Image Provider Interface Demo

1.0

Generated by Doxygen 1.8.17

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

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

Class Documentation

4.1 Buffer< Type, size > Class Template Reference

Ring buffer with mutex.

`#include <buffer.h>`

Public Member Functions

- **Buffer** ()
- ~**Buffer** ()=default
- unsigned int **Size** () const
- bool **Empty** () const

  Is this buffer empty?
- void **Push** (const Type &obj)

  Push an element.
- bool **Pop** (Type &obj)

  Pop an element.
- Type & **operator[ ]** (unsigned int id)

4.1.1 Detailed Description

template<typename Type, unsigned int size>
class Buffer< Type, size >

Ring buffer with mutex.

Template Parameters

Type	Type of elements in this ring buffer.
size	Max size of this ring buffer.

Attention

Size must be 2^N.

Definition at line 16 of file buffer.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Buffer()

```
template<typename Type , unsigned int size>
Buffer< Type, size >::  Buffer ( )  [inline]
```

Definition at line 26 of file buffer.h.

4.1.2.2 ~Buffer()

```
template<typename Type , unsigned int size>
Buffer< Type, size >::~ Buffer ( )  [default]
```

4.1.3 Member Function Documentation

4.1.3.1 Empty()

```
template<typename Type , unsigned int size>
bool  Buffer< Type, size >::Empty ( ) const  [inline]
```

Is this buffer empty?

Returns

A boolean shows if ring buffer is empty.

Definition at line 39 of file buffer.h.

4.1.3.2 operator[]()

```
template<typename Type , unsigned int size>
Type&  Buffer< Type, size >::operator[] (
             unsigned int id )  [inline]
```

Definition at line 74 of file buffer.h.

4.1.3.3 Pop()

```
template<typename Type , unsigned int size>
bool  Buffer< Type, size >::Pop (
             Type & obj )  [inline]
```

Pop an element.

Parameters

out	*obj*	Output element.

Returns

A boolean shows if ring buffer is not empty.

Definition at line 63 of file buffer.h.

4.1.3.4 Push()

```
template<typename Type , unsigned int size>
void  Buffer< Type, size >::Push (
             const Type & obj )  [inline]
```

Push an element.

Parameters

in	*obj*	Input element.

Definition at line 45 of file buffer.h.

4.1.3.5 Size()

```
template<typename Type , unsigned int size>
unsigned int  Buffer< Type, size >::Size ( ) const  [inline]
```

Returns

Size of this buffer, which is specified when it is constructed.

Definition at line 33 of file buffer.h.

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

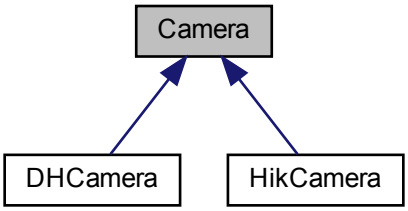
- modules/image-provider-base/ **buffer.h**

4.2 Camera Class Reference

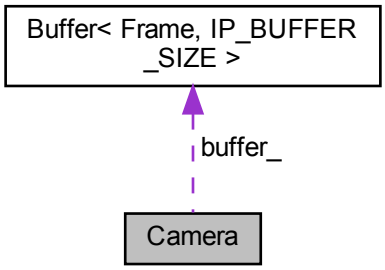
Camera (p. 10) base class.

`#include <camera_base.h>`

Inheritance diagram for Camera:



Collaboration diagram for Camera:



Public Member Functions

- **Camera** ()
- virtual ~**Camera** ()=default
- virtual bool **OpenCamera** (const std::string &serial_number, const std::string &config_file)=0

  Open a camera.
- virtual bool **CloseCamera** ()=0

  Close the opened camera.
- virtual bool **GetFrame** (**Frame** &frame)=0

  Get a frame with image and time stamp from internal image buffer.
- virtual bool **StartStream** ()=0

  Start the stream.

- virtual bool **StopStream** ()=0

  Stop the stream.
- virtual bool **IsConnected** ()=0

  Check if current device is connected.
- virtual bool **ImportConfigurationFile** (const std::string &file_path)=0

  Import current config to specified file.
- virtual bool **ExportConfigurationFile** (const std::string &file_path)=0

  Export current config to specified file.
- virtual bool **SetExposureTime** (uint32_t exposure_time)=0

  Set exposure time.
- virtual bool **SetGainValue** (float gain)=0

  Set gain value.
- std::string **GetSerialNumber** ()

  Get serial number.

Protected Attributes

- std::string **serial_number_**
- bool **stream_running_**
- pthread_t **daemon_thread_id_**
- bool **stop_daemon_thread_flag_**
- **Buffer**< **Frame**, **IP_BUFFER_SIZE** > **buffer_**

4.2.1 Detailed Description

Camera (p. 10) base class.

Note

You cannot directly construct objects.
Instead, find camera types in subclass documents, include **camera_factory.h** (p. 60) and use CF_CREAT↩
E_CAMERA macro.

Definition at line 13 of file camera_base.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Camera()

```
Camera::Camera ( )  [inline]
```

Definition at line 15 of file camera_base.h.

4.2.2.2 ~Camera()

```
virtual Camera::~Camera ( )  [virtual], [default]
```

4.2.3 Member Function Documentation

4.2.3.1 CloseCamera()

```
virtual bool Camera::CloseCamera ( )  [pure virtual]
```

Close the opened camera.

Returns

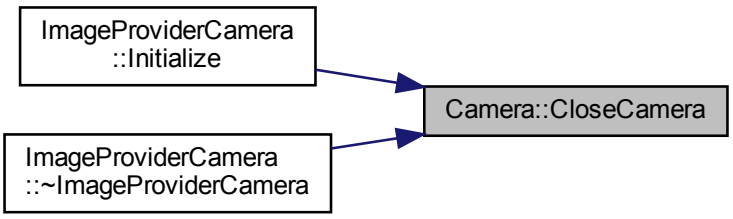
A boolean shows whether the camera is normally closed.

Attention

No matter what is returned, the camera will not be valid.

Implemented in **DHCamera** (p. 28), and **HikCamera** (p. 37).

Here is the caller graph for this function:



4.2.3.2 ExportConfigurationFile()

```
virtual bool Camera::ExportConfigurationFile (
            const std::string & file_path )  [pure virtual]
```

Export current config to specified file.

Parameters

in	*file_path*	File path.

Returns

A boolean shows if config is successfully saved.

Implemented in **DHCamera** (p. 28), and **HikCamera** (p. 37).

4.2.3.3 GetFrame()

```
virtual bool Camera::GetFrame (
              Frame & frame )  [pure virtual]
```

Get a frame with image and time stamp from internal image buffer.

Parameters

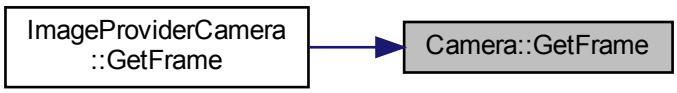
out	*frame*	Acquired frame will be stored here.

Returns

A boolean shows if buffer is not empty, or if you can successfully get an frame.

Implemented in **DHCamera** (p. 29), and **HikCamera** (p. 38).

Here is the caller graph for this function:



4.2.3.4 GetSerialNumber()

```
std::string Camera::GetSerialNumber ( )  [inline]
```

Get serial number.

Returns

Serial number of this camera.

Definition at line 95 of file camera_base.h.

4.2.3.5 ImportConfigurationFile()

```
virtual bool Camera::ImportConfigurationFile (
            const std::string & file_path )  [pure virtual]
```

Import current config to specified file.

Parameters

in	*file_path*	File path.

Returns

A boolean shows if config is successfully imported.

Implemented in **DHCamera** (p. 29), and **HikCamera** (p. 38).

4.2.3.6 IsConnected()

```
virtual bool Camera::IsConnected ( )  [pure virtual]
```

Check if current device is connected.

Returns

A boolean shows current device is connected.

Implemented in **DHCamera** (p. 30), and **HikCamera** (p. 39).

4.2.3.7 OpenCamera()

```
virtual bool Camera::OpenCamera (
            const std::string & serial_number,
            const std::string & config_file )  [pure virtual]
```

Open a camera.

Parameters

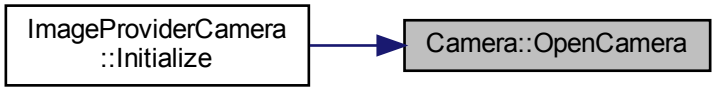
in	*serial_number*	Serial number of the camera you wanna open.
in	*config_file*	Will load config from this file.

Returns

A boolean shows whether the camera is successfully opened.

Implemented in **DHCamera** (p. 30), and **HikCamera** (p. 39).

Here is the caller graph for this function:



4.2.3.8 SetExposureTime()

```
virtual bool Camera::SetExposureTime (
            uint32_t exposure_time )  [pure virtual]
```

Set exposure time.

Parameters

exposure_time	Exposure time, automatically converted to corresponding data type.

Returns

A boolean shows if exposure time is successfully set.

Implemented in **DHCamera** (p. 31), and **HikCamera** (p. 40).

4.2.3.9 SetGainValue()

```
virtual bool Camera::SetGainValue (
            float gain )  [pure virtual]
```

Set gain value.

Parameters

gain	Gain value, automatically converted to corresponding data type.

Returns

A boolean shows if gain value is successfully set.

Implemented in **DHCamera** (p. 32), and **HikCamera** (p. 41).

4.2.3.10 StartStream()

```
virtual bool Camera::StartStream ( )  [pure virtual]
```

Start the stream.

Returns

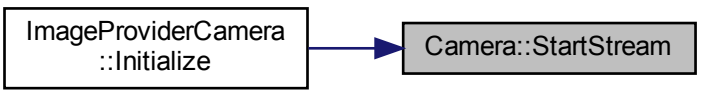
Whether stream is started normally.

Attention

This function will return false when stream is already started or camera is not opened.

Implemented in **DHCamera** (p. 32), and **HikCamera** (p. 41).

Here is the caller graph for this function:



4.2.3.11 StopStream()

```
virtual bool Camera::StopStream ( )  [pure virtual]
```

Stop the stream.

