ifm3d

Generated by Doxygen 1.9.2

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

Class Documentation

3.1 ifm3d::ByteBuffer< Derived > Class Template Reference

`#include <byte_buffer.h>`

Public Types

- using **Ptr** = std::shared_ptr< ByteBuffer< Derived > >

Public Member Functions

- ByteBuffer ()
- virtual ~ByteBuffer ()
- **ByteBuffer** (ByteBuffer &&)
- ByteBuffer & **operator=** (ByteBuffer &&)
- **ByteBuffer** (const ByteBuffer &src_buff)
- ByteBuffer & **operator=** (const ByteBuffer &src_buff)
- std::vector< std::uint8_t > Bytes ()
- bool Dirty () const noexcept
- void SetBytes (std::vector< std::uint8_t > &buff, bool copy=false)
- std::vector< float > Extrinsics ()
- std::vector< float > Intrinsics ()
- std::vector< float > InverseIntrinsics ()
- std::vector< std::uint32_t > ExposureTimes ()
- ifm3d::TimePointT TimeStamp ()
- std::vector< ifm3d::TimePointT > TimeStamps ()
- float IlluTemp ()
- std::string JSONModel ()
- void Organize ()

Protected Member Functions

- template<typename T >
  void ImCreate (ifm3d::image_chunk im, std::uint32_t fmt, std::size_t idx, std::uint32_t width, std::uint32_t height, int nchan, std::uint32_t npts, const std::vector< std::uint8_t > &bytes)
- template<typename T >
  void CloudCreate (std::uint32_t fmt, std::size_t xidx, std::size_t yidx, std::size_t zidx, std::uint32_t width, std::uint32_t height, std::uint32_t npts, const std::vector< std::uint8_t > &bytes)
- void _SetDirty (bool flg) noexcept

Protected Attributes

- bool dirty_
- std::vector< std::uint8_t > bytes_
- std::vector< float > extrinsics_
- std::vector< float > intrinsics_
- std::vector< float > inverseIntrinsics_
- std::vector< std::uint32_t > exposure_times_
- std::vector< ifm3d::TimePointT > time_stamps_
- float illu_temp_
- std::string json_model_

3.1.1 Detailed Description

template<typename Derived>
class ifm3d::ByteBuffer< Derived >

The ByteBuffer class is used to hold a validated byte buffer from the sensor that represents a single time-synchronized set of images based on the current schema mask set on the active framegrabber.

The ByteBuffer imposes no specific image or point cloud data structure. This class is intended to be subclassed where more user-friendly data structures can be used to gain access to the bytes in a semantically meaningful manner.

There are two primary interfaces (documented below) that image container developers should implement. They are:

`ImCreate` `CloudCreate`

These functions will be called as customization hooks at the time of parsing the raw image bytes from the sensor. It is the contract of this interface that all calls to `ImCreate` will be made before the single call to `CloudCreate`. The reason for this part of the contract is to give all image container implementers the ability to set the point cloud intensity data from one of the image containers. That is, at the time of calling `CloudCreate` all image data (e.g., amplitude or gray) data are avaialble for coloring the point cloud intensity pixels. In addition, it is guaranteed that the first call to `ImCreate` will be the confidence image. So, the confidence bits for a given pixel can be consulting while constructing the payload images.

We note that the ploymorphic behaviors implemented by this class are static (resolved at compile-time via CRTP) rather than dynamic at runtime (via a vtable). This is why there are no interface functions declared `virtual`.

NOTE: The ByteBuffer is NOT thread safe!

3.1.2 Constructor & Destructor Documentation

3.1.2.1 ByteBuffer()

```
template<typename Derived >
ifm3d::ByteBuffer< Derived >::ByteBuffer ( )
```

Default initializes instance vars

3.1.2.2 ~ByteBuffer()

```
template<typename Derived >
virtual ifm3d::ByteBuffer< Derived >::~ByteBuffer ( )  [virtual]
```

RAII dealloc

3.1.3 Member Function Documentation

3.1.3.1 _SetDirty()

```
template<typename Derived >
void ifm3d::ByteBuffer< Derived >::_SetDirty (
            bool flg )  [protected], [noexcept]
```

Mutates the dirty flag

3.1.3.2 Bytes()

```
template<typename Derived >
std::vector<std::uint8_t> ifm3d::ByteBuffer< Derived >::Bytes ( )
```

Returns a copy of the underlying byte buffer read from the camera

3.1.3.3 CloudCreate()

```
template<typename Derived >
template<typename T >
void ifm3d::ByteBuffer< Derived >::CloudCreate (
            std::uint32_t fmt,
            std::size_t xidx,
            std::size_t yidx,
            std::size_t zidx,
            std::uint32_t width,
            std::uint32_t height,
            std::uint32_t npts,
            const std::vector< std::uint8_t > & bytes )  [inline], [protected]
```

This function is part of the ByteBuffer interface, intended to be overloaded by image container implementers. It is a callback hook that is called once for each frame recieved by the framegrabber if the cartesian data are specified in the current pcic schema. All 2D image callbacks (i.e., `ImCreate`) are guaranteed to be called prior to this function – this is to allow for "coloring" the intensity channel of the point cloud with data from one of the 2D images (already parsed). It is also implied that for a given frame, before this function is called, the image container implementer will have had the opportunity to construct the confidence image associated with this frame.

Parameters

in	*fmt*	The pixel format of the image (see `ifm3d::pixel_format`)
in	*xidx*	The index into `bytes` where the x-coords start
in	*yidx*	The index into `bytes` where the y-coords start
in	*zidx*	The index into `bytes` where the z-coords start
in	*width*	Number of columns in the point cloud
in	*height*	Number of rows in the point cloud

3.1.3.4 Dirty()

```
template<typename Derived >
bool ifm3d::ByteBuffer< Derived >::Dirty ( ) const  [noexcept]
```

Returns the state of the 'dirty' flag

3.1.3.5 ExposureTimes()

```
template<typename Derived >
std::vector<std::uint32_t> ifm3d::ByteBuffer< Derived >::ExposureTimes ( )
```

Returns a 3-element vector containing the exposure times (usec) for the current frame. Unused exposure times are reported as 0.

If all elements are reported as 0 either the exposure times are not configured to be returned back in the data stream from the camera or an error in parsing them has occured.

3.1.3.6 Extrinsics()

```
template<typename Derived >
std::vector<float> ifm3d::ByteBuffer< Derived >::Extrinsics ( )
```

Returns a 6-element vector containing the extrinsic calibration of the camera. NOTE: This is the extrinsics WRT to the ifm optical frame.

The elements are: tx, ty, tz, rot_x, rot_y, rot_z

Translation units are mm, rotations are degrees

Users of this library are highly DISCOURAGED from using the extrinsic calibration data stored on the camera itself.

3.1.3.7 IlluTemp()

```
template<typename Derived >
float ifm3d::ByteBuffer< Derived >::IlluTemp ( )
```

Returns the temperature of the illumination unit.

NOTE: To get the temperature of the illumination unit to the frame, you need to make sure your current pcic schema asks for it.

3.1.3.8 ImCreate()

```
template<typename Derived >
template<typename T >
void ifm3d::ByteBuffer< Derived >::ImCreate (
            ifm3d::image_chunk im,
            std::uint32_t fmt,
            std::size_t idx,
            std::uint32_t width,
            std::uint32_t height,
            int nchan,
            std::uint32_t npts,
            const std::vector< std::uint8_t > & bytes )  [inline], [protected]
```

This function is part of the ByteBuffer interface, intended to be overloaded by image container implementers. It is a callback hook that is called once for each 2D image type specified in the current pcic schema for each frame recieved by the framegrabber. All 2D image callbacks are guaranteed to be called prior to the `CloudCreate` callback – this is to allow for "coloring" the intensity channel of the point cloud with data from one of the 2D images (already parsed)

For a given frame, the first `ImCreate` callback will be the confidence image.

Parameters

in	*im*	The 2D image type currently being processed
in	*fmt*	The pixel format of the image (see `ifm3d::pixel_format`)
in	*idx*	The index into the byte buffer, `bytes`, as to where the pixel data begin
in	*width*	The image width (pixels)
in	*height*	The image height (pixels)
in	*nchan*	The number of channels in the image
in	*npts*	The total number of image points (width ∗ height)
in	*bytes*	A const reference to the byte buffer to process

3.1.3.9 Intrinsics()

```
template<typename Derived >
std::vector<float> ifm3d::ByteBuffer< Derived >::Intrinsics ( )
```

Returns a 16-element vector containing the intrinsic calibration of the camera.

