T>
struct cva::ac::Point2< T >

A point in two-dimensional space whose coordinates are of type `T`.

Precondition

T is an arithmetic type.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 template<typename T> constexpr cva::ac::Point2< T >::Point2 ( ) [inline]

Constructs a point whose coordinates are both `T()`.

5.3.2.2 template<typename T> constexpr cva::ac::Point2< T >::Point2 (T *x*, T *y*) [inline]

Constructs a point with given coordinates `x` and `y`.

5.3.3 Friends And Related Function Documentation

5.3.3.1 template<typename T> bool operator!= (const Point2< T > & *left*, const Point2< T > & *right*) [friend]

Returns the opposite of operator==().

5.3.3.2 template<typename T> bool operator== (const Point2< T > & *left*, const Point2< T > & *right*) [friend]

Returns whether both coordinates of `left` compare equal to those of `right`.

5.3.4 Member Data Documentation

5.3.4.1 template<typename T> T cva::ac::Point2< T >::x

The x coordinate.

5.3.4.2 template<typename T> T cva::ac::Point2< T >::y

The y coordinate.

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

- geometry.hpp

5.4 cva::ac::Rectangle< T > Class Template Reference

A rectangle of points with coordinates of type `T`.

`#include <geometry.hpp>`

Public Member Functions

- constexpr Rectangle ()=default

  Constructs a rectangle whose start and end points are both (T(), T()).
- Rectangle (const Point2< T > &start, const Point2< T > &end)

  Constructs a rectangle with the given start and end points.
- Rectangle (T start_x, T start_y, T end_x, T end_y)

  Equivalent to `Rectangle(Point2<T>(start_x, start_y), Point2<T>(end_x, end_y))`.
- T area () const

  Returns area of the rectangle.
- constexpr Point2< T > start () const
- constexpr Point2< T > end () const
- constexpr T startX () const
- constexpr T startY () const
- constexpr T endX () const
- constexpr T endY () const

Friends

- Rectangle< T > operator& (const Rectangle< T > &left, const Rectangle< T > &right)

  Returns intersection rectangle.

5.4.1 Detailed Description

template<typename T>
class cva::ac::Rectangle< T >

A rectangle of points with coordinates of type `T`.

A rectangle's start point is its corner with the smallest coordinates, while its end point is its corner with the largest coordinates. A rectangle is assumed to include all points with coordinates between its start point's (inclusive) and its end point's (exclusive).

5.4.2 Constructor & Destructor Documentation

5.4.2.1 template<typename T> constexpr cva::ac::Rectangle< T >::Rectangle ( ) [default]

Constructs a rectangle whose start and end points are both (T(), T()).

5.4.2.2 template<typename T> cva::ac::Rectangle< T >::Rectangle (const Point2< T > & *start*, const Point2< T > & *end*) [inline]

Constructs a rectangle with the given start and end points.

Precondition

start.x <= end.x
start.y <= end.y

5.4.2.3 template<typename T> cva::ac::Rectangle< T >::Rectangle (T *start_x*, T *start_y*, T *end_x*, T *end_y*) [inline]

Equivalent to `Rectangle(Point2<T>(start_x, start_y), Point2<T>(end_x, end_y))`.

5.4.3 Member Function Documentation

5.4.3.1 template<typename T> T cva::ac::Rectangle< T >::area ( ) const [inline]

Returns area of the rectangle.

5.4.3.2 template<typename T> constexpr Point2<T> cva::ac::Rectangle< T >::end ( ) const [inline]

Returns the end point as a Point2.

5.4.3.3 template<typename T> constexpr T cva::ac::Rectangle< T >::endX ( ) const [inline]

Returns the x coordinate of the end point.

5.4.3.4 template<typename T> constexpr T cva::ac::Rectangle< T >::endY ( ) const [inline]

Returns the y coordinate of the end point.

5.4.3.5 template<typename T> constexpr Point2<T> cva::ac::Rectangle< T >::start ( ) const [inline]

Returns the start point as a Point2.

5.4.3.6 template<typename T> constexpr T cva::ac::Rectangle< T >::startX ( ) const [inline]

Returns the x coordinate of the start point.