Returns

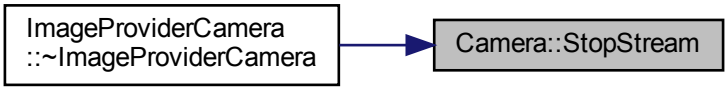
Whether stream is stopped normally.

Attention

This function will return false when stream is not started or camera is not opened.

Implemented in **DHCamera** (p. 32), and **HikCamera** (p. 41).

Here is the caller graph for this function:



4.2.4 Member Data Documentation

4.2.4.1 buffer_

```
Buffer< Frame, IP_BUFFER_SIZE> Camera::buffer_  [protected]
```

Definition at line 102 of file camera_base.h.

4.2.4.2 daemon_thread_id_

```
pthread_t Camera::daemon_thread_id_  [protected]
```

Definition at line 100 of file camera_base.h.

4.2.4.3 serial_number_

```
std::string Camera::serial_number_  [protected]
```

Definition at line 98 of file camera_base.h.

4.2.4.4 stop_daemon_thread_flag_

```
bool Camera::stop_daemon_thread_flag_  [protected]
```

Definition at line 101 of file camera_base.h.

4.2.4.5 stream_running_

```
bool Camera::stream_running_  [protected]
```

Definition at line 99 of file camera_base.h.

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

- modules/camera-base/ **camera_base.h**

4.3 CameraFactory Class Reference

Singleton camera factory.

```
#include <camera_factory.h>
```

Public Member Functions

- **CameraFactory** (const **CameraFactory** &)=delete
- **CameraFactory** & **operator=** (const **CameraFactory** &)=delete
- void **RegisterCamera** (const std::string &camera_type_name, **CameraRegistryBase** *registry)
  Register a camera type.
- **Camera** * **CreateCamera** (const std::string &camera_type_name)
  Create a camera whose type is registered to factory.

Static Public Member Functions

- static **CameraFactory** & **Instance** ()
  Get the only instance of camera factory.

4.3.1 Detailed Description

Singleton camera factory.

Use **CameraFactory::Instance()** (p. 20) to get the only instance pointer.
Use macro **CF_CREATE_CAMERA(camera_type_name)** (p. 61) to create a camera instance.
Use CameraFactory::Instance::RegisterCamera(camera_type_name, ∗registry) to register a type of camera.

Warning

Camera (p. 10) factory will not check whether CameraType is really subclass of **Camera** (p. 10) base class. (Thus, you should ensure that all callings of **CameraRegistry** (p. 21) constructor are completely under control.)

Definition at line 41 of file camera_factory.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 CameraFactory()

```
CameraFactory::CameraFactory (
            const  CameraFactory &  )  [delete]
```

4.3.3 Member Function Documentation

4.3.3.1 CreateCamera()

```
Camera* CameraFactory::CreateCamera (
            const std::string & camera_type_name )  [inline]
```

Create a camera whose type is registered to factory.

Parameters

in	*camera_type_name*	Type name of camera.

Returns

A pointer to crated camera.

Note

You may use macro **CF_CREATE_CAMERA(camera_type_name)** (p. 61) instead of call this function.

Definition at line 72 of file camera_factory.h.

4.3.3.2 Instance()

```
static CameraFactory& CameraFactory::Instance ( ) [inline], [static]
```

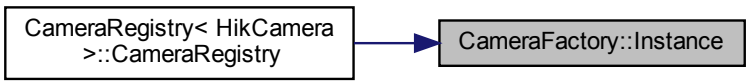
Get the only instance of camera factory.

Returns

A camera factory object.

Definition at line 51 of file camera_factory.h.

Here is the caller graph for this function:



4.3.3.3 operator=()

```
CameraFactory& CameraFactory::operator= (
            const CameraFactory & ) [delete]
```

4.3.3.4 RegisterCamera()

```
void CameraFactory::RegisterCamera (
            const std::string & camera_type_name,
             CameraRegistryBase * registry ) [inline]
```

Register a camera type.

Parameters

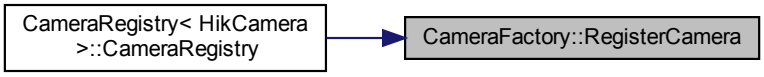
in	*camera_type_name*	Type name of camera.
in	*registry*	A registry object of camera.

Warning

You may call this function only when you're programming for a new type of camera.

Definition at line 62 of file camera_factory.h.

Here is the caller graph for this function:



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

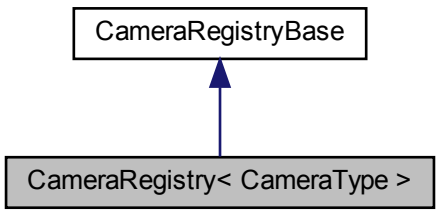
- modules/camera-base/ **camera_factory.h**

4.4 CameraRegistry< CameraType > Class Template Reference

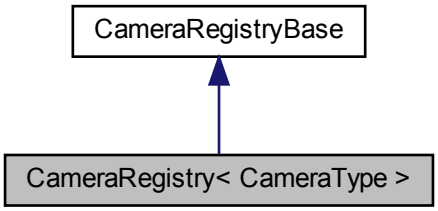
Templated camera registry class.

`#include <camera_factory.h>`

Inheritance diagram for CameraRegistry< CameraType >:



Collaboration diagram for CameraRegistry< CameraType >:



Public Member Functions

- **CameraRegistry** (const std::string &camera_type_name)
  Constructor of camera registry.
- **Camera** * **CreateCamera** () final
  Create a camera of this type.

Additional Inherited Members

4.4.1 Detailed Description

template<class CameraType>
class CameraRegistry< CameraType >

Templated camera registry class.

Template Parameters

CameraType	**Camera** (p. 10) type inherited from base class **Camera** (p. 10).

Attention

Once object is constructed, this type of camera will immediately be registered to camera factory.
This means the constructed object is useless and should not appear in any other place.
(Thus, template class though this is, it's better to be treated as a function.)

Warning

Camera (p. 10) factory will not check whether CameraType is really subclass of **Camera** (p. 10) base class.
(Thus, you should ensure that all callings of **CameraRegistry** (p. 21) constructor are completely under control.)

Definition at line 101 of file camera_factory.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 CameraRegistry()

```
template<class CameraType >
CameraRegistry< CameraType >::  CameraRegistry (
            const std::string & camera_type_name )  [inline], [explicit]
```

Constructor of camera registry.

Parameters

in	*camera_type_name*	Type name of camera.

Definition at line 107 of file camera_factory.h.

4.4.3 Member Function Documentation

4.4.3.1 CreateCamera()

```
template<class CameraType >
Camera* CameraRegistry< CameraType >::CreateCamera ( )  [inline], [final], [virtual]
```

Create a camera of this type.

Returns

A camera pointer.

Warning

NEVER directly call this function. Instead, it should be called by camera factory.

Implements **CameraRegistryBase** (p. 25).

Definition at line 116 of file camera_factory.h.

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

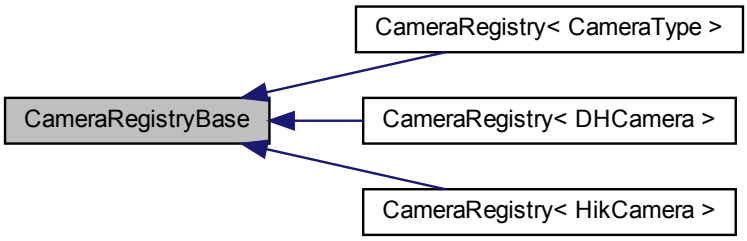
- modules/camera-base/ **camera_factory.h**

4.5 CameraRegistryBase Class Reference

Base class of camera registry.

`#include <camera_factory.h>`

Inheritance diagram for CameraRegistryBase:



Public Member Functions

- virtual **Camera** * **CreateCamera** ()=0
- **CameraRegistryBase** (const **CameraRegistryBase** &)=delete
- **CameraRegistryBase** & **operator=** (const **CameraRegistryBase** &)=delete

Protected Member Functions

- **CameraRegistryBase** ()=default
- virtual ~**CameraRegistryBase** ()=default

4.5.1 Detailed Description

Base class of camera registry.

Warning

You should use its subclass **CameraRegistry** (p. 21) instead of this base class.

Definition at line 17 of file camera_factory.h.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 CameraRegistryBase() [1/2]

```
CameraRegistryBase::CameraRegistryBase (
            const  CameraRegistryBase &  )  [delete]
```

4.5.2.2 CameraRegistryBase() [2/2]

```
CameraRegistryBase::CameraRegistryBase ( )  [protected], [default]
```

4.5.2.3 ~CameraRegistryBase()

```
virtual CameraRegistryBase::~CameraRegistryBase ( )  [protected], [virtual], [default]
```

4.5.3 Member Function Documentation

4.5.3.1 CreateCamera()

```
virtual  Camera* CameraRegistryBase::CreateCamera ( )  [pure virtual]
```

Implemented in **CameraRegistry< CameraType >** (p. 23), **CameraRegistry< DHCamera >** (p. 23), and **CameraRegistry< HikCamera >** (p. 23).

4.5.3.2 operator=()

```
CameraRegistryBase& CameraRegistryBase::operator= (
            const  CameraRegistryBase &  )  [delete]
```

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

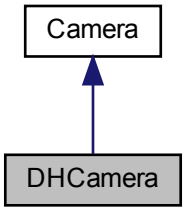
- modules/camera-base/ **camera_factory.h**

4.6 DHCamera Class Reference

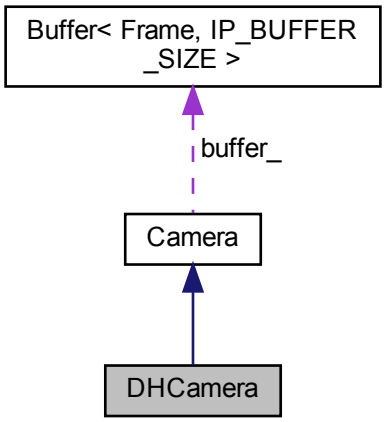
DaHeng camera class implementation.