The elements are:

Name Data type Unit Description fx 32 bit float px Focal length of the camera in the sensor's x axis direction. fy 32 bit float px Focal length of the camera in the sensor's yaxis direction. mx 32 bit float px Main point in the sensor's x direction my 32 bit float px Main point in the sensor's y direction alpha 32 bit float dimensionless Skew parameter k1 32 bit float dimensionless First radial distortion coefficient k2 32 bit float dimensionless Second radial distortion coefficient k5 32 bit float dimensionless Third radial distortion coefficient k3 32 bit float dimensionless First tangential distortion coefficient k4 32 bit float dimensionless Second tangential distortion coefficient transX 32 bit float mm Translation along x-direction in meters. transY 32 bit float mm Translation along y-direction in meters. transZ 32 bit float mm Translation along z-direction in meters. rotX 32 bit float degree Rotation along x-axis in radians. Positive values indicate clockwise rotation. rotY 32 bit float degree Rotation along y-axis in radians. Positive values indicate clockwise rotation. rotZ 32 bit float degree Rotation along z-axis in radians. Positive values indicate clockwise rotation.

3.1.3.10 InverseIntrinsics()

```
template<typename Derived >
std::vector<float> ifm3d::ByteBuffer< Derived >::InverseIntrinsics ( )
```

Returns a 16-element vector containing the inverse intrinsic calibration of the camera. See Intrinsics() for further information

3.1.3.11 JSONModel()

```
template<typename Derived >
std::string ifm3d::ByteBuffer< Derived >::JSONModel ( )
```

Returns the JSON model of the output of the active application

NOTE: To get the JSON data for the application running on the device, you need to make sure your current pcic schema asks for it by including ifm3d::JSON_MODEL in the schema. This will return a blank JSON string ("{}") for Camera devices like the O3D303, versus ifm Smart Sensors like the O3D302.

3.1.3.12 Organize()

```
template<typename Derived >
void ifm3d::ByteBuffer< Derived >::Organize ( )
```

This is the interface hook that synchronizes the internally wrapped byte buffer with the semantically meaningful image/cloud data structures. Within the overall `ifm3d` framework, this function is called by the `FrameGrabber` when a complete "frame packet" has been recieved. This then parses the bytes and, in-line, will statically dispatch to the underly dervied class to populate their image/cloud data structures.

Additionally, this function will populate the extrinsics, exposure times, timestamp, and illumination temperature as appropriate and subject to the current pcic schema.

3.1.3.13 SetBytes()

```
template<typename Derived >
void ifm3d::ByteBuffer< Derived >::SetBytes (
            std::vector< std::uint8_t > & buff,
            bool copy = false )
```

Sets the data from the passed in 'buff' to the internally wrapped byte buffer. This function assumes the passed in 'buff' is valid.

By default, this function will take in `buff` and `swap` contents with its internal buffer so that the operation is O(1) and requires no data copies. If you want copy behavior, specify the `copy` flag and complexity will be linear in the size of the byte buffer which is driven by the schema mask currently applied to the running framegrabber.

Parameters

`in`	*buff*	Raw data bytes to copy/swap to internal buffers
`in`	*copy*	If true, the data are copied from `buff` to the internally wrapped buffer and `buff` will remain unchanged.

3.1.3.14 TimeStamp()

```
template<typename Derived >
ifm3d::TimePointT ifm3d::ByteBuffer< Derived >::TimeStamp ( )
```

Returns the time stamp of the image data.

NOTE: To get the timestamp of the confidence data, you need to make sure your current pcic schema mask have enabled confidence data.

3.1.3.15 TimeStamps()

```
template<typename Derived >
std::vector<ifm3d::TimePointT> ifm3d::ByteBuffer< Derived >::TimeStamps ( )
```

Returns the time stamps of the image data for O3X devices Value at index 0 will represent the time at which phase data is read. Value at index 1 will represent the time at which datais transferred over ethernet.

Returns the timestamps of phase data for O3R device

3.1.4 Member Data Documentation

3.1.4.1 bytes_

```
template<typename Derived >
std::vector<std::uint8_t> ifm3d::ByteBuffer< Derived >::bytes_  [protected]
```

Raw bytes read off the wire from the camera.

3.1.4.2 dirty_

```
template<typename Derived >
bool ifm3d::ByteBuffer< Derived >::dirty_  [protected]
```

Flag used to indicate if the wrapped byte buffer needs to be 'Organized'. I.e., in a subclass, this would indicate if your parsed out image data structures need to be synchronized to the underlying byte buffer or not.

3.1.4.3 exposure_times_

```
template<typename Derived >
std::vector<std::uint32_t> ifm3d::ByteBuffer< Derived >::exposure_times_  [protected]
```

Exposure time(s) (up to 3), registered to the current frame.

3.1.4.4 extrinsics_

```
template<typename Derived >
std::vector<float> ifm3d::ByteBuffer< Derived >::extrinsics_  [protected]
```

Extrinsic calibration WRT camera optical frame: tx, ty, tz, rotx, roty, rotz. Translation units are mm, rotational units are degrees.

3.1.4.5 illu_temp_

```
template<typename Derived >
float ifm3d::ByteBuffer< Derived >::illu_temp_  [protected]
```

Temperature of the illumination unit synchronized in time with the current frame data.

3.1.4.6 intrinsics_

```
template<typename Derived >
std::vector<float> ifm3d::ByteBuffer< Derived >::intrinsics_  [protected]
```

Intrinsic calibration WRT camera lense

3.1.4.7 inverseIntrinsics_

```
template<typename Derived >
std::vector<float> ifm3d::ByteBuffer< Derived >::inverseIntrinsics_  [protected]
```

Inverse intrinsic calibration WRT camera lense:

3.1.4.8 json_model_

```
template<typename Derived >
std::string ifm3d::ByteBuffer< Derived >::json_model_  [protected]
```

JSON string of the active application output

3.1.4.9 time_stamps_

```
template<typename Derived >
std::vector<ifm3d::TimePointT> ifm3d::ByteBuffer< Derived >::time_stamps_  [protected]
```

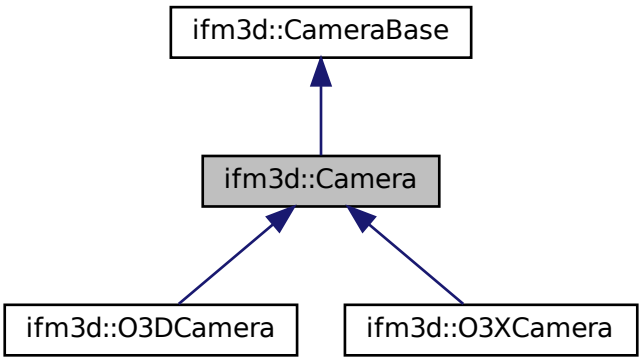
Camera timestamps of the current frame

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

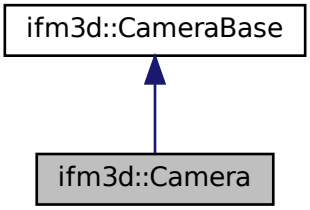
- /home/usmasslo/gitlab/ifm3d/modules/framegrabber/include/ifm3d/fg/byte_buffer.h

3.2 ifm3d::Camera Class Reference

Inheritance diagram for ifm3d::Camera:



Collaboration diagram for ifm3d::Camera:



Public Types

- using **Ptr** = std::shared_ptr< Camera >
- using **boot_mode** = ifm3d::CameraBase::boot_mode
- using **operating_mode** = ifm3d::CameraBase::operating_mode
- using **trigger_mode** = ifm3d::CameraBase::trigger_mode
- using **import_flags** = ifm3d::CameraBase::import_flags
- using **spatial_filter** = ifm3d::CameraBase::spatial_filter
- using **temporal_filter** = ifm3d::CameraBase::temporal_filter
- using **mfilt_mask_size** = ifm3d::CameraBase::mfilt_mask_size
- using **device_family** = ifm3d::CameraBase::device_family

Public Member Functions

- Camera (const std::string &ip=ifm3d::DEFAULT_IP, const std::uint16_t xmlrpc_port=ifm3d::DEFAULT_↩ XMLRPC_PORT, const std::string &password=ifm3d::DEFAULT_PASSWORD)
- virtual std::string Password ()
- virtual std::string SessionID ()
- virtual void FactoryReset ()
- virtual std::string RequestSession ()
- virtual bool CancelSession ()
- virtual bool CancelSession (const std::string &sid)
- virtual int Heartbeat (int hb)
- virtual std::unordered_map< std::string, std::string > **NetInfo** ()
- virtual std::unordered_map< std::string, std::string > **TimeInfo** ()
- virtual void SetTemporaryApplicationParameters (const std::unordered_map< std::string, std::string > &params)
- virtual int ActiveApplication ()
- virtual json ApplicationList ()
- virtual std::vector< std::string > ApplicationTypes ()
- virtual std::vector< std::string > ImagerTypes ()
- virtual int CopyApplication (int idx)
- virtual int CreateApplication (const std::string &type=DEFAULT_APPLICATION_TYPE)
- virtual void DeleteApplication (int idx)
- virtual void SetCurrentTime (int epoch_secs=-1)
- virtual std::vector< std::uint8_t > UnitVectors ()
- virtual std::vector< std::uint8_t > ExportIFMConfig ()
- virtual std::vector< std::uint8_t > ExportIFMApp (int idx)
- virtual void ImportIFMConfig (const std::vector< std::uint8_t > &bytes, std::uint16_t flags=0x0)
- virtual int ImportIFMApp (const std::vector< std::uint8_t > &bytes)
- virtual void SetPassword (std::string password="")
- json ToJSON () override
- void FromJSON (const json &j) override
- void ForceTrigger () override