5.4.3.7 template<typename T> constexpr T cva::ac::Rectangle< T >::startY ( ) const [inline]

Returns the y coordinate of the start point.

5.4.4 Friends And Related Function Documentation

5.4.4.1 template<typename T> Rectangle<T> operator& (const Rectangle< T > & *left*, const Rectangle< T > & *right*) [friend]

Returns intersection rectangle.

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

- geometry.hpp

5.5 cva::ac::Span< T > Class Template Reference

A non-owning reference to a (possibly-empty) contiguous sequence of objects of type `T`.

`#include <span.hpp>`

Public Member Functions

- constexpr Span ()

  Equivalend to Span(`nullptr`, `nullptr`).
- Span (T *begin, T *end)

  Constructs a span with `begin` as the start point and `end` as the end point.
- Span (T *begin, std::size_t size)

  If begin is null, equivalent to Span(`nullptr`, `nullptr`); otherwise, equivalent to Span(`begin`, `begin` + `size`).
- template<typename U , typename = typename std::enable_if< std::is_convertible<U(*)[ ], T(*)[ ]>::value>::type>
  constexpr Span (Span< U > that)

  Constructs a span that refers to the same sequence as `that`.
- constexpr std::size_t size () const

  Returns the number of elements in the referenced sequence.
- constexpr T * begin () const

  Returns the start point.
- constexpr T * end () const

  Returns the end point.
- T & operator[ ] (std::size_t i) const

  Returns a reference to the element of the referenced sequence with index `i`.
- Span subspan (std::size_t start, std::size_t end) const

  Returns a span with begin() + `start` as the start point and begin() + `end` as the end point.

5.5.1 Detailed Description

template<typename T>
class cva::ac::Span< T >

A non-owning reference to a (possibly-empty) contiguous sequence of objects of type `T`.

A span is defined by a pair of pointers: its start and end points. If the start and end points are the same, the span references an empty sequence. Otherwise, it references a sequence containing all objects that are between the one pointed to by the start point (inclusive) and the one pointed to by the end point (exclusive).

5.5.2 Constructor & Destructor Documentation

5.5.2.1 template<typename T> constexpr cva::ac::Span< T >::Span ( ) [inline]

Equivalend to Span(`nullptr`, `nullptr`).

5.5.2.2 template<typename T> cva::ac::Span< T >::Span (T * *begin*, T * *end*) [inline]

Constructs a span with `begin` as the start point and `end` as the end point.

Precondition

Either `begin` and `end` are both null, or neither are.
`begin` <= `end`

5.5.2.3 template<typename T> cva::ac::Span< T >::Span (T * *begin*, std::size_t *size*) [inline]

If begin is null, equivalent to Span(nullptr, nullptr); otherwise, equivalent to Span(begin, begin + size).

Precondition

Either size is 0, or begin is not null.

5.5.2.4 template<typename T> template<typename U , typename = typename std::enable_if< std::is_convertible<U(*)[ ], T(*)[ ]>::value>::type> constexpr cva::ac::Span< T >::Span (Span< U > *that*) [inline]

Constructs a span that refers to the same sequence as that.

Precondition

T is either U or U with additional cv-qualifiers.

5.5.3 Member Function Documentation

5.5.3.1 template<typename T> constexpr T* cva::ac::Span< T >::begin ( ) const [inline]

Returns the start point.

5.5.3.2 template<typename T> constexpr T* cva::ac::Span< T >::end ( ) const [inline]

Returns the end point.

5.5.3.3 template<typename T> T& cva::ac::Span< T >::operator[ ] (std::size_t *i*) const [inline]

Returns a reference to the element of the referenced sequence with index i.

Precondition

i < size()

5.5.3.4 template<typename T> constexpr std::size_t cva::ac::Span< T >::size ( ) const [inline]

Returns the number of elements in the referenced sequence.

5.5.3.5 template<typename T> Span cva::ac::Span< T >::subspan (std::size_t *start*, std::size_t *end*) const [inline]

Returns a span with begin() + `start` as the start point and begin() + `end` as the end point.

Precondition

`start` <= `end` <= size()

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

- span.hpp

5.6 cva::ac::Version Class Reference

Version number of a library which consists of major, minor and patch numbers.