`#include <camera_dh.h>`

Inheritance diagram for DHCamera:



Collaboration diagram for DHCamera:



Public Member Functions

- **DHCamera** ()
- **DHCamera** (const **DHCamera** &)=delete
- **DHCamera** & **operator=** (const **DHCamera** &)=delete
- ~**DHCamera** () final=default
- bool **OpenCamera** (const std::string &serial_number, const std::string &config_file) final

  Open a camera.

- bool **CloseCamera** () final

  Close the opened camera.
- bool **StartStream** () final

  Start the stream.
- bool **StopStream** () final

  Stop the stream.
- bool **IsConnected** () final

  Check if current device is connected.
- bool **GetFrame** (**Frame** &frame) final

  Get a frame with image and time stamp from internal image buffer.
- bool **ExportConfigurationFile** (const std::string &file_path) final

  Export current config to specified file.
- bool **ImportConfigurationFile** (const std::string &file_path) final

  Import current config to specified file.
- bool **SetExposureTime** (uint32_t exposure_time) final

  Set exposure time.
- bool **SetGainValue** (float gain_value) final

  Set gain value.

Additional Inherited Members

4.6.1 Detailed Description

DaHeng camera class implementation.

Warning

NEVER directly use this class to create camera!
Instead, turn to **CameraFactory** (p. 18) class and use CF_CREATE_CAMERA("DHCamera").

Definition at line 62 of file camera_dh.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 DHCamera() [1/2]

```
DHCamera::DHCamera ( )  [inline]
```

Definition at line 64 of file camera_dh.h.

4.6.2.2 DHCamera() [2/2]

```
DHCamera::DHCamera (
            const  DHCamera &  )  [delete]
```

4.6.2.3 ~DHCamera()

```
DHCamera::~DHCamera ( )  [final], [default]
```

4.6.3 Member Function Documentation

4.6.3.1 CloseCamera()

```
bool DHCamera::CloseCamera ( )  [final], [virtual]
```

Close the opened camera.

Returns

A boolean shows whether the camera is normally closed.

Attention

No matter what is returned, the camera will not be valid.

Implements **Camera** (p. 12).

Definition at line 118 of file camera_dh.cpp.

4.6.3.2 ExportConfigurationFile()

```
bool DHCamera::ExportConfigurationFile (
            const std::string & file_path )  [inline], [final], [virtual]
```

Export current config to specified file.

Parameters

in	*file_path*	File path.

Returns

A boolean shows if config is successfully saved.

Implements **Camera** (p. 12).

Definition at line 88 of file camera_dh.h.

Here is the caller graph for this function:

4.6.3.3 GetFrame()

```
bool DHCamera::GetFrame (
              Frame & frame )  [inline], [final], [virtual]
```

Get a frame with image and time stamp from internal image buffer.

Parameters

out	*frame*	Acquired frame will be stored here.

Returns

A boolean shows if buffer is not empty, or if you can successfully get an frame.

Implements **Camera** (p. 13).

Definition at line 86 of file camera_dh.h.

4.6.3.4 ImportConfigurationFile()

```
bool DHCamera::ImportConfigurationFile (
             const std::string & file_path )  [inline], [final], [virtual]
```

Import current config to specified file.

Parameters

in	*file_path*	File path.

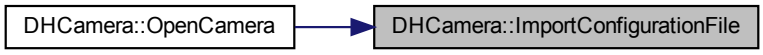
Returns

A boolean shows if config is successfully imported.

Implements **Camera** (p. 14).

Definition at line 95 of file camera_dh.h.

Here is the caller graph for this function:



4.6.3.5 IsConnected()

```
bool DHCamera::IsConnected ( )  [final], [virtual]
```

Check if current device is connected.

Returns

A boolean shows current device is connected.

Implements **Camera** (p. 14).

Definition at line 214 of file camera_dh.cpp.

4.6.3.6 OpenCamera()

```
bool DHCamera::OpenCamera (
            const std::string & serial_number,
            const std::string & config_file )  [final], [virtual]
```

Open a camera.

Parameters

in	*serial_number*	Serial number of the camera you wanna open.
in	*config_file*	Will load config from this file.

Returns

A boolean shows whether the camera is successfully opened.

Implements **Camera** (p. 14).

Definition at line 17 of file camera_dh.cpp.

Here is the call graph for this function:

4.6.3.7 operator=()

```
DHCamera& DHCamera::operator= (
            const  DHCamera &  )  [delete]
```

4.6.3.8 SetExposureTime()

```
bool DHCamera::SetExposureTime (
            uint32_t exposure_time )  [inline], [final], [virtual]
```

Set exposure time.

Parameters

exposure_time	Exposure time, automatically converted to corresponding data type.

Returns

A boolean shows if exposure time is successfully set.

Implements **Camera** (p. 15).

Definition at line 102 of file camera_dh.h.

4.6.3.9 SetGainValue()

```
bool DHCamera::SetGainValue (
            float gain )  [inline], [final], [virtual]
```

Set gain value.

Parameters

gain	Gain value, automatically converted to corresponding data type.

Returns

A boolean shows if gain value is successfully set.

Implements **Camera** (p. 15).

Definition at line 114 of file camera_dh.h.

4.6.3.10 StartStream()

```
bool DHCamera::StartStream ( )  [final], [virtual]
```

Start the stream.

Returns

Whether stream is started normally.

Attention

This function will return false when stream is already started or camera is not opened.

Implements **Camera** (p. 16).

Definition at line 170 of file camera_dh.cpp.

Here is the call graph for this function:

4.6.3.11 StopStream()

```
bool DHCamera::StopStream ( )  [final], [virtual]
```

Stop the stream.

Returns

Whether stream is stopped normally.

Attention

This function will return false when stream is not started or camera is not opened.

Implements **Camera** (p. 16).

Definition at line 190 of file camera_dh.cpp.

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

- modules/camera-dh/ **camera_dh.h**
- modules/camera-dh/ **camera_dh.cpp**

4.7 Frame Struct Reference

Single frame structure.

```
#include <frame.h>
```

Public Member Functions

- **Frame** (cv::Mat &_image, uint64_t _time_stamp)
- **Frame** ()

Public Attributes

- cv::Mat **image**
- uint64_t **time_stamp**

4.7.1 Detailed Description

Single frame structure.

2 ways of initializing method provided:
(Default) Directly use **Frame()** (p. 34) to initialize an empty and useless frame.
(Manual) Use **Frame(_image, _time_stamp)** (p. 33) to initialize a complete frame.

Definition at line 12 of file frame.h.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 Frame() [1/2]

```
Frame::Frame (
            cv::Mat & _image,
            uint64_t _time_stamp )  [inline]
```

Definition at line 16 of file frame.h.

4.7.2.2 Frame() [2/2]

```
Frame::Frame ( )  [inline]
```

Definition at line 18 of file frame.h.

4.7.3 Member Data Documentation

4.7.3.1 image

```
cv::Mat Frame::image
```

Definition at line 13 of file frame.h.

4.7.3.2 time_stamp

```
uint64_t Frame::time_stamp
```

Definition at line 14 of file frame.h.

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

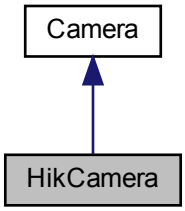
- modules/image-provider-base/ **frame.h**

4.8 HikCamera Class Reference

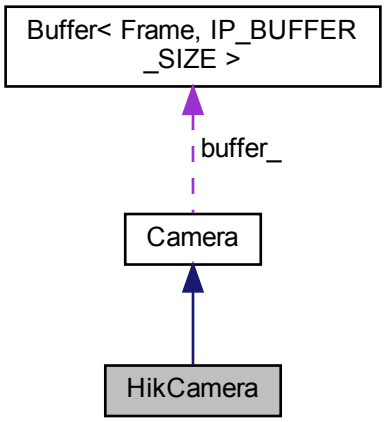
HikRobot camera class implementation.

`#include <camera_hik.h>`

Inheritance diagram for HikCamera:



Collaboration diagram for HikCamera:



Public Member Functions

- **HikCamera** ()
- **HikCamera** (const **HikCamera** &)=delete
- **HikCamera** & **operator=** (const **HikCamera** &)=delete
- ~**HikCamera** () final=default
- bool **OpenCamera** (const std::string &, const std::string &) final

  Open a camera.

- bool **StartStream** () final

  Start the stream.
- bool **GetFrame** (**Frame** &frame) final

  Get a frame with image and time stamp from internal image buffer.
- bool **StopStream** () final

  Stop the stream.
- bool **CloseCamera** () final

  Close the opened camera.
- bool **IsConnected** () final

  Check if current device is connected.
- bool **ExportConfigurationFile** (const std::string &file_path) final

  Export current config to specified file.
- bool **ImportConfigurationFile** (const std::string &file_path) final

  Import current config to specified file.
- bool **SetExposureTime** (uint32_t exposure_time) final

  Set exposure time.
- bool **SetGainValue** (float gain) final

  Set gain value.

Additional Inherited Members

4.8.1 Detailed Description

HikRobot camera class implementation.

Warning

NEVER directly use this class to create camera!
Instead, turn to **CameraFactory** (p. 18) class and use CF_CREATE_CAMERA("HikCamera").

Definition at line 15 of file camera_hik.h.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 HikCamera() [1/2]

```
HikCamera::HikCamera ( )  [inline]
```

Definition at line 17 of file camera_hik.h.

4.8.2.2 HikCamera() [2/2]

```
HikCamera::HikCamera (
            const  HikCamera &  )  [delete]
```

4.8.2.3 ~HikCamera()

```
HikCamera::~HikCamera ( )  [final], [default]
```

4.8.3 Member Function Documentation

4.8.3.1 CloseCamera()

```
bool HikCamera::CloseCamera ( )  [final], [virtual]
```

Close the opened camera.

Returns

A boolean shows whether the camera is normally closed.

Attention

No matter what is returned, the camera will not be valid.

Implements **Camera** (p. 12).

Definition at line 202 of file camera_hik.cpp.

4.8.3.2 ExportConfigurationFile()

```
bool HikCamera::ExportConfigurationFile (
            const std::string & file_path )  [inline], [final], [virtual]
```

Export current config to specified file.

Parameters

in	*file_path*	File path.

Returns

A boolean shows if config is successfully saved.

Implements **Camera** (p. 12).

Definition at line 40 of file camera_hik.h.