Static Public Member Functions

- static Ptr MakeShared (const std::string &ip=ifm3d::DEFAULT_IP, const std::uint16_t xmlrpc_port=ifm3d::↩ DEFAULT_XMLRPC_PORT, const std::string &password=ifm3d::DEFAULT_PASSWORD)

Protected Member Functions

- void FromJSON_ (const json &j_curr, const json &j_new, std::function< void(const std::string &, const std↩ ::string &)> SetFunc, std::function< void()> SaveFunc, const std::string &name, int idx=-1)
- json **ToJSON_** (const bool open_session=true)
- json **getApplicationInfosToJSON** ()

Static Protected Member Functions

- static bool getAppJSON (int index, const json &j, json &app)

Protected Attributes

- std::unique_ptr< Impl > **pImpl**

3.2.1 Constructor & Destructor Documentation

3.2.1.1 Camera()

```
ifm3d::Camera::Camera (
            const std::string & ip = ifm3d::DEFAULT_IP,
            const std::uint16_t xmlrpc_port = ifm3d::DEFAULT_XMLRPC_PORT,
            const std::string & password = ifm3d::DEFAULT_PASSWORD )
```

Initializes the camera interface utilizing library defaults for password, ip, and xmlrpc port unless explicitly passed in.

Parameters

in	*ip*	The ip address of the camera
in	*xmlrpc_port*	The tcp port the sensor's XMLRPC server is listening on
in	*password*	Password required for establishing an "edit session" with the sensor. Edit sessions allow for mutating camera parameters and persisting those changes.

3.2.2 Member Function Documentation

3.2.2.1 ActiveApplication()

```
virtual int ifm3d::Camera::ActiveApplication ( )  [virtual]
```

Returns the integer index of the active application. A negative number indicates no application is marked as active on the sensor.

3.2.2.2 ApplicationList()

```
virtual json ifm3d::Camera::ApplicationList ( )  [virtual]
```

Delivers basic information about all applications stored on the device. A call to this function does not require establishing a session with the camera.

The returned information is encoded as an array of JSON objects. Each object in the array is basically a dictionary with the following keys: 'index', 'id', 'name', 'description', 'active'

Returns

A JSON encoding of the application information

Exceptions

ifm3d::error_t	upon error

3.2.2.3 ApplicationTypes()

```
virtual std::vector<std::string> ifm3d::Camera::ApplicationTypes ( )  [virtual]
```

Lists the valid application types supported by the sensor.

Returns

A vector of strings listing the available types of applications supported by the sensor. Each element of the vector is a string suitable to passing to `CreateApplication`.

Exceptions

ifm3d::error_t	upon error

3.2.2.4 CancelSession() [1/2]

```
virtual bool ifm3d::Camera::CancelSession ( )  [virtual]
```

Explictly stops the current session with the sensor.

NOTE: This function returns a boolean indicating the success/failure of cancelling the session. The reason we return a bool and explicitly supress exceptions is because we want to cancel any open sessions in the camera dtor and we do not want to throw in the dtor.

Returns

true if the session was cancelled properly, false if an exception was caught trying to close the session. Details will be logged.

3.2.2.5 CancelSession() [2/2]

```
virtual bool ifm3d::Camera::CancelSession (
            const std::string & sid )  [virtual]
```

Attempts to cancel a session with a particular session id.

Returns

true if the session was cancelled properly, false if an exception was caught trying to close the session. Details will be logged.

3.2.2.6 CopyApplication()

```
virtual int ifm3d::Camera::CopyApplication (
            int idx )  [virtual]
```

Creates a new application by copying the configuration of another application. The device will generate an ID for the new application and put it on a free index.

Parameters

in	*idx*	The index of the application to copy

Returns

Index of the new application

Exceptions

ifm3d::error_t	upon error

3.2.2.7 CreateApplication()

```
virtual int ifm3d::Camera::CreateApplication (
            const std::string & type = DEFAULT_APPLICATION_TYPE )  [virtual]
```

Creates a new application on the camera of the given type.

To figure out valid `type`s, you should call the `AvailableApplicationTypes()` method.

Upon creation of the application, the embedded device will initialize all parameters as necessary based on the type. However, based on the type, the application may not be in an *activatable* state. That is, it can be created and saved on the device, but it cannot be marked as active.

Parameters

in	*type*	The (optional) application type to create. By default, it will create a new "Camera" application.

Returns

The index of the new application.

3.2.2.8 DeleteApplication()

```
virtual void ifm3d::Camera::DeleteApplication (
            int idx )  [virtual]
```

Deletes the application at the specified index from the sensor.

Parameters

in	*idx*	The index of the application to delete throw ifm3d::error_t upon error

3.2.2.9 ExportIFMApp()

```
virtual std::vector<std::uint8_t> ifm3d::Camera::ExportIFMApp (
            int idx ) [virtual]
```

Export the application at the specified index into a byte array suitable for writing to a file. The exported bytes represent the IFM serialization of an application.

This function provides compatibility with tools like IFM's Vision Assistant.

Parameters

in	*idx*	The index of the application to export.

Returns

A vector of bytes representing the IFM serialization of the exported application.

Exceptions

ifm3d::error_t	upon error

3.2.2.10 ExportIFMConfig()

```
virtual std::vector<std::uint8_t> ifm3d::Camera::ExportIFMConfig ( ) [virtual]
```

Exports the entire camera configuration in a format compatible with Vision Assistant.

3.2.2.11 FactoryReset()

```
virtual void ifm3d::Camera::FactoryReset ( ) [virtual]
```

Sets the camera configuration back to the state in which it shipped from the ifm factory.

3.2.2.12 ForceTrigger()

```
void ifm3d::Camera::ForceTrigger ( ) [override], [virtual]
```

Sends a S/W trigger to the camera over XMLRPC.

The O3X does not S/W trigger over PCIC, so, this function has been developed specficially for it. For other sensors, this is a NOOP.

Reimplemented from ifm3d::CameraBase.

3.2.2.13 FromJSON()

```
void ifm3d::Camera::FromJSON (
            const json & j )  [override], [virtual]
```

Configures the camera based on the parameter values of the passed in JSON. This function is *the* way to tune the camera/application/imager/etc. parameters.

Parameters

in	*json*	A json object encoding a camera configuration to apply to the hardware.

- Device parameters are processed and saved persistently

Exceptions

ifm3d::error_t	upon error - if this throws an exception, you are encouraged to check the log file as a best effort is made to be as descriptive as possible as to the specific error that has occured.

Reimplemented from ifm3d::CameraBase.

3.2.2.14 FromJSON_()

```
void ifm3d::Camera::FromJSON_ (
            const json & j_curr,
            const json & j_new,
            std::function< void(const std::string &, const std::string &)> SetFunc,
            std::function< void()> SaveFunc,
            const std::string & name,
            int idx = -1 )  [protected]
```

Handles parsing a selected sub-tree of a potential input JSON file, setting the parameters as appropriate on the camera, and saving them persistently.

Parameters

in	*j_curr*	The current configuration
in	*j_new*	The desired configuration
in	*SetFunc*	The setter function to call for each parameter
in	*SaveFunc*	The function to call to persist the values
in	*name*	A descriptive name for the sub-tree (used to make log messages useful).
in	*idx*	An application index to put into edit mode prior to setting parameters.

3.2.2.15 getAppJSON()

```
static bool ifm3d::Camera::getAppJSON (
            int index,
            const json & j,
            json & app )  [static], [protected]
```

Return json of an app with given index from camera configuration json.

Parameters

in	*index*	Index of application to return
in	*j*	The current configuration
out	*app*	Output json of the application when found or empty json

Returns

True when application was found

3.2.2.16 Heartbeat()

```
virtual int ifm3d::Camera::Heartbeat (
            int hb )  [virtual]
```

Heartbeat messages are used to keep a session with the sensor alive. This function sends a heartbeat message to the sensor and sets when the next heartbeat message is required.

Parameters

in	*hb*	The time (seconds) of when the next heartbeat message will be required.

Returns

The current timeout interval in seconds for heartbeat messages.