`#include <api.hpp>`

Public Member Functions

- constexpr Version (std::uint32_t major=0, std::uint32_t minor=0, std::uint32_t patch=0)

  Constructs a version object with given `major` `minor` and `patch` numbers.
- constexpr std::uint32_t major () const

  Returns the major number of the version.
- constexpr std::uint32_t minor () const

  Returns the minor number of the version.
- constexpr std::uint32_t patch () const

  Returns the patch number of the version.
- std::string toString () const

  Returns the version string: major.minor.patch.

Friends

- bool operator== (const Version &left, const Version &right)

  Returns whether all components of `left` compare equal to those of `right`.
- bool operator!= (const Version &left, const Version &right)

  Returns the opposite of operator==().

5.6.1 Detailed Description

Version number of a library which consists of major, minor and patch numbers.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 constexpr cva::ac::Version::Version (std::uint32_t *major* = 0, std::uint32_t *minor* = 0, std::uint32_t *patch* = 0) [inline], [explicit]

Constructs a version object with given `major` `minor` and `patch` numbers.

5.6.3 Member Function Documentation

5.6.3.1 constexpr std::uint32_t cva::ac::Version::major ( ) const [inline]

Returns the major number of the version.

5.6.3.2 constexpr std::uint32_t cva::ac::Version::minor ( ) const [inline]

Returns the minor number of the version.

5.6.3.3 constexpr std::uint32_t cva::ac::Version::patch ( ) const [inline]

Returns the patch number of the version.

5.6.3.4 std::string cva::ac::Version::toString ( ) const [inline]

Returns the version string: major.minor.patch.

5.6.4 Friends And Related Function Documentation

5.6.4.1 bool operator!= (const Version & *left*, const Version & *right*) [friend]

Returns the opposite of operator==().

5.6.4.2 bool operator== (const Version & *left*, const Version & *right*) [friend]

Returns whether all components of `left` compare equal to those of `right`.

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

- api.hpp

5.7 cva::ac::View2d< T > Class Template Reference

A non-owning reference to a two-dimensional block of pixels, each of which is a contiguous sequence of one or more channels of type T.

`#include <image_view.hpp>`

Public Member Functions

- View2d (std::size_t width, std::size_t height, std::size_t nc, std::ptrdiff_t stride, T *base)

  Constructs a view to a `width` x `height` block pixels of `nc` channels each, with the given `stride` and `base`.
- template<typename U , typename = typename std::enable_if< std::is_convertible<U(*)[ ], T(*)[ ]>::value>::type>
  View2d (const View2d< U > &that)

  Constructs a view to the same block as `that`.
- std::size_t width () const

  Returns the width of the referenced block.
- std::size_t height () const

  Returns the height of the referenced block.
- std::size_t nc () const

  Returns the number of channels in a pixel of the referenced block.
- std::ptrdiff_t stride () const

  Returns the view's stride.
- T * base () const

  Returns the view's base.
- T * row (std::size_t y)

  Returns a pointer to the first pixel of the row with index `y`.

5.7.1 Detailed Description

template<typename T>
class cva::ac::View2d< T >

A non-owning reference to a two-dimensional block of pixels, each of which is a contiguous sequence of one or more channels of type T.

Every pixel has the same number of channels.

The pixels must be stored in row-major order. The pixels within each row must be contiguous, but the rows themselves don't have to be. The distance in bytes between the first pixels of each pair of adjacent rows must be the same and is called the view's stride.

A pointer to the pixel with index (0, 0) is called the view's base.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 template<typename T> cva::ac::View2d< T >::View2d (std::size_t *width*, std::size_t *height*, std::size_t *nc*, std::ptrdiff_t *stride*, T * *base*) [inline]

Constructs a view to a `width` x `height` block pixels of `nc` channels each, with the given `stride` and `base`.

Precondition

`width` > 0, `height` > 0, `nc` > 0
`stride` >= `width` * `nc` * `sizeof`(T)
`base` is not null.

5.7.2.2 template<typename T> template<typename U , typename = typename std::enable_if< std::is_convertible<U(*)[ ], T(*)[ ]>::value>::type> cva::ac::View2d< T >::View2d (const View2d< U > & *that*) [inline]

Constructs a view to the same block as `that`.