Here is the caller graph for this function:

4.8.3.3 GetFrame()

```
bool HikCamera::GetFrame (
              Frame & frame )  [inline], [final], [virtual]
```

Get a frame with image and time stamp from internal image buffer.

Parameters

out	*frame*	Acquired frame will be stored here.

Returns

A boolean shows if buffer is not empty, or if you can successfully get an frame.

Implements **Camera** (p. 13).

Definition at line 29 of file camera_hik.h.

4.8.3.4 ImportConfigurationFile()

```
bool HikCamera::ImportConfigurationFile (
             const std::string & file_path )  [inline], [final], [virtual]
```

Import current config to specified file.

Parameters

in	*file_path*	File path.

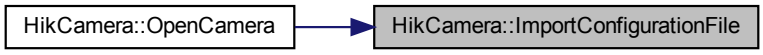
Returns

A boolean shows if config is successfully imported.

Implements **Camera** (p. 14).

Definition at line 51 of file camera_hik.h.

Here is the caller graph for this function:



4.8.3.5 IsConnected()

```
bool HikCamera::IsConnected ( )  [inline], [final], [virtual]
```

Check if current device is connected.

Returns

A boolean shows current device is connected.

Implements **Camera** (p. 14).

Definition at line 35 of file camera_hik.h.

4.8.3.6 OpenCamera()

```
bool HikCamera::OpenCamera (
            const std::string & serial_number,
            const std::string & config_file )  [final], [virtual]
```

Open a camera.

Parameters

in	*serial_number*	Serial number of the camera you wanna open.
in	*config_file*	Will load config from this file.

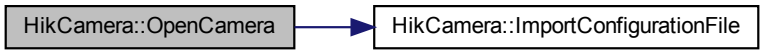
Returns

A boolean shows whether the camera is successfully opened.

Implements **Camera** (p. 14).

Definition at line 15 of file camera_hik.cpp.

Here is the call graph for this function:



4.8.3.7 operator=()

```
HikCamera& HikCamera::operator= (
            const  HikCamera &  )  [delete]
```

4.8.3.8 SetExposureTime()

```
bool HikCamera::SetExposureTime (
            uint32_t exposure_time )  [inline], [final], [virtual]
```

Set exposure time.

Parameters

exposure_time	Exposure time, automatically converted to corresponding data type.

Returns

A boolean shows if exposure time is successfully set.

Implements **Camera** (p. 15).

Definition at line 62 of file camera_hik.h.

4.8.3.9 SetGainValue()

```
bool HikCamera::SetGainValue (
            float gain )  [inline], [final], [virtual]
```

Set gain value.

Parameters

gain	Gain value, automatically converted to corresponding data type.

Returns

A boolean shows if gain value is successfully set.

Implements **Camera** (p. 15).

Definition at line 66 of file camera_hik.h.

4.8.3.10 StartStream()

```
bool HikCamera::StartStream ( )  [final], [virtual]
```

Start the stream.

Returns

Whether stream is started normally.

Attention

This function will return false when stream is already started or camera is not opened.

Implements **Camera** (p. 16).

Definition at line 146 of file camera_hik.cpp.

Here is the call graph for this function:

4.8.3.11 StopStream()

```
bool HikCamera::StopStream ( )  [final], [virtual]
```

Stop the stream.

Returns

Whether stream is stopped normally.

Attention

This function will return false when stream is not started or camera is not opened.

Implements **Camera** (p. 16).

Definition at line 185 of file camera_hik.cpp.

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

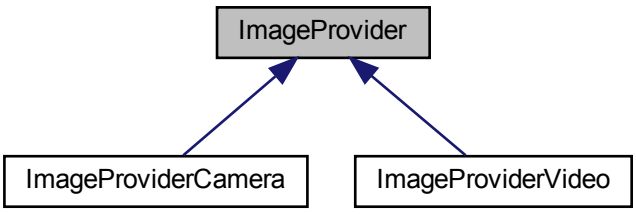
- modules/camera-hik/ **camera_hik.h**
- modules/camera-hik/ **camera_hik.cpp**

4.9 ImageProvider Class Reference

Image provider base class.

```
#include <image_provider_base.h>
```

Inheritance diagram for ImageProvider:



Public Member Functions

- **ImageProvider** ()=default
- virtual ~**ImageProvider** ()=default
- **ImageProvider** (const **ImageProvider** &)=delete
- **ImageProvider** & **operator=** (const **ImageProvider** &)=delete
- virtual bool **Initialize** (const std::string &file_path)=0

  Initialize by specified configuration file.
- virtual bool **GetFrame** (**Frame** &frame)=0

  Get a frame.

Protected Attributes

- cv::Mat **intrinsic_mat_**
- cv::Mat **distortion_mat_**

4.9.1 Detailed Description

Image provider base class.

Note

You cannot directly construct objects.
Instead, find camera types in subclass documents, include **image_provider_factory.h** (p. 69) and use IPF↩
_CREATE_IMAGE_PROVIDER macro.

Definition at line 13 of file image_provider_base.h.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 ImageProvider() [1/2]

```
ImageProvider::ImageProvider ( )  [default]
```

4.9.2.2 ~ImageProvider()

```
virtual ImageProvider::~ImageProvider ( )  [virtual], [default]
```

4.9.2.3 ImageProvider() [2/2]

```
ImageProvider::ImageProvider (
            const  ImageProvider &  )  [delete]
```

4.9.3 Member Function Documentation

4.9.3.1 GetFrame()

```
virtual bool ImageProvider::GetFrame (
              Frame & frame )  [pure virtual]
```

Get a frame.

Parameters

out	*frame*	OpenCV image reference.

Returns

A boolean shows if frame is complete.

Implemented in **ImageProviderVideo** (p. 57), and **ImageProviderCamera** (p. 47).

4.9.3.2 Initialize()

```
virtual bool ImageProvider::Initialize (
            const std::string & file_path )  [pure virtual]
```

Initialize by specified configuration file.

Parameters

in	*file_path*	Configuration file path.

Returns

A boolean shows if initialization succeeded.

Implemented in **ImageProviderVideo** (p. 58), and **ImageProviderCamera** (p. 47).

4.9.3.3 operator=()

```
ImageProvider& ImageProvider::operator= (
            const  ImageProvider &  )  [delete]
```

4.9.4 Member Data Documentation

4.9.4.1 distortion_mat_

```
cv::Mat ImageProvider::distortion_mat_  [protected]
```

Definition at line 39 of file image_provider_base.h.

4.9.4.2 intrinsic_mat_

```
cv::Mat ImageProvider::intrinsic_mat_  [protected]
```

Definition at line 38 of file image_provider_base.h.

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

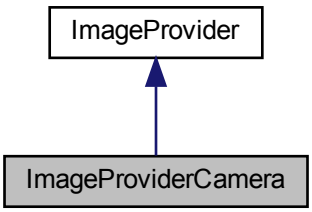
- modules/image-provider-base/ **image_provider_base.h**

4.10 ImageProviderCamera Class Reference

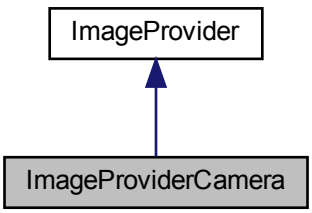
Camera (p. 10) image provider class implementation.

```
#include <image_provider_camera.h>
```

Inheritance diagram for ImageProviderCamera:



Collaboration diagram for ImageProviderCamera:



Public Member Functions

- **ImageProviderCamera** ()
- ~**ImageProviderCamera** () final
- bool **Initialize** (const std::string &) final

  Initialize by specified configuration file.
- bool **GetFrame** (**Frame** &frame) final

  Get a frame.

Additional Inherited Members

4.10.1 Detailed Description

Camera (p. 10) image provider class implementation.

Warning

NEVER directly use this class to create image provider!
Instead, turn to **ImageProviderFactory** (p. 48) class and use IPF_CREATE_IMAGE_PROVIDER("IP↩ Camera").

Definition at line 12 of file image_provider_camera.h.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 ImageProviderCamera()

```
ImageProviderCamera::ImageProviderCamera ( )  [inline]
```

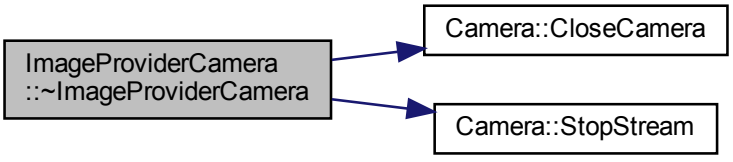
Definition at line 14 of file image_provider_camera.h.

4.10.2.2 ~ImageProviderCamera()

```
ImageProviderCamera::~ImageProviderCamera ( )  [final]
```

Definition at line 12 of file image_provider_camera.cpp.

Here is the call graph for this function:



4.10.3 Member Function Documentation

4.10.3.1 GetFrame()

```
bool ImageProviderCamera::GetFrame (
              Frame & frame )  [inline], [final], [virtual]
```

Get a frame.

Parameters

out	*frame*	OpenCV image reference.

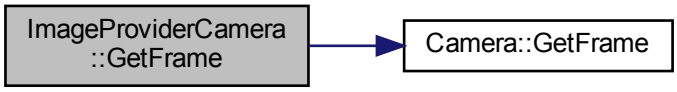
Returns

A boolean shows if frame is complete.

Implements **ImageProvider** (p. 43).

Definition at line 20 of file image_provider_camera.h.

Here is the call graph for this function:



4.10.3.2 Initialize()

```
bool ImageProviderCamera::Initialize (
             const std::string & file_path )  [final], [virtual]
```

Initialize by specified configuration file.

Parameters

in	*file_path*	Configuration file path.

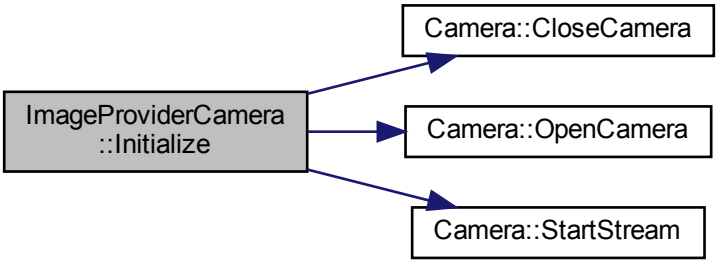
Returns

A boolean shows if initialization succeeded.