Exceptions

ifm3d::error_t	upon error

3.2.2.17 ImagerTypes()

```
virtual std::vector<std::string> ifm3d::Camera::ImagerTypes ( )  [virtual]
```

Lists the valid imager types supported by the sensor.

Returns

A vector of strings listing the available types of imagers supported by the sensor.

Exceptions

ifm3d::error_t	upon error

3.2.2.18 ImportIFMApp()

```
virtual int ifm3d::Camera::ImportIFMApp (
            const std::vector< std::uint8_t > & bytes )  [virtual]
```

Import the IFM-encoded application.

This function provides compatibility with tools like IFM's Vision Assistant. An application configuration exported from VA, can be imported using this function.

Parameters

in	*bytes*	The raw bytes from the zip'd JSON file. NOTE: This function will base64 encode the data for tranmission over XML-RPC.

Returns

The index of the imported application.

3.2.2.19 ImportIFMConfig()

```
virtual void ifm3d::Camera::ImportIFMConfig (
            const std::vector< std::uint8_t > & bytes,
            std::uint16_t flags = 0x0 )  [virtual]
```

Imports an entire camera configuration from a format compatible with Vision Assistant.

3.2.2.20 MakeShared()

```
static Ptr ifm3d::Camera::MakeShared (
            const std::string & ip = ifm3d::DEFAULT_IP,
            const std::uint16_t xmlrpc_port = ifm3d::DEFAULT_XMLRPC_PORT,
            const std::string & password = ifm3d::DEFAULT_PASSWORD )  [static]
```

Factory function for instantiating the proper subclass based on h/w probing.

This function provides a convenient way for users of the library to write hardware independent code. This function probes the connected hardware and returns a proper subclass based upon the returned `DeviceType`. In the event that the hardware is not connected, the error is trapped and an instance of the base class is returned. The net result of not having an instance of a subclass is: 1) worse performance, 2) errors will come back from the sensor rather than the library – some of which may be hard to debug.

Parameters

in	*ip*	The ip address of the camera
in	*xmlrpc_port*	The tcp port the sensor's XMLRPC server is listening on
in	*password*	Password required for establishing an "edit session" with the sensor. Edit sessions allow for mutating camera parameters and persisting those changes.

3.2.2.21 Password()

```
virtual std::string ifm3d::Camera::Password ( )  [virtual]
```

The password associated with this Camera instance

3.2.2.22 RequestSession()

```
virtual std::string ifm3d::Camera::RequestSession ( )  [virtual]
```

Requests an edit-mode session with the camera.

In order to (permanently) mutate parameters on the camera, an edit session needs to be established. Only a single edit sesson may be established at any one time with the camera (think of it as a global mutex on the camera state – except if you ask for the mutex and it is already taken, an exception will be thrown).

Most typical use-cases for end-users will not involve establishing an edit-session with the camera. To mutate camera parameters, the `FromJSON` family of functions should be used, which, under-the-hood, on the user's behalf, will establish the edit session and gracefully close it. There is an exception. For users who plan to modulate imager parameters (temporary parameters) on the fly while running the framegrabber, managing the session manually is necessary. For this reason, we expose this method in the public `Camera` interface.

NOTE: The session timeout is implicitly set to `ifm3d::MAX_HEARTBEAT` after the session has been successfully established.

Returns

The session id issued or accepted by the camera (see IFM3D_SESSION_ID environment variable)

Exceptions

ifm3d::error_t	if an error is encountered.

3.2.2.23 SessionID()

```
virtual std::string ifm3d::Camera::SessionID ( )  [virtual]
```

Retrieves the active session id

3.2.2.24 SetCurrentTime()

```
virtual void ifm3d::Camera::SetCurrentTime (
            int epoch_secs = -1 )  [virtual]
```

Explicitly sets the current time on the camera.

Parameters

in	*epoch_secs*	Time since the Unix epoch in seconds. A value less than 0 will implicitly set the time to the current system time.

3.2.2.25 SetPassword()

```
virtual void ifm3d::Camera::SetPassword (
            std::string password = "" )  [virtual]
```

Sets or disable the password on the camera.

Parameters

in	*password*	is the password string. If the password is blank, password is disabled

Exceptions

ifm3d::error_t	upon error

3.2.2.26 SetTemporaryApplicationParameters()

```
virtual void ifm3d::Camera::SetTemporaryApplicationParameters (
            const std::unordered_map< std::string, std::string > & params )  [virtual]
```

Sets temporary application parameters in run mode.

The changes are not persistent and are lost when entering edit mode or turning the device off. The parameters "ExposureTime" and "ExposureTimeRatio" of the imager configuration are supported. All additional parameters are ignored (for now). Exposure times are clamped to their allowed range, depending on the exposure mode. The user must provide the complete set of parameters depending on the exposure mode, i.e., "ExposureTime" only for single exposure modes and both "ExposureTime" and "ExposureTimeRatio" for double exposure modes. Otherwise, behavior is undefined.

Parameters

in	*params*	The parameters to set on the camera.

Exceptions

ifm3d::error_t	upon error

3.2.2.27 ToJSON()

```
json ifm3d::Camera::ToJSON ( )  [override], [virtual]
```

Serializes the state of the camera to JSON.

The JSON interface returned here is the excellent `JSON for Modern C++`.

This function (along with its `std::string` equivalent `ToJSONStr()`) provides the primary gateway into obtaining the current parameter settings for the camera and PMD imager. Data returned from this function can be manipulated as a `json` object, then fed into `FromJSON(...)` to mutate parameter settings on the camera.

Returns

A JSON object representation of the current state of the hardware.

Exceptions

ifm3d::error_t	upon error

Reimplemented from ifm3d::CameraBase.

3.2.2.28 UnitVectors()

```
virtual std::vector<std::uint8_t> ifm3d::Camera::UnitVectors ( )  [virtual]
```

For cameras that support fetching the Unit Vectors over XML-RPC, this function will return those data as a binary blob.

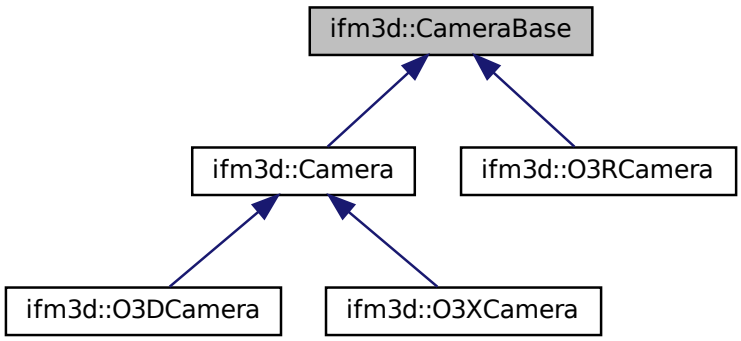
The documentation for this class was generated from the following file:

- /home/usmasslo/gitlab/ifm3d/modules/camera/include/ifm3d/camera/camera.h

3.3 ifm3d::CameraBase Class Reference

```
#include <camera_base.h>
```

Inheritance diagram for ifm3d::CameraBase:



Public Types

- enum class boot_mode : int { **PRODUCTIVE** = 0 , **RECOVERY** = 1 }
- enum class operating_mode : int { **RUN** = 0 , **EDIT** = 1 }
- enum class trigger_mode : int { **FREE_RUN** = 1 , **SW** = 2 }
- enum class import_flags : int { **GLOBAL** = 0x1 , **NET** = 0x2 , **APPS** = 0x10 }
- enum class spatial_filter : int { **OFF** = 0x0 , **MEDIAN** = 0x1 , **MEAN** = 0x2 , **BILATERAL** = 0x3 }
- enum class temporal_filter : int { **OFF** = 0x0 , **MEAN** = 0x1 , **ADAPTIVE_EXP** = 0x2 }
- enum class mfilt_mask_size : int { **_3x3** = 0 , **_5x5** = 1 }
- enum class **device_family** : int { **UNKNOWN** = 0 , **O3D** = 1 , **O3X** = 2 , **O3R** = 3 }
- enum class **swu_version** : int { **SWU_NOT_SUPPORTED** = 0 , **SWU_V1** = 1 , **SWU_V2** = 2 }
- using **Ptr** = std::shared_ptr< CameraBase >