Precondition

`T` is either `U` or `U` with additional cv-qualifiers.

5.7.3 Member Function Documentation

5.7.3.1 template<typename T> T* cva::ac::View2d< T >::base ( ) const [inline]

Returns the view's base.

5.7.3.2 template<typename T> std::size_t cva::ac::View2d< T >::height ( ) const [inline]

Returns the height of the referenced block.

5.7.3.3 template<typename T> std::size_t cva::ac::View2d< T >::nc ( ) const [inline]

Returns the number of channels in a pixel of the referenced block.

5.7.3.4 template<typename T> T* cva::ac::View2d< T >::row (std::size_t *y*) [inline]

Returns a pointer to the first pixel of the row with index `y`.

Precondition

`y` < height()

5.7.3.5 template<typename T> std::ptrdiff_t cva::ac::View2d< T >::stride ( ) const [inline]

Returns the view's stride.

5.7.3.6 template<typename T> std::size_t cva::ac::View2d< T >::width ( ) const [inline]

Returns the width of the referenced block.

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

- inc/cva/ac/image_view.hpp

Chapter 6

File Documentation

6.1 api.hpp File Reference

```
#include <cstdint>
#include <string>
```

Classes

- class cva::ac::Version

  Version number of a library which consists of major, minor and patch numbers.

Namespaces

- cva
- cva::ac

Macros

- #define CVA_AC_SHARED_LIBRARY_EXPORT

  Expands to a platform-dependent qualifier that can be used to mark a definition as exported from the current shared library.
- #define CVA_AC_SHARED_LIBRARY_IMPORT

  Expands to a platform-dependent qualifier that can be used to mark a definition as imported from an external shared library.

6.1.1 Macro Definition Documentation

6.1.1.1 #define CVA_AC_SHARED_LIBRARY_EXPORT

Expands to a platform-dependent qualifier that can be used to mark a definition as exported from the current shared library.

On an unsupported platform, expands to nothing.

6.1.1.2 #define CVA_AC_SHARED_LIBRARY_IMPORT

Expands to a platform-dependent qualifier that can be used to mark a definition as imported from an external shared library.

On an unsupported platform, expands to nothing.

6.2 async.hpp File Reference

```
#include <cstdint>
```

Classes

- class cva::ac::PendingRequest

  A handle to an asynchronous operation that is either in progress or has completed.

Namespaces

- cva
- cva::ac

Enumerations

- enum cva::ac::WaitResult { cva::ac::WaitResult::COMPLETED, cva::ac::WaitResult::TIMED_OUT }

  Result of a wait for an asynchronous operation's completion.

6.3 common.hpp File Reference

```
#include "api.hpp"
```

Namespaces

- cva
- cva::ac

Macros

- #define CVA_AC_EXPORT CVA_AC_SHARED_LIBRARY_IMPORT

Functions

- Version cva::ac::version ()

  Returns version number of the library.

6.3.1 Macro Definition Documentation

6.3.1.1 #define CVA_AC_EXPORT CVA_AC_SHARED_LIBRARY_IMPORT

6.4 common.hpp File Reference

```
#include <cva/ac/api.hpp>
```

Macros

- #define CVA_AC_OCV_EXPORT CVA_AC_SHARED_LIBRARY_IMPORT

6.4.1 Macro Definition Documentation

6.4.1.1 #define CVA_AC_OCV_EXPORT CVA_AC_SHARED_LIBRARY_IMPORT

6.5 error_reporting.hpp File Reference

```
#include <sstream>
```

Namespaces

- cva
- cva::ac

Functions

- void cva::ac::outputToStream (std::ostream &)

  Does nothing.
- template<typename Arg0 , typename... Args>
  void cva::ac::outputToStream (std::ostream &os, const Arg0 &arg0, const Args &...args)

  Streams `arg0` and every element of `args` into `os`, in turn.
- template<typename E , typename... Args>
  void cva::ac::failRequirement (const char *class_name, const char *function_name, const Args &...args)

  Throws an exception of type `E` constructed with a string created from `class_name`, `function_name`, and `args`.
- template<typename E , typename... Args>
  void cva::ac::require (const char *class_name, const char *function_name, bool condition, const Args &...args)

  Calls failRequirement(`class_name`, `function_name`, `args`...) unless `condition` is true.