Implements **ImageProvider** (p. 44).

Definition at line 22 of file image_provider_camera.cpp.

Here is the call graph for this function:



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

- modules/image-provider-camera/ **image_provider_camera.h**
- modules/image-provider-camera/ **image_provider_camera.cpp**

4.11 ImageProviderFactory Class Reference

Singleton image provider factory.

`#include <image_provider_factory.h>`

Public Member Functions

- **ImageProviderFactory** (const **ImageProviderFactory** &)=delete
- **ImageProviderFactory** & **operator=** (const **ImageProviderFactory** &)=delete
- void **RegisterImageProvider** (const std::string &ip_type_name, **ImageProviderRegistryBase** *registry)
  Register an image provider type.
- **ImageProvider** * **CreateImageProvider** (const std::string &ip_type_name)
  Create an image provider whose type is registered to factory.

Static Public Member Functions

- static **ImageProviderFactory** & **Instance** ()
  Get the only instance of image provider factory.

4.11.1 Detailed Description

Singleton image provider factory.

Use **ImageProviderFactory::Instance()** (p. 50) to get the only instance pointer.
Use macro **IPF_CREATE_IMAGE_PROVIDER(ip_type_name)** (p. 70) to create an image provider instance.
Use ImageProviderFactory::Instance::RegisterImageProvider(ip_type_name, ∗registry) to register a type of image provider.

Warning

Image provider factory will not check whether IPType is really subclass of **ImageProvider** (p. 42) base class. (Thus, you should ensure that all callings of **ImageProviderRegistry** (p. 51) constructor are completely under control.)

Definition at line 42 of file image_provider_factory.h.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 ImageProviderFactory()

```
ImageProviderFactory::ImageProviderFactory (
            const  ImageProviderFactory &  )  [delete]
```

4.11.3 Member Function Documentation

4.11.3.1 CreateImageProvider()

```
ImageProvider* ImageProviderFactory::CreateImageProvider (
            const std::string & ip_type_name )  [inline]
```

Create an image provider whose type is registered to factory.

Parameters

in	*ip_type_name*	Type name of image provider.

Returns

A pointer to crated image provider.

Note

You may use macro **IPF_CREATE_IMAGE_PROVIDER(ip_type_name)** (p. 70) instead of call this function.

Definition at line 74 of file image_provider_factory.h.

4.11.3.2 Instance()

```
static ImageProviderFactory& ImageProviderFactory::Instance ( ) [inline], [static]
```

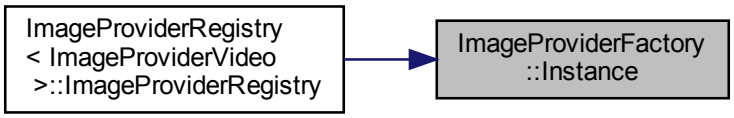
Get the only instance of image provider factory.

Returns

An image provider factory object.

Definition at line 52 of file image_provider_factory.h.

Here is the caller graph for this function:



4.11.3.3 operator=()

```
ImageProviderFactory& ImageProviderFactory::operator= (
            const ImageProviderFactory & ) [delete]
```

4.11.3.4 RegisterImageProvider()

```
void ImageProviderFactory::RegisterImageProvider (
            const std::string & ip_type_name,
             ImageProviderRegistryBase * registry ) [inline]
```

Register an image provider type.

Parameters

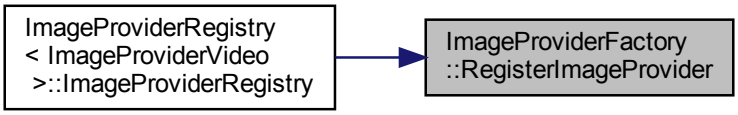
in	*ip_type_name*	Type name of image provider.
in	*registry*	A registry object of image provider.

Warning

You may call this function only when you're programming for a new type of image provider.

Definition at line 63 of file image_provider_factory.h.

Here is the caller graph for this function:



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

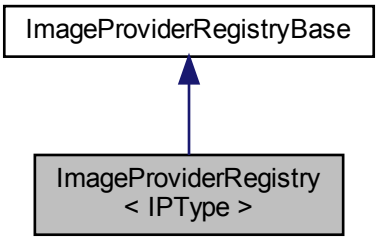
- modules/image-provider-base/ **image_provider_factory.h**

4.12 ImageProviderRegistry< IPType > Class Template Reference

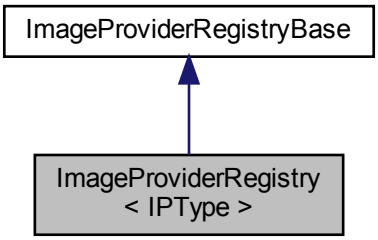
Templated image provider registry class.

`#include <image_provider_factory.h>`

Inheritance diagram for ImageProviderRegistry< IPType >:



Collaboration diagram for ImageProviderRegistry< IPType >:



Public Member Functions

- **ImageProviderRegistry** (const std::string &ip_type_name)

  Constructor of image provider registry.
- **ImageProvider** * **CreateImageProvider** () final

  Create an image provider of this type.

Additional Inherited Members

4.12.1 Detailed Description

template<class IPType>
class ImageProviderRegistry< IPType >

Templated image provider registry class.

Template Parameters

IPType	Image provider type inherited from base class **ImageProvider** (p. 42).

Attention

Once object is constructed, this type of image provider will immediately be registered to image provider factory.
This means the constructed object is useless and should not appear in any other place.
(Thus, template class though this is, it's better to be treated as a function.)

Warning

Image provider factory will not check whether IPType is really subclass of **ImageProvider** (p. 42) base class. (Thus, you should ensure that all callings of **ImageProviderRegistry** (p. 51) constructor are completely under control.)

Definition at line 104 of file image_provider_factory.h.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 ImageProviderRegistry()

```
template<class IPType >
ImageProviderRegistry< IPType >::  ImageProviderRegistry (
            const std::string & ip_type_name )  [inline], [explicit]
```

Constructor of image provider registry.

Parameters

in	*ip_type_name*	Type name of image provider.

Definition at line 110 of file image_provider_factory.h.

4.12.3 Member Function Documentation

4.12.3.1 CreateImageProvider()

```
template<class IPType >
ImageProvider* ImageProviderRegistry< IPType >::CreateImageProvider ( )  [inline], [final],
[virtual]
```

Create an image provider of this type.

Returns

An image provider pointer.

Warning

NEVER directly call this function. Instead, it should be called by image provider factory.

Implements **ImageProviderRegistryBase** (p. 55).

Definition at line 119 of file image_provider_factory.h.

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

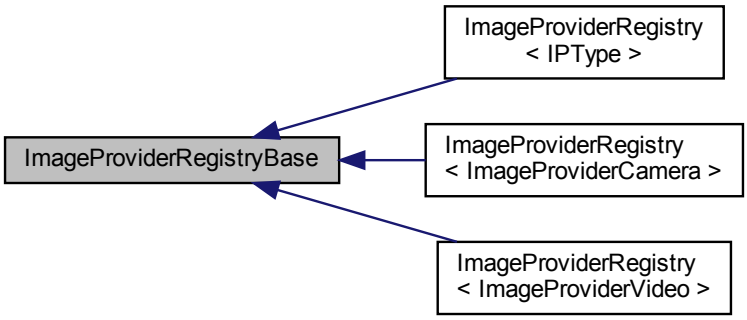
- modules/image-provider-base/ **image_provider_factory.h**

4.13 ImageProviderRegistryBase Class Reference

Base class of image provider registry.

`#include <image_provider_factory.h>`

Inheritance diagram for ImageProviderRegistryBase:



Public Member Functions

- virtual **ImageProvider** * **CreateImageProvider** ()=0
- **ImageProviderRegistryBase** (const **ImageProviderRegistryBase** &)=delete
- **ImageProviderRegistryBase** & **operator=** (const **ImageProviderRegistryBase** &)=delete

Protected Member Functions

- **ImageProviderRegistryBase** ()=default
- virtual ~**ImageProviderRegistryBase** ()=default

4.13.1 Detailed Description

Base class of image provider registry.

Warning

You should use its subclass **ImageProviderRegistry** (p. 51) instead of this base class.

Definition at line 17 of file image_provider_factory.h.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 ImageProviderRegistryBase() [1/2]

```
ImageProviderRegistryBase::ImageProviderRegistryBase (
            const  ImageProviderRegistryBase &  )  [delete]
```

4.13.2.2 ImageProviderRegistryBase() [2/2]

```
ImageProviderRegistryBase::ImageProviderRegistryBase ( )  [protected], [default]
```

4.13.2.3 ~ImageProviderRegistryBase()

```
virtual ImageProviderRegistryBase::~ImageProviderRegistryBase ( )  [protected], [virtual],
[default]
```

4.13.3 Member Function Documentation

4.13.3.1 CreateImageProvider()

```
virtual  ImageProvider* ImageProviderRegistryBase::CreateImageProvider ( )  [pure virtual]
```

Implemented in **ImageProviderRegistry< IPType >** (p. 53), **ImageProviderRegistry< ImageProviderCamera >** (p. 53), and **ImageProviderRegistry< ImageProviderVideo >** (p. 53).

4.13.3.2 operator=()

```
ImageProviderRegistryBase& ImageProviderRegistryBase::operator= (
            const  ImageProviderRegistryBase &  )  [delete]
```

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

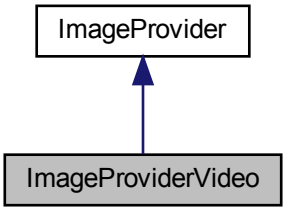
- modules/image-provider-base/ **image_provider_factory.h**

4.14 ImageProviderVideo Class Reference

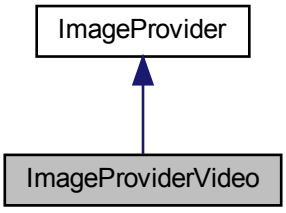
Video image provider class implementation.