Public Member Functions

- CameraBase (const std::string &ip=ifm3d::DEFAULT_IP, const std::uint16_t xmlrpc_port=ifm3d::DEFAULT↩ _XMLRPC_PORT)
- virtual ~CameraBase ()
- **CameraBase** (CameraBase &&)=delete
- CameraBase & **operator=** (CameraBase &&)=delete
- **CameraBase** (CameraBase &)=delete
- CameraBase & **operator=** (CameraBase &)=delete
- virtual std::string IP ()
- virtual std::uint16_t XMLRPCPort ()
- virtual void Reboot (const boot_mode &mode=ifm3d::CameraBase::boot_mode::PRODUCTIVE)
- virtual void ForceTrigger ()
- virtual std::string DeviceType (bool use_cached=true)
- virtual device_family WhoAmI ()
- virtual bool AmI (device_family family)
- virtual std::string DeviceParameter (const std::string &key)
- virtual std::vector< std::string > TraceLogs (int count)
- virtual json ToJSON ()
- virtual std::string ToJSONStr ()
- virtual void FromJSON (const json &j)
- virtual void FromJSONStr (const std::string &jstr)
- bool CheckMinimumFirmwareVersion (unsigned int major, unsigned int minor, unsigned int patch)
- virtual ifm3d::CameraBase::swu_version SwUpdateVersion ()

Static Public Member Functions

- static std::vector< ifm3d::IFMNetworkDevice > DeviceDiscovery ()

  This function Provides a convinent way to find all ifm devices on the network.
- static Ptr MakeShared (const std::string &ip=ifm3d::DEFAULT_IP, const std::uint16_t xmlrpc_port=ifm3d::↩ DEFAULT_XMLRPC_PORT, const std::string &password=ifm3d::DEFAULT_PASSWORD)

Protected Member Functions

- int DeviceID ()
- bool **checkDeviceID** (int deviceID, int minID, int maxID)
- std::shared_ptr< XMLRPCWrapper > **XWrapper** ()

Protected Attributes

- std::unique_ptr< Impl > **pImpl**
- std::string device_type_

3.3.1 Detailed Description

Software interface to an ifm 3D camera

The `Camera` class implements the underlying network protocol for communicating with the ifm hardware. Via this communication layer, this class exposes objects that can be used to mutate and tune the camera parameters including those of the underlying pmd imager.

3.3.2 Member Enumeration Documentation

3.3.2.1 boot_mode

```
enum class ifm3d::CameraBase::boot_mode :  int  [strong]
```

Camera boot up modes:

Productive: the normal runtime firmware comes up Recovery: allows you to flash new firmware

3.3.2.2 import_flags

```
enum class ifm3d::CameraBase::import_flags :  int  [strong]
```

Import flags used when importing a Vision Assistant configuration

3.3.2.3 mfilt_mask_size

```
enum class ifm3d::CameraBase::mfilt_mask_size : int [strong]
```

Convenient constants for median filter mask sizes

3.3.2.4 operating_mode

```
enum class ifm3d::CameraBase::operating_mode : int [strong]
```

Camera operating modes: run (streaming pixel data), edit (configuring the device/applications).

3.3.2.5 spatial_filter

```
enum class ifm3d::CameraBase::spatial_filter : int [strong]
```

Convenience constants for spatial filter types

3.3.2.6 temporal_filter

```
enum class ifm3d::CameraBase::temporal_filter : int [strong]
```

Convenience constants for temporal filter types

3.3.2.7 trigger_mode

```
enum class ifm3d::CameraBase::trigger_mode : int [strong]
```

Image acquisition trigger modes

3.3.3 Constructor & Destructor Documentation

3.3.3.1 CameraBase()

```
ifm3d::CameraBase::CameraBase (
            const std::string & ip = ifm3d::DEFAULT_IP,
            const std::uint16_t xmlrpc_port = ifm3d::DEFAULT_XMLRPC_PORT )
```

Initializes the camera interface utilizing library defaults for password, ip, and xmlrpc port unless explicitly passed in.

Parameters

in	*ip*	The ip address of the camera
in	*xmlrpc_port*	The tcp port the sensor's XMLRPC server is listening on
in	*password*	Password required for establishing an "edit session" with the sensor. Edit sessions allow for mutating camera parameters and persisting those changes.

3.3.3.2 ~CameraBase()

```
virtual ifm3d::CameraBase::~CameraBase ( )  [virtual]
```

The dtor will cancel any open edit sessions with the camera.

3.3.4 Member Function Documentation

3.3.4.1 AmI()

```
virtual bool ifm3d::CameraBase::AmI (
            device_family family )  [virtual]
```

This is a convenience function for checking whether a device is one of the specified device family

Parameters

in	*family*	The family to check for

Returns

true if the device is part of the family

3.3.4.2 CheckMinimumFirmwareVersion()

```
bool ifm3d::CameraBase::CheckMinimumFirmwareVersion (
            unsigned int major,
            unsigned int minor,
            unsigned int patch )
```

Checks for a minimum ifm camera software version

Parameters

in	*major*	Major version of software
in	*minor*	Minor Version of software
in	*patch*	Patch Number of software

return True if current software version is greater or equal to the value passed

3.3.4.3 DeviceDiscovery()

```
static std::vector<ifm3d::IFMNetworkDevice> ifm3d::CameraBase::DeviceDiscovery ( )  [static]
```

This function Provides a convinent way to find all ifm devices on the network.

Returns

: vector of ip-address all the discovered devices on network.

3.3.4.4 DeviceID()

```
int ifm3d::CameraBase::DeviceID ( )  [protected]
```

Implements the serialization of the camera state to JSON.

Parameters

in	*open_session*	if false function will work on already opened session

Returns

A JSON object representation of the current camera state.

3.3.4.5 DeviceParameter()

```
virtual std::string ifm3d::CameraBase::DeviceParameter (
            const std::string & key )  [virtual]
```

Convenience accessor for extracting a device parameters (i.e., no edit session created on the camera)

3.3.4.6 DeviceType()

```
virtual std::string ifm3d::CameraBase::DeviceType (
            bool use_cached = true )  [virtual]
```

This is a convenience function for extracting out the device type of the connected camera. The primary intention of this function is for internal usage (i.e., to trigger conditional logic based on the model hardware we are talking to) however, it will likely be useful in application-level logic as well, so, it is available in the public interface.

Parameters

in	*use_cached*	If set to true, a cached lookup of the device type will be used as the return value. If false, it will make a network call to the camera to get the "real" device type. The only reason for setting this to `false` would be if you expect over the lifetime of your camera instance that you will swap out (for example) an O3D for an O3X (or vice versa) – literally, swapping out the network cables while an object instance is still alive. If that is not something you are worried about, leaving this set to true should result in a signficant performance increase.

3.3.4.7 ForceTrigger()

```
virtual void ifm3d::CameraBase::ForceTrigger ( )  [virtual]
```

Sends a S/W trigger to the camera over XMLRPC.

The O3X does not S/W trigger over PCIC, so, this function has been developed specficially for it. For other sensors, this is a NOOP.

Reimplemented in ifm3d::Camera.

3.3.4.8 FromJSON()

```
virtual void ifm3d::CameraBase::FromJSON (
            const json & j )  [virtual]
```

Configures the camera based on the parameter values of the passed in JSON. This function is *the* way to tune the camera/application/imager/etc. parameters.

Parameters

in	*json*	A json object encoding a camera configuration to apply to the hardware.

- Device parameters are processed and saved persistently

Exceptions

ifm3d::error_t	upon error - if this throws an exception, you are encouraged to check the log file as a best effort is made to be as descriptive as possible as to the specific error that has occured.

Reimplemented in ifm3d::Camera, and ifm3d::O3RCamera.

3.3.4.9 FromJSONStr()

```
virtual void ifm3d::CameraBase::FromJSONStr (
            const std::string & jstr )  [virtual]
```

Accepts a string with properly formatted/escaped JSON text, converts it to a `json` object, and call `FromJSON()` on it.

See also

FromJSON

3.3.4.10 IP()

```
virtual std::string ifm3d::CameraBase::IP ( )  [virtual]
```

The IP address associated with this Camera instance

3.3.4.11 MakeShared()

```
static Ptr ifm3d::CameraBase::MakeShared (
            const std::string & ip = ifm3d::DEFAULT_IP,
            const std::uint16_t xmlrpc_port = ifm3d::DEFAULT_XMLRPC_PORT,
            const std::string & password = ifm3d::DEFAULT_PASSWORD )  [static]
```

Factory function for instantiating the proper subclass based on h/w probing.

This function provides a convenient way for users of the library to write hardware independent code. This function probes the connected hardware and returns a proper subclass based upon the returned `DeviceType`. In the event that the hardware is not connected, the error is trapped and an instance of the base class is returned. The net result of not having an instance of a subclass is: 1) worse performance, 2) errors will come back from the sensor rather than the library – some of which may be hard to debug.

Parameters

in	*ip*	The ip address of the camera
in	*xmlrpc_port*	The tcp port the sensor's XMLRPC server is listening on
in	*password*	Password required for establishing an "edit session" with the sensor. Edit sessions allow for mutating camera parameters and persisting those changes.