6.6 example.dox File Reference

6.7 geometry.hpp File Reference

```
#include "error_reporting.hpp"
#include <stdexcept>
#include <type_traits>
#include <algorithm>
```

Classes

- struct cva::ac::Point2< T >

  A point in two-dimensional space whose coordinates are of type `T`.
- class cva::ac::Rectangle< T >

  A rectangle of points with coordinates of type `T`.

Namespaces

- cva
- cva::ac

6.8 image_view.hpp File Reference

```
#include "common.hpp"
#include "error_reporting.hpp"
#include <cstdlib>
#include <stdexcept>
#include <type_traits>
```

Classes

- class cva::ac::View2d< T >

  A non-owning reference to a two-dimensional block of pixels, each of which is a contiguous sequence of one or more channels of type T.
- class cva::ac::ImageViewT< T >

  A non-owning reference to an image.

Namespaces

- cva
- cva::ac

Typedefs

- using cva::ac::ConstImageView = ImageViewT< const void >
- using cva::ac::ImageView = ImageViewT< void >

Enumerations

- enum cva::ac::ImageFormat { cva::ac::ImageFormat::LABEL_8, cva::ac::ImageFormat::INTENSITY_8, cva::ac::ImageFormat::RGB_8, cva::ac::ImageFormat::BGR_8 }

  An identifier of a specific number of channels and channel type of a pixel, as well as an intended interpretation of such a pixel.

Functions

- std::size_t cva::ac::imageFormatNumChannels (ImageFormat format)

  Returns the number of channels identified by `format`.
- void cva::ac::imageFormatRequireMatchingNumChannels (const char *class_name, const char *function_name, ImageFormat format, const char *format_str, std::size_t nc, const char *nc_str)

  Throws an exception if `nc` *is not imageFormatNumChannels(*`format`*).*
- template<typename T >
  void cva::ac::imageFormatRequireMatchingType (const char *class_name, const char *function_name, ImageFormat format, const char *format_str, const char *type_str)

  Throws an exception if `T` *does not match* `format`.

6.9 image_view.hpp File Reference

```
#include "common.hpp"
#include <cva/ac/image_view.hpp>
#include <opencv2/core.hpp>
```

Namespaces

- cva
- cva::ac
- cva::ac::ocv

Functions

- CVA_AC_SHARED_LIBRARY_IMPORT ImageView cva::ac::ocv::toImageView (ImageFormat format, const cv::Mat &mat)

6.10 precision.hpp File Reference

```
#include "common.hpp"
#include <iosfwd>
```

Namespaces

- cva
- cva::ac

Enumerations

- enum cva::ac::Precision {
  cva::ac::Precision::FP16, cva::ac::Precision::FP32, cva::ac::Precision::I8, cva::ac::Precision::I16,
  cva::ac::Precision::I32, cva::ac::Precision::Q78, cva::ac::Precision::U8, cva::ac::Precision::U16 }

  A number's precision/range.

Functions

- std::ostream & cva::ac::operator<< (std::ostream &os, Precision p)

  Streams a string representation of `p` *into* `os`*.*

6.11 span.hpp File Reference

```
#include "error_reporting.hpp"
#include <cstdlib>
#include <stdexcept>
#include <type_traits>
#include <vector>
```

Classes

- class cva::ac::Span< T >

  A non-owning reference to a (possibly-empty) contiguous sequence of objects of type `T`*.*

Namespaces

- cva
- cva::ac

Functions

- template<typename T >
  Span< T > cva::ac::toSpan (Span< T > span)

  Returns a copy of `span`*.*
- template<typename T , std::size_t N>
  Span< T > cva::ac::toSpan (T(&array)[N])

  Returns a span that references the sequence of all elements of `array`*.*
- template<typename T , typename Allocator >
  Span< T > cva::ac::toSpan (std::vector< T, Allocator > &vector)

  Returns a span that references the sequence of all elements of `vector`*.*
- template<typename T , typename Allocator >
  Span< const T > cva::ac::toSpan (const std::vector< T, Allocator > &vector)