`#include <image_provider_video.h>`

Inheritance diagram for ImageProviderVideo:



Collaboration diagram for ImageProviderVideo:



Public Member Functions

- **ImageProviderVideo** ()=default
- ~**ImageProviderVideo** () final
- bool **Initialize** (const std::string &) final

  Initialize by specified configuration file.
- bool **GetFrame** (**Frame** &frame) final

  Get a frame.

Additional Inherited Members

4.14.1 Detailed Description

Video image provider class implementation.

Warning

NEVER directly use this class to create image provider!
Instead, turn to **ImageProviderFactory** (p. 48) class and use IPF_CREATE_IMAGE_PROVIDER("IPVideo").

Definition at line 13 of file image_provider_video.h.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 ImageProviderVideo()

```
ImageProviderVideo::ImageProviderVideo ( )  [default]
```

4.14.2.2 ~ImageProviderVideo()

```
ImageProviderVideo::~ImageProviderVideo ( )  [final]
```

Definition at line 11 of file image_provider_video.cpp.

4.14.3 Member Function Documentation

4.14.3.1 GetFrame()

```
bool ImageProviderVideo::GetFrame (
            Frame & frame )  [inline], [final], [virtual]
```

Get a frame.

Parameters

out	*frame*	OpenCV image reference.

Returns

A boolean shows if frame is complete.

Implements **ImageProvider** (p. 43).

Definition at line 21 of file image_provider_video.h.

4.14.3.2 Initialize()

```
bool ImageProviderVideo::Initialize (
            const std::string & file_path )  [final], [virtual]
```

Initialize by specified configuration file.

Parameters

in	*file_path*	Configuration file path.

Returns

A boolean shows if initialization succeeded.

Implements **ImageProvider** (p. 44).

Definition at line 19 of file image_provider_video.cpp.

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

- modules/image-provider-video/ **image_provider_video.h**
- modules/image-provider-video/ **image_provider_video.cpp**

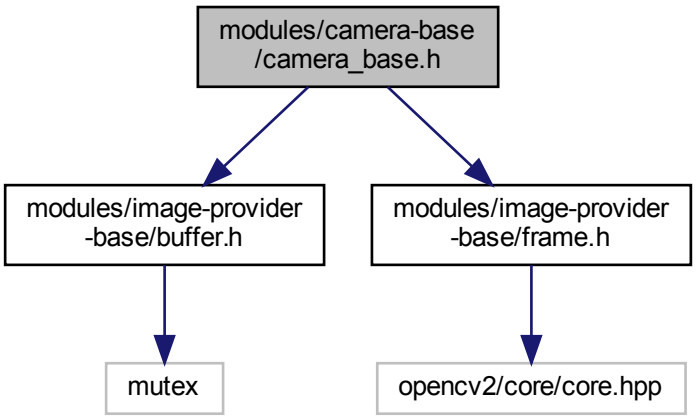
Chapter 5

File Documentation

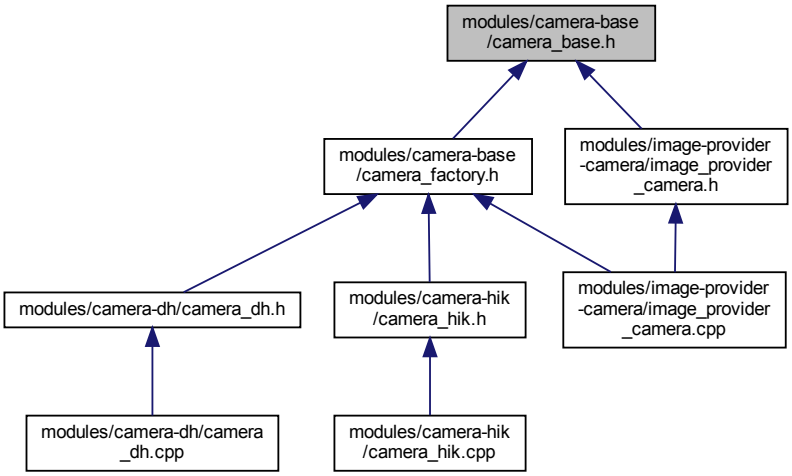
5.1 modules/camera-base/camera_base.h File Reference

```
#include "modules/image-provider-base/buffer.h"
#include "modules/image-provider-base/frame.h"
```

Include dependency graph for camera_base.h:



This graph shows which files directly or indirectly include this file:



Classes

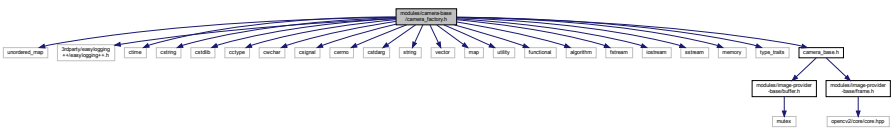
- class **Camera**

  Camera *(*p. *10) base class.*

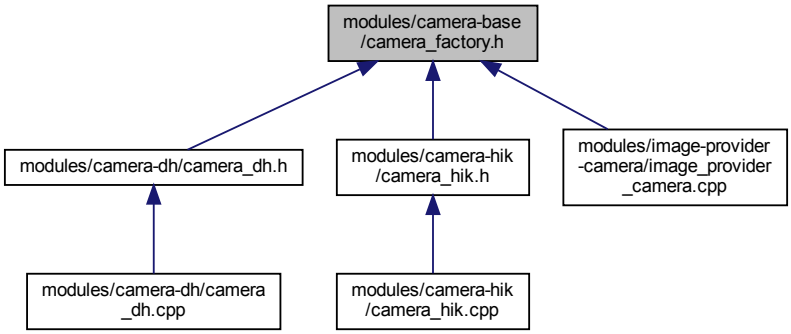
5.2 modules/camera-base/camera_factory.h File Reference

```
#include <unordered_map>
#include "3rdparty/easylogging++/easylogging++.h"
#include "camera_base.h"
```

Include dependency graph for camera_factory.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **CameraRegistryBase**

  Base class of camera registry.
- class **CameraFactory**

  Singleton camera factory.
- class **CameraRegistry**< **CameraType** >

  Templated camera registry class.

Macros

- #define **CF_CREATE_CAMERA**(camera_type_name) **CameraFactory::Instance**().CreateCamera(camera↩ _type_name)

  *A macro to create a camera of specified type name. For details, turn to class **CameraFactory** (p. 18).*

5.2.1 Macro Definition Documentation

5.2.1.1 CF_CREATE_CAMERA

```
#define CF_CREATE_CAMERA(
            camera_type_name )   CameraFactory::Instance().CreateCamera(camera_type_name)
```

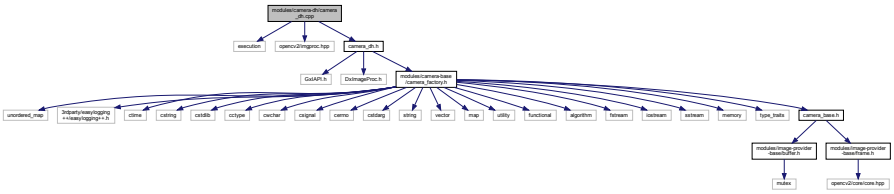
A macro to create a camera of specified type name. For details, turn to class **CameraFactory** (p. 18).

Definition at line 11 of file camera_factory.h.

5.3 modules/camera-dh/camera_dh.cpp File Reference

```
#include <execution>
#include <opencv2/imgproc.hpp>
#include "camera_dh.h"
```

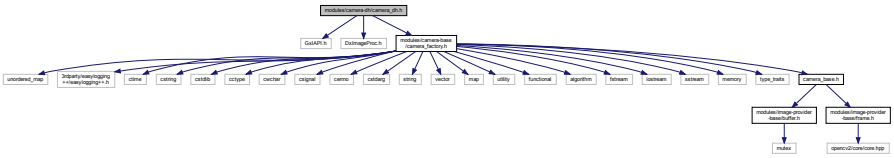
Include dependency graph for camera_dh.cpp:



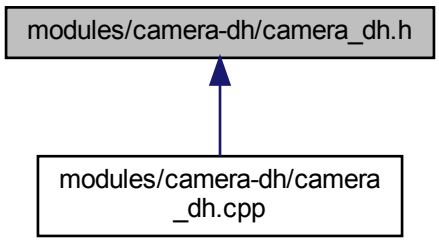
5.4 modules/camera-dh/camera_dh.h File Reference

```
#include "GxIAPI.h"
#include "DxImageProc.h"
#include "modules/camera-base/camera_factory.h"
```

Include dependency graph for camera_dh.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **DHCamera**

  DaHeng camera class implementation.

Macros

- #define **GX_OPEN_CAMERA_CHECK_STATUS**(status_code)

  This macro is used to check if the device is successfully initialized.
- #define **GX_CHECK_STATUS**(status_code)

  This macro is used to check if parameters are successfully modified or set.
- #define **GX_START_STOP_STREAM_CHECK_STATUS**(status_code)

  This macro is used to check if the stream is successfully opened or closed.

5.4.1 Macro Definition Documentation

5.4.1.1 GX_CHECK_STATUS

```
#define GX_CHECK_STATUS(
               status_code )
```

Value:

```
    if ((status_code) != GX_STATUS_SUCCESS) {  \
    LOG(ERROR) « GetErrorInfo(status_code);   \
    return false;                              \
}
```

This macro is used to check if parameters are successfully modified or set.

Attention

!! DO NOT use this macro in other place !!

Definition at line 33 of file camera_dh.h.

5.4.1.2 GX_OPEN_CAMERA_CHECK_STATUS

```
#define GX_OPEN_CAMERA_CHECK_STATUS(
               status_code )
```

Value:

```
    if ((status_code) != GX_STATUS_SUCCESS) {             \
        LOG(ERROR) « GetErrorInfo(status_code);          \
        (status_code) = GXCloseDevice(device_);           \
        if ((status_code) != GX_STATUS_SUCCESS)           \
            LOG(ERROR) « GetErrorInfo(status_code);      \
        device_ = nullptr;                                \
        serial_number_ = "";                              \
        if (!camera_count_) {                             \
            (status_code) = GXCloseLib();                 \
            if ((status_code) != GX_STATUS_SUCCESS)       \
                LOG(ERROR) « GetErrorInfo(status_code);  \
        }                                                 \
    return false;                                         \
}
```

This macro is used to check if the device is successfully initialized.