3.3.4.12 Reboot()

```
virtual void ifm3d::CameraBase::Reboot (
            const boot_mode & mode = ifm3d::CameraBase::boot_mode::PRODUCTIVE )  [virtual]
```

Reboot the sensor

Parameters

in	*mode*	The system mode to boot into upon restart of the sensor

Exceptions

ifm3d::error_t	upon error

Reimplemented in ifm3d::O3RCamera.

3.3.4.13 SwUpdateVersion()

```
virtual ifm3d::CameraBase::swu_version ifm3d::CameraBase::SwUpdateVersion ( )  [virtual]
```

Checks the swupdater version supported by device

Returns

sw_version supported by device

Reimplemented in ifm3d::O3RCamera.

3.3.4.14 ToJSON()

```
virtual json ifm3d::CameraBase::ToJSON ( )  [virtual]
```

Serializes the state of the camera to JSON.

The JSON interface returned here is the excellent `JSON for Modern C++`.

This function (along with its `std::string` equivalent `ToJSONStr()`) provides the primary gateway into obtaining the current parameter settings for the camera and PMD imager. Data returned from this function can be manipulated as a `json` object, then fed into `FromJSON(...)` to mutate parameter settings on the camera.

Returns

A JSON object representation of the current state of the hardware.

Exceptions

ifm3d::error_t	upon error

Reimplemented in ifm3d::Camera, and ifm3d::O3RCamera.

3.3.4.15 ToJSONStr()

```
virtual std::string ifm3d::CameraBase::ToJSONStr ( )  [virtual]
```

A stringified version of the JSON object returned by `ToJSON()`.

Returns

A string version of the Camera's JSON representation.

See also

ToJSON

3.3.4.16 TraceLogs()

```
virtual std::vector<std::string> ifm3d::CameraBase::TraceLogs (
            int count )  [virtual]
```

Delivers the trace log from the camera A session is not required to call this function.

Returns

A 'vector' of 'std::string' for each entry in the tracelog

Exceptions

ifm3d::error_t	upon error

3.3.4.17 WhoAmI()

```
virtual device_family ifm3d::CameraBase::WhoAmI ( )  [virtual]
```

This function can be used to retrieve the family of the connected device

Returns

the device family of the connected device.

Reimplemented in ifm3d::O3DCamera, ifm3d::O3RCamera, and ifm3d::O3XCamera.

3.3.4.18 XMLRPCPort()

```
virtual std::uint16_t ifm3d::CameraBase::XMLRPCPort ( )  [virtual]
```

The XMLRPC Port associated with this Camera instance

3.3.5 Member Data Documentation

3.3.5.1 device_type_

```
std::string ifm3d::CameraBase::device_type_  [protected]
```

The cached device type of the connected device

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

- /home/usmasslo/gitlab/ifm3d/modules/camera/include/ifm3d/camera/camera_base.h

3.4 ifm3d::DistanceImageInfo Class Reference

Public Member Functions

- **DistanceImageInfo** (const float dist_res, const float ampl_res, const std::vector< float > &amp_norm_fctrs, const std::vector< float > &extr_opt_to_usr, const IntrinsicCalibration &intr_calib, const IntrinsicCalibration &inv_intr_calib, const std::vector< std::uint16_t > &distance_buffer, const std::vector< std::uint16_t > &amplitude_buffer, const std::vector< uint64_t > &timestamps_nsec, const std::vector< float > &exposure_times_sec, const std::uint32_t width, const std::uint32_t height)
- std::vector< std::uint8_t > **getXYZDVector** ()
- std::vector< std::uint8_t > **getAmplitudeVector** ()
- auto **getExtrinsicOpticToUser** ()
- auto **getIntrinsicCalibration** ()
- auto **getInverseIntrinsicCalibration** ()
- auto **getNPTS** ()
- std::vector< uint64_t > **getTimestamps** ()

  returns the timestamps in nano seconds
- std::vector< float > **getExposureTimes** ()

  return the exposure time for each phase data

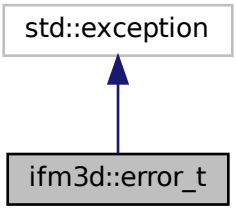
The documentation for this class was generated from the following file:

- /home/usmasslo/gitlab/ifm3d/modules/framegrabber/include/ifm3d/fg/distance_image_info.h

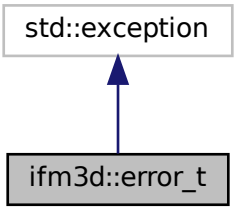
3.5 ifm3d::error_t Class Reference

`#include <err.h>`

Inheritance diagram for ifm3d::error_t:



Collaboration diagram for ifm3d::error_t:



Public Member Functions

- error_t (int errnum, const std::string &msg="")
- virtual const char * what () const noexcept
- int code () const noexcept
- const char * message () const noexcept

3.5.1 Detailed Description

Exception wrapper for library and system errors encountered by the library.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 error_t()

```
ifm3d::error_t::error_t (
            int errnum,
            const std::string & msg = "" )
```

The ctor simply sets the error value and optional message into a local instance variables that may be retrieved with a call to code() and message().

3.5.3 Member Function Documentation

3.5.3.1 code()

```
int ifm3d::error_t::code ( ) const  [noexcept]
```

Accessor to the underlying error code

3.5.3.2 message()

```
const char* ifm3d::error_t::message ( ) const  [noexcept]
```

Accessor to the underlying error msg

3.5.3.3 what()

```
virtual const char* ifm3d::error_t::what ( ) const  [virtual], [noexcept]
```

Exception message

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

- /home/usmasslo/gitlab/ifm3d/modules/camera/include/ifm3d/camera/err.h

3.6 ifm3d::FrameGrabber Class Reference

```
#include <frame_grabber.h>
```

Public Types

- using **Ptr** = std::shared_ptr< FrameGrabber >

Public Member Functions

- FrameGrabber (ifm3d::CameraBase::Ptr cam, std::uint16_t mask=ifm3d::DEFAULT_SCHEMA_MASK, const std::uint16_t nat_pcic_port=ifm3d::PCIC_PORT)
- virtual ~FrameGrabber ()
- **FrameGrabber** (FrameGrabber &&)=delete
- FrameGrabber & **operator=** (FrameGrabber &&)=delete
- **FrameGrabber** (FrameGrabber &)=delete
- FrameGrabber & **operator=** (const FrameGrabber &)=delete
- void SWTrigger ()
- template<typename T >
  bool WaitForFrame (ifm3d::ByteBuffer< T > *buff, long timeout_millis=0, bool copy_buff=false, bool organize=true)

Protected Member Functions

- bool WaitForFrame (long timeout_millis, std::function< void(std::vector< std::uint8_t > &)> set_bytes)

3.6.1 Detailed Description

Implements a TCP FrameGrabber connected to the camera passed to its ctor

3.6.2 Constructor & Destructor Documentation

3.6.2.1 FrameGrabber()

```
ifm3d::FrameGrabber::FrameGrabber (
            ifm3d::CameraBase::Ptr cam,
            std::uint16_t mask = ifm3d::DEFAULT_SCHEMA_MASK,
            const std::uint16_t nat_pcic_port = ifm3d::PCIC_PORT )
```

Stores a reference to the passed in camera shared pointer and starts a worker thread to stream in pixel data from the device.

Parameters

in	*cam*	The camera instance to grab frames from
in	*mask*	A bitmask encoding the image acquisition schema to stream in from the camera.
in	*nat_pcic_port*	Port for devices behind NAT router, user must provide this value according to there NAT router configuration.

3.6.2.2 ~FrameGrabber()

```
virtual ifm3d::FrameGrabber::~FrameGrabber ( )  [virtual]
```

Cleans up resources held by the framegrabbing thread object and blocks until the operating system thread stops.

3.6.3 Member Function Documentation

3.6.3.1 SWTrigger()

```
void ifm3d::FrameGrabber::SWTrigger ( )
```

Triggers the camera for image acquisition

You should be sure to set the `TriggerMode` for your application to `SW` in order for this to be effective. This function simply does the triggering, data are still received asynchronously via `WaitForFrame()`.

Calling this function when the camera is not in `SW` trigger mode or on a device that does not support software-trigger should result in a NOOP and no error will be returned (no exceptions thrown). However, we do not recommend calling this function in a tight framegrabbing loop when you know it is not needed. The "cost" of the NOOP is undefined and incurring it is not recommended.

3.6.3.2 WaitForFrame() [1/2]

```
template<typename T >
bool ifm3d::FrameGrabber::WaitForFrame (
            ifm3d::ByteBuffer< T > * buff,
            long timeout_millis = 0,
            bool copy_buff = false,
            bool organize = true )  [inline]
```

This function is used to grab and parse out time synchronized image data from the camera. It will call `SetBytes` on the passed in `ByteBuffer` as well as (optionally, but by default) call `Organize`. Calling `Organize` is the default behavior so the `buff` output parameter is assumed to be synchronized and ready for analysis provided this function returns true. In certain applications, it may be a performance enhancement to not call `Organize` but rather handle that outside of the `FrameGrabber`.