Attention

!! DO NOT use this macro in other place !!

Definition at line 13 of file camera_dh.h.

5.4.1.3 GX_START_STOP_STREAM_CHECK_STATUS

```
#define GX_START_STOP_STREAM_CHECK_STATUS(
              status_code )
```

Value:

```
if ((status_code) != GX_STATUS_SUCCESS) {                \
    if (raw_16_to_8_cache_ != nullptr) {                 \
        delete[] raw_16_to_8_cache_;                     \
        raw_16_to_8_cache_ = nullptr;                    \
    }                                                    \
    if (raw_8_to_rgb_24_cache_ != nullptr) {             \
        delete[] raw_8_to_rgb_24_cache_;                 \
        raw_8_to_rgb_24_cache_ = nullptr;                \
    }                                                    \
    LOG(ERROR) « GetErrorInfo(status_code);             \
    return false;                                        \
}
```

This macro is used to check if the stream is successfully opened or closed.

Attention

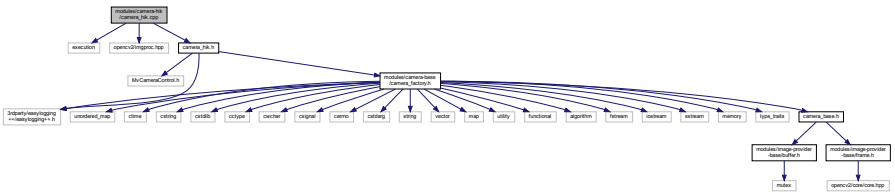
!! DO NOT use this macro in other place !!

Definition at line 43 of file camera_dh.h.

5.5 modules/camera-hik/camera_hik.cpp File Reference

```
#include <execution>
#include <opencv2/imgproc.hpp>
#include "camera_hik.h"
```

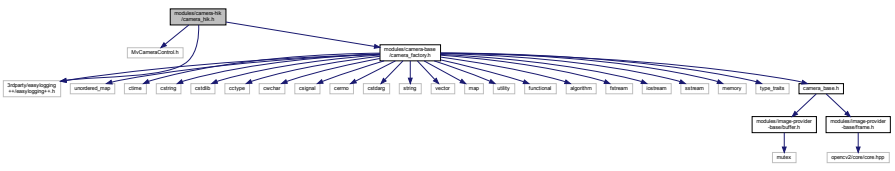
Include dependency graph for camera_hik.cpp:



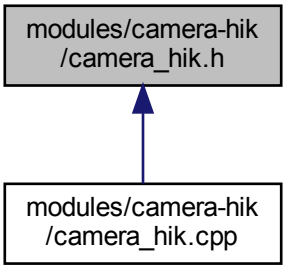
5.6 modules/camera-hik/camera_hik.h File Reference

```
#include "MvCameraControl.h"
#include "3rdparty/easylogging++/easylogging++.h"
#include "modules/camera-base/camera_factory.h"
```

Include dependency graph for camera_hik.h:



This graph shows which files directly or indirectly include this file:



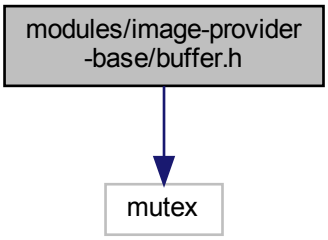
Classes

- class **HikCamera**

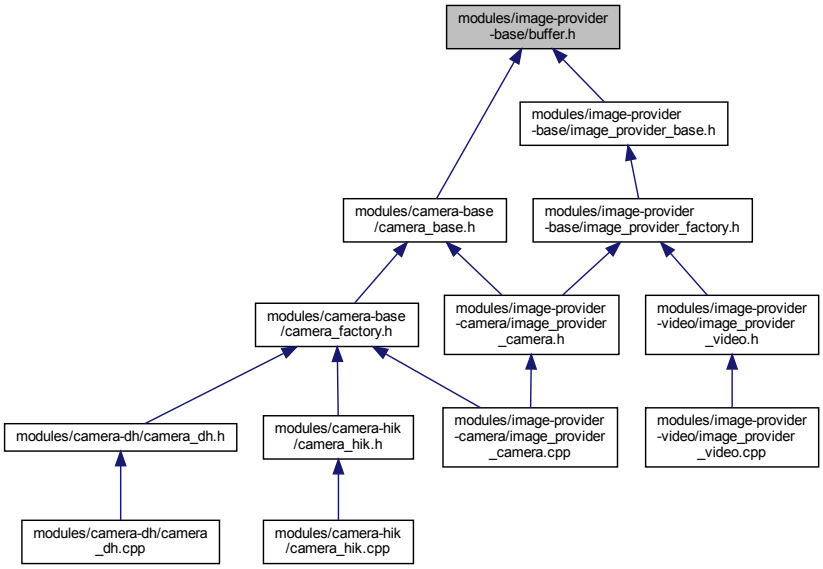
  HikRobot camera class implementation.

5.7 modules/image-provider-base/buffer.h File Reference

`#include <mutex>`
Include dependency graph for buffer.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **Buffer**< **Type, size** >

  Ring buffer with mutex.

Macros

- #define **IP_BUFFER_SIZE** 4

  Buffer *(p. 7) size for image provider and camera.*

5.7.1 Macro Definition Documentation

5.7.1.1 IP_BUFFER_SIZE

```
#define IP_BUFFER_SIZE 4
```

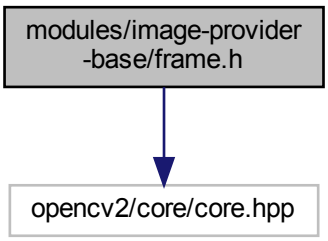
Buffer (p. 7) size for image provider and camera.

Definition at line 7 of file buffer.h.

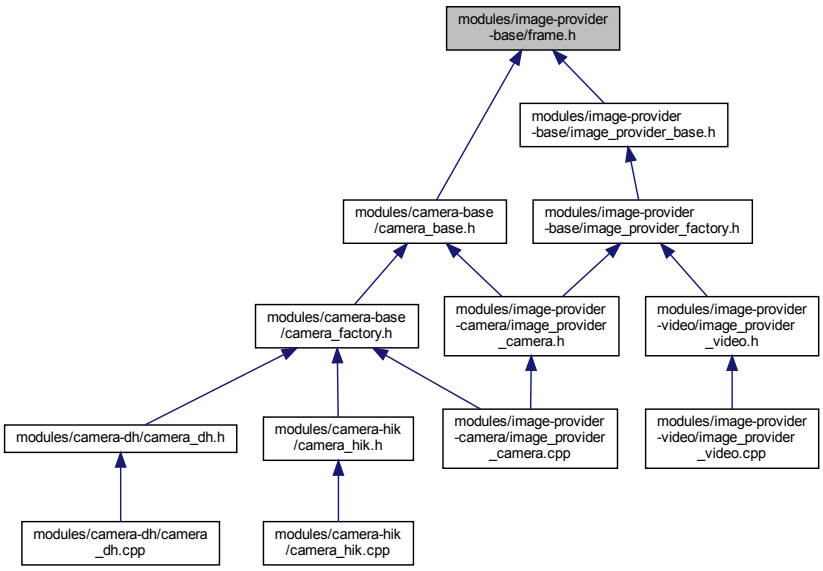
5.8 modules/image-provider-base/frame.h File Reference

```
#include <opencv2/core/core.hpp>
```

Include dependency graph for frame.h:



This graph shows which files directly or indirectly include this file:



Classes

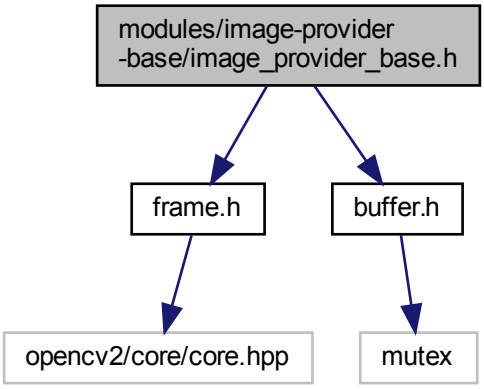
- struct **Frame**

  Single frame structure.

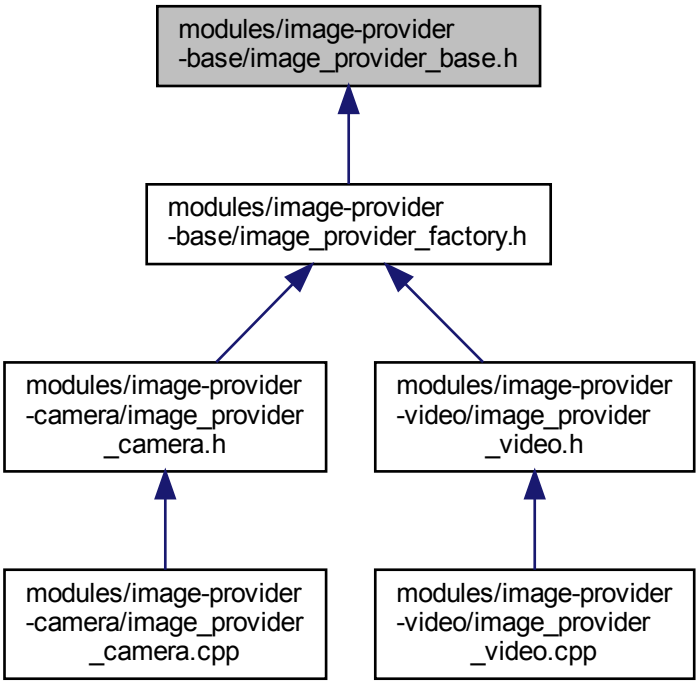
5.9 modules/image-provider-base/image_provider_base.h File Reference

```
#include "frame.h"
#include "buffer.h"
```

Include dependency graph for image_provider_base.h:



This graph shows which files directly or indirectly include this file:



Classes

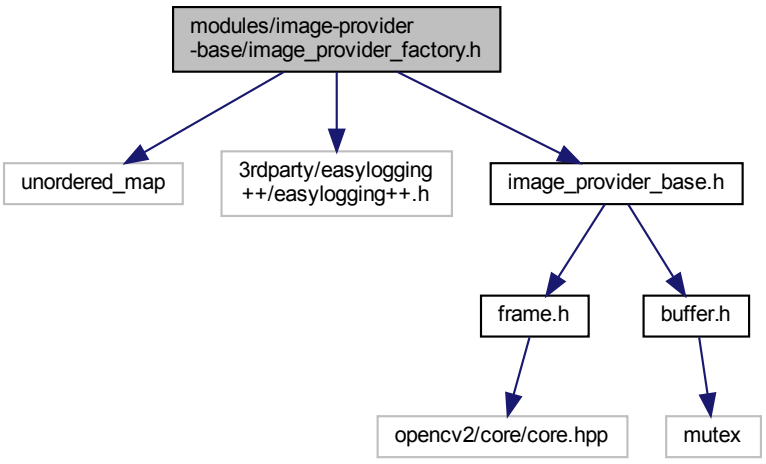
- class **ImageProvider**

  Image provider base class.