Parameters

`out`	*buff*	A pointer to an `ifm3d::ByteBuffer<Dervied>` object to update with the latest data from the camera.
`in`	*timeout_millis*	Timeout in millis to wait for new image data from the FrameGrabber. If `timeout_millis` is set to 0, this function will block indefinitely.
`in`	*copy_buff*	Flag indicating whether the framegrabber's internal buffer should be copied (O(n)) or swapped (O(1)) with the raw bytes of the passed in `buff`. You should only flag this as `true` if you are planning to use multiple clients with a single `FrameGrabber` – even then, think carefully before copying data around.
`in`	*organize*	Flag indicating whether or not `Organize` should be called on the `ByteBuffer` before returning.

Returns

true if a new buffer was acquired w/in `timeout_millis`, false otherwise.

3.6.3.3 WaitForFrame() [2/2]

```
bool ifm3d::FrameGrabber::WaitForFrame (
            long timeout_millis,
            std::function< void(std::vector< std::uint8_t > &)> set_bytes )  [protected]
```

This is a convenience/wrapper function used to proxy `WaitForFrame` calls through to the pimpl class w/o requiring the pimpl to know about the ByteBuffer CRTP class hierarchy – i.e., it operates on a vector of bytes not an `ifm3d::ByteBuffer<Derived>`.

Parameters

in	*timeout_millis*	Timeout in millis to wait for new image data from the FrameGrabber. If `timeout_millis` is set to 0, this function will block indefinitely.
in	*set_bytes*	A mutator function that will be called with the latest frame data bytes from the camera.

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

- /home/usmasslo/gitlab/ifm3d/modules/framegrabber/include/ifm3d/fg/frame_grabber.h

3.7 ifm3d::IFMNetworkDevice Class Reference

Public Member Functions

- **IFMNetworkDevice** (Data &data, const std::string &ip_address_via_interface)
- std::string GetIPAddress () const
- std::string GetMACAddress () const
- std::string GetNetmask () const
- std::string GetGateway () const
- uint16_t GetPort () const
- uint16_t GetFlag () const
- std::string GetHostName () const
- std::string GetDeviceName () const
- uint16_t GetVendorId () const
- uint16_t GetDeviceId () const
- std::string GetFoundVia () const

3.7.1 Member Function Documentation

3.7.1.1 GetDeviceId()

```
uint16_t ifm3d::IFMNetworkDevice::GetDeviceId ( ) const
```

Device ID of the device

3.7.1.2 GetDeviceName()

```
std::string ifm3d::IFMNetworkDevice::GetDeviceName ( ) const
```

Device name

3.7.1.3 GetFlag()

```
uint16_t ifm3d::IFMNetworkDevice::GetFlag ( ) const
```

Device gives some additional information via those flags

3.7.1.4 GetFoundVia()

```
std::string ifm3d::IFMNetworkDevice::GetFoundVia ( ) const
```

Founf via interface

3.7.1.5 GetGateway()

```
std::string ifm3d::IFMNetworkDevice::GetGateway ( ) const
```

Gateway of the device

3.7.1.6 GetHostName()

```
std::string ifm3d::IFMNetworkDevice::GetHostName ( ) const
```

Hostname of the device

3.7.1.7 GetIPAddress()

```
std::string ifm3d::IFMNetworkDevice::GetIPAddress ( ) const
```

Ip Address of the device

3.7.1.8 GetMACAddress()

```
std::string ifm3d::IFMNetworkDevice::GetMACAddress ( ) const
```

Mac Address of the device

3.7.1.9 GetNetmask()

```
std::string ifm3d::IFMNetworkDevice::GetNetmask ( ) const
```

Netmask of the network of camera

3.7.1.10 GetPort()

```
uint16_t ifm3d::IFMNetworkDevice::GetPort ( ) const
```

Port on which device discovery is done

3.7.1.11 GetVendorId()

```
uint16_t ifm3d::IFMNetworkDevice::GetVendorId ( ) const
```

Vendor ID of the device

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

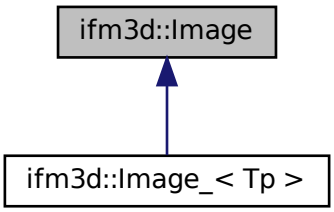
- /home/usmasslo/gitlab/ifm3d/modules/camera/include/ifm3d/camera/ifm_network_device.h

3.8 ifm3d::Image Class Reference

The class Image represent a STL conatiner to stored image data from the ifm devices in 2 dimension and supports multiple channel. data is stores in sequnetial memory layout and class provides function template to access the pixel. Creating an Image object :

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

Inheritance diagram for ifm3d::Image:



Classes

- struct Iterator

Public Member Functions

- Image ()
- **Image** (const std::uint32_t cols, const std::uint32_t rows, const std::uint32_t nchannel, ifm3d::pixel_format format)
- **Image** (Image &&)=default
- Image & **operator=** (Image &&)=default
- **Image** (const Image &)=default
- Image & **operator=** (const Image &)=default
- void **create** (const std::uint32_t cols, const std::uint32_t rows, const std::uint32_t nchannel, ifm3d::pixel_↩ format format)
- Image **clone** () const

  Creates a full copy of the array and the underlying data.
- std::uint32_t **height** () const
- std::uint32_t **width** () const
- std::uint32_t **nchannels** () const
- ifm3d::pixel_format **dataFormat** () const
- template<typename T = std::uint8_t>
  T * ptr (const std::uint32_t row)

  returns a pointer to the specified Image row.
- template<typename T = std::uint8_t>
  T * ptr (const std::uint32_t row, const std::uint32_t col)

  Pointer to the Pixel at row,col.
- template<typename T >
  T & **at** (const std::size_t index)
- template<typename T >
  T & **at** (const std::uint32_t row, const std::uint32_t col)
- template<typename T >
  void **setTo** (const T val, ifm3d::Image &mask)
- template<typename T >
  Iterator< T > **begin** ()
- template<typename T >
  Iterator< T > **end** ()

3.8.1 Detailed Description

The class Image represent a STL conatiner to stored image data from the ifm devices in 2 dimension and supports multiple channel. data is stores in sequnetial memory layout and class provides function template to access the pixel. Creating an Image object :

- Use the Create(cols, rows, nchannel, ifm3d::pixel_format) method or the similar Image(cols,rows, nchannel, type) constructor.

For example, FORMAT_8U means a 8-bit array, FORMAT_32F floating-point array, and so on.

```
//a 100 x 100 Image of type 8U
ifm3d::Image image(100,100,1,ifm3d::FORMAT_8U);
// and now turn image to a 10 x10 3-channel 8-bit matrix.
// The old content will be deallocated
image.create(10,10,3,ifm3d::FORMAT_8U);
```

note: create() allocates new memory.

- Accessing the pixels use at<T>(index) or at<T>(i,j) to access the pixel this return the reference to the pixel. A pixel is defined as structure of n-channel values at a given index or pixel position in 2D array

to access a pixel in Image I (100,100,1,ifm3d::FORMAT_8U) at 50,50 position
```
auto pixel = I<uint8_t>(50,50);
// if working as Index array then
auto index = 50*100 + 50 ;
auto pixel = I<uint8_t>(index);
```

changing the pixel value can be done as follow : writing 100 at pixel postion 50,50
```
I<uint8_t>(50,50) = 100;
I<uint8_t>(index) = 100;
```

to access a pixel in n-channel Image I (100,100,3,ifm3d::FORMAT_8U) at 50,50 position This will be the case accessing the values for 3 channel Image

as pixel is structure of the values of n-chanel at given position.
```
auto pixel = I<Point3D<uint8_t»(50,50);
//now individual channel values can be access with
val[0], val[1] , val[2]
```

-Processing the whole array If you need to process a whole Image, the most efficient way is to get the pointer to the row first, and then just use the plain C operator [] :
```
Image I(100,100,1,FORMAT_8U);
 for(int i = 0; i < I.height(); i++)
 {
     const uint8_t* rowi = M.ptr<uint8_t>(i);
     for(int j = 0; j < I.width(); j++)
         {
           //some operation here
         }
 }
```

One can aslo use range based for loops with adapter explained in ifm3d::IteratorAdapter section

3.8.2 Constructor & Destructor Documentation

3.8.2.1 Image()

```
ifm3d::Image::Image ( )
```

These are various constructors that form a Image. default constructor for forming a Image user furher needs to call create Method to actually allocates the Memory

3.8.3 Member Function Documentation

3.8.3.1 ptr() [1/2]

```
template<typename T = std::uint8_t>
T* ifm3d::Image::ptr (
             const std::uint32_t row )
```

returns a pointer to the specified Image row.