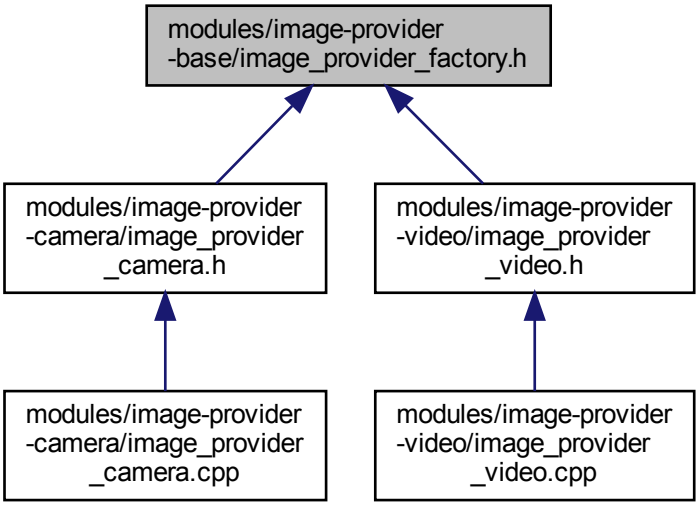
5.10 modules/image-provider-base/image_provider_factory.h File Reference

```
#include <unordered_map>
#include "3rdparty/easylogging++/easylogging++.h"
#include "image_provider_base.h"
```

Include dependency graph for image_provider_factory.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **ImageProviderRegistryBase**
  Base class of image provider registry.
- class **ImageProviderFactory**
  Singleton image provider factory.
- class **ImageProviderRegistry**< **IPType** >
  Templated image provider registry class.

Macros

- #define **IPF_CREATE_IMAGE_PROVIDER**(ip_type_name) **ImageProviderFactory::Instance**().Create↩ ImageProvider(ip_type_name)
  *A macro to create an image provider of specified type name. For details, turn to class **ImageProviderFactory** (p. 48).*

5.10.1 Macro Definition Documentation

5.10.1.1 IPF_CREATE_IMAGE_PROVIDER

```
#define IPF_CREATE_IMAGE_PROVIDER(
            ip_type_name )  ImageProviderFactory::Instance().CreateImageProvider(ip_type_↩
name)
```

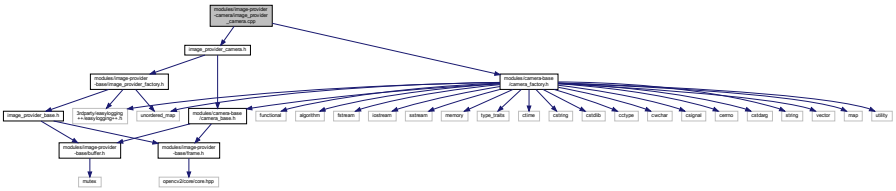
A macro to create an image provider of specified type name. For details, turn to class **ImageProviderFactory** (p. 48).

Definition at line 11 of file image_provider_factory.h.

5.11 modules/image-provider-camera/image_provider_camera.cpp File Reference

```
#include "image_provider_camera.h"
#include "modules/camera-base/camera_factory.h"
```

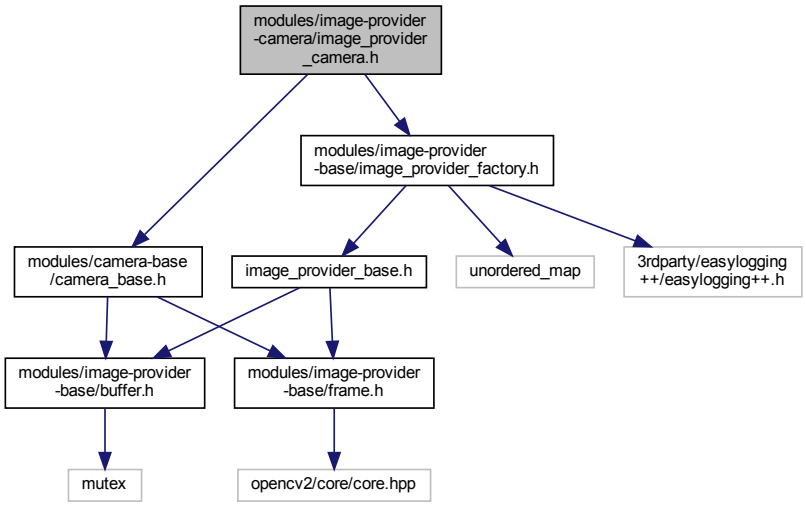
Include dependency graph for image_provider_camera.cpp:



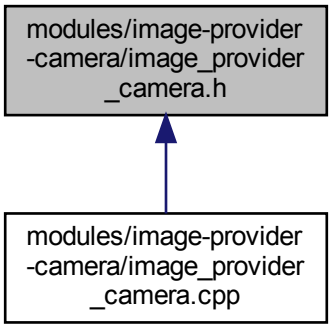
5.12 modules/image-provider-camera/image_provider_camera.h File Reference

```
#include "modules/camera-base/camera_base.h"
#include "modules/image-provider-base/image_provider_factory.h"
```

Include dependency graph for image_provider_camera.h:



This graph shows which files directly or indirectly include this file:



Classes

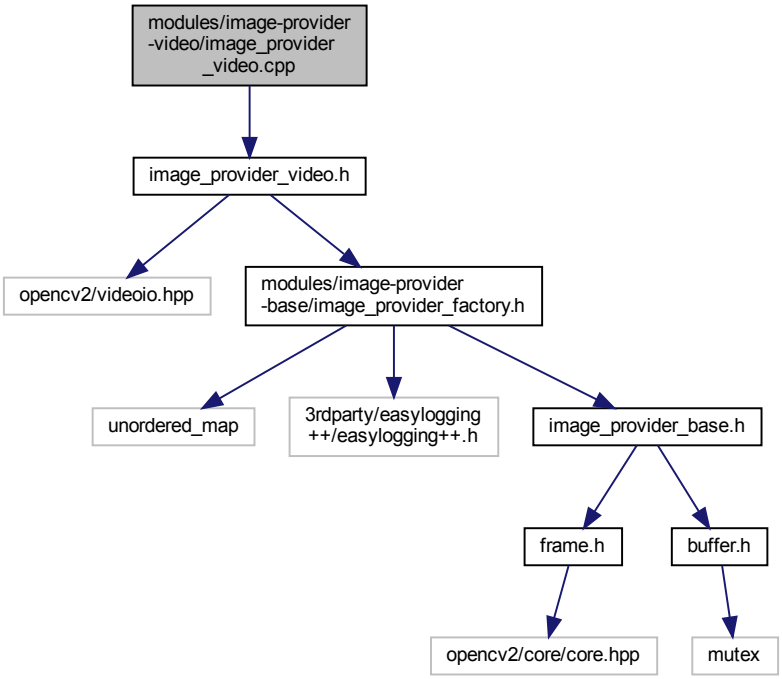
- class **ImageProviderCamera**

  Camera *(*p. *10) image provider class implementation.*

5.13 modules/image-provider-video/image_provider_video.cpp File Reference

```
#include "image_provider_video.h"
```

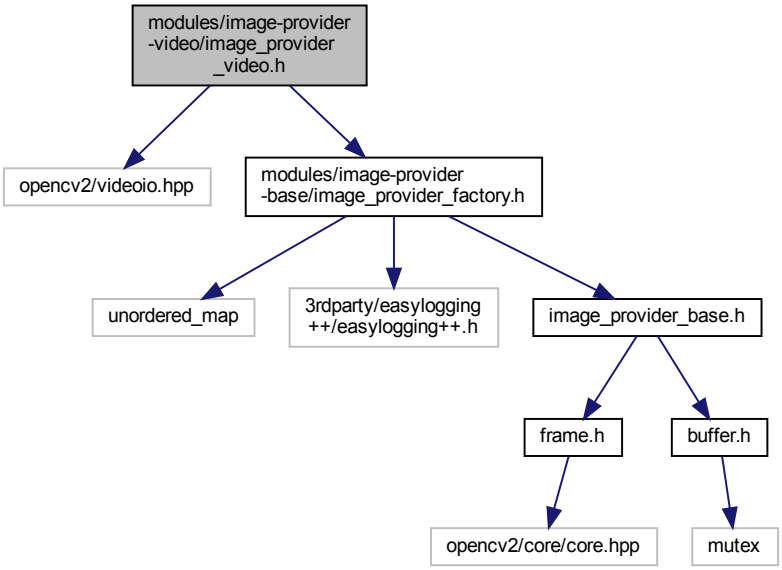
Include dependency graph for image_provider_video.cpp:



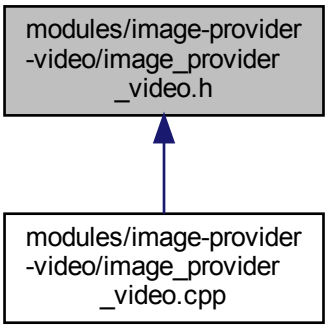
5.14 modules/image-provider-video/image_provider_video.h File Reference

```
#include <opencv2/videoio.hpp>
#include "modules/image-provider-base/image_provider_factory.h"
```

Include dependency graph for image_provider_video.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **ImageProviderVideo**

  Video image provider class implementation.

Index