Parameters

row	number

3.8.3.2 ptr() [2/2]

```
template<typename T = std::uint8_t>
T* ifm3d::Image::ptr (
             const std::uint32_t row,
             const std::uint32_t col )
```

Pointer to the Pixel at row,col.

Parameters

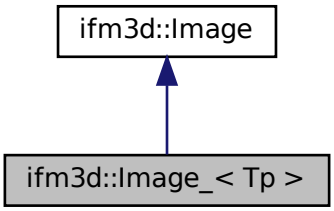
row	1st dimension index
col	2nd dimension index

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

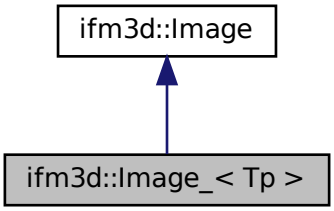
- /home/usmasslo/gitlab/ifm3d/modules/stlimage/include/ifm3d/stlimage/image.h

3.9 ifm3d::Image_< Tp > Class Template Reference

Inheritance diagram for ifm3d::Image_< Tp >:



Collaboration diagram for ifm3d::Image_< Tp >:



Public Member Functions

- **Image_** (const std::uint32_t cols, const std::uint32_t rows)
- **Image_** (Image_< Tp > &&)=default
- Image_ & **operator=** (Image_< Tp > &&)=default
- **Image_** (const Image_< Tp > &)=default
- Image_ & **operator=** (const Image_< Tp > &)=default
- **Image_** (const Image &)
- Image_ & **operator=** (const Image &)
- void **create** (const std::uint32_t cols, const std::uint32_t rows)
- Image_ **clone** () const

  Creates a full copy of the array and the underlying data.
- std::uint32_t **height** () const
- std::uint32_t **width** () const
- std::uint32_t **nchannels** () const
- ifm3d::pixel_format **dataFormat** () const
- Tp * ptr (const std::uint32_t row)

  returns a pointer to the specified Image row.
- Tp * ptr (const std::uint32_t row, const std::uint32_t col)

  Pointer to the Pixel at row,col.
- Tp & **at** (const std::size_t index)
- Tp & **at** (const std::uint32_t row, const std::uint32_t col)
- void **setTo** (const Tp val, ifm3d::Image &mask)
- Iterator< Tp > **begin** ()
- Iterator< Tp > **end** ()

3.9.1 Member Function Documentation

3.9.1.1 ptr() [1/2]

```
template<typename Tp >
Tp* ifm3d::Image_< Tp >::ptr (
            const std::uint32_t row )
```

returns a pointer to the specified Image row.

Parameters

row	number

3.9.1.2 ptr() [2/2]

```
template<typename Tp >
Tp* ifm3d::Image_< Tp >::ptr (
            const std::uint32_t row,
            const std::uint32_t col )
```

Pointer to the Pixel at row,col.

Parameters

row	1st dimension index
col	2nd dimension index

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

- /home/usmasslo/gitlab/ifm3d/modules/stlimage/include/ifm3d/stlimage/image.h

3.10 ifm3d::IntrinsicCalibration Struct Reference

Public Attributes

- uint32_t **model_iD**
- float **model_parameters** [NR_MODEL_PARAMS]

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

- /home/usmasslo/gitlab/ifm3d/modules/framegrabber/include/ifm3d/fg/distance_image_info.h

3.11 ifm3d::Image::Iterator< T > Struct Template Reference

Public Types

- using **iterator_category** = std::random_access_iterator_tag
- using **difference_type** = std::ptrdiff_t
- using **value_type** = T
- using **pointer** = T *
- using **reference** = T &

Public Member Functions

- **Iterator** (uint8_t *ptr)
- reference **operator*** () const
- pointer **operator->** ()
- Iterator & **operator++** ()
- Iterator **operator++** (std::int32_t)
- bool **operator-** (const Iterator &rhs) const noexcept

Friends

- bool **operator==** (const Iterator &a, const Iterator &b)
- bool **operator!=** (const Iterator &a, const Iterator &b)

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

- /home/usmasslo/gitlab/ifm3d/modules/stlimage/include/ifm3d/stlimage/image.h

3.12 ifm3d::IteratorAdapter< T > Class Template Reference

Public Member Functions

- **IteratorAdapter** (Image &it)
- auto **begin** ()
- auto **end** ()

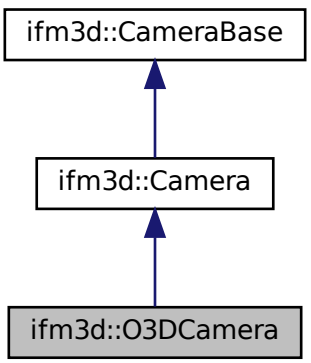
The documentation for this class was generated from the following file:

- /home/usmasslo/gitlab/ifm3d/modules/stlimage/include/ifm3d/stlimage/image.h

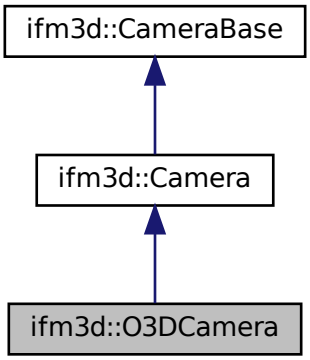
3.13 ifm3d::O3DCamera Class Reference

`#include <camera_o3d.h>`

Inheritance diagram for ifm3d::O3DCamera:



Collaboration diagram for ifm3d::O3DCamera:



Public Types

- using **Ptr** = std::shared_ptr< O3DCamera >

Public Member Functions

- **O3DCamera** (const std::string &ip=ifm3d::DEFAULT_IP, const std::uint16_t xmlrpc_port=ifm3d::DEFAULT↩ _XMLRPC_PORT, const std::string &password=ifm3d::DEFAULT_PASSWORD)
- **O3DCamera** (O3DCamera &&)=delete
- O3DCamera & **operator=** (O3DCamera &&)=delete
- **O3DCamera** (O3DCamera &)=delete
- O3DCamera & **operator=** (O3DCamera &)=delete
- std::unordered_map< std::string, std::string > TimeInfo () override
- device_family WhoAmI () override

Additional Inherited Members

3.13.1 Detailed Description

Camera specialization for O3D

3.13.2 Member Function Documentation

3.13.2.1 TimeInfo()

```
std::unordered_map<std::string, std::string> ifm3d::O3DCamera::TimeInfo ( )  [override], [virtual]
```

Reimplemented from ifm3d::Camera.

3.13.2.2 WhoAmI()

```
device_family ifm3d::O3DCamera::WhoAmI ( )  [override], [virtual]
```

This function can be used to retrieve the family of the connected device

Returns

the device family of the connected device.

Reimplemented from ifm3d::CameraBase.

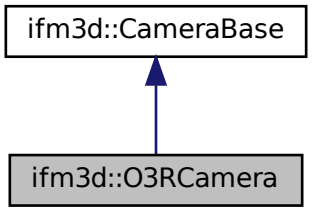
The documentation for this class was generated from the following file:

- /home/usmasslo/gitlab/ifm3d/modules/camera/include/ifm3d/camera/camera_o3d.h

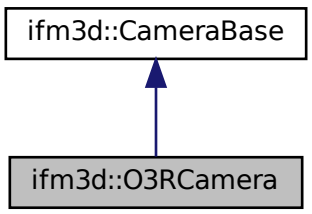
3.14 ifm3d::O3RCamera Class Reference

```
#include <camera_o3r.h>
```

Inheritance diagram for ifm3d::O3RCamera:



Collaboration diagram for ifm3d::O3RCamera:



Public Types

- using **Ptr** = std::shared_ptr< O3RCamera >

Public Member Functions

- **O3RCamera** (const std::string &ip=ifm3d::DEFAULT_IP, const std::uint16_t xmlrpc_port=ifm3d::DEFAULT↩ _XMLRPC_PORT)
- **O3RCamera** (O3RCamera &&)=delete
- O3RCamera & **operator=** (O3RCamera &&)=delete
- **O3RCamera** (O3RCamera &)=delete
- O3RCamera & **operator=** (O3RCamera &)=delete
- virtual void FactoryReset (bool keepNetworkSettings)
- json GetSchema ()
- json Get (const std::vector< std::string > &path=std::vector< std::string >())
- json ResolveConfig (const json::json_pointer &ptr)
- void Set (const json &j)
- json GetInit ()
- void SaveInit ()
- std::string GetInitStatus ()
- void Lock (const std::string &password)
- void Unlock (const std::string &password)
- std::vector< PortInfo > Ports ()
- PortInfo Port (const std::string &port)
- void Reboot (const boot_mode &mode=ifm3d::CameraBase::boot_mode::PRODUCTIVE) override
- device_family WhoAmI () override
- ifm3d::CameraBase::swu_version SwUpdateVersion () override
- json ToJSON () override
- void FromJSON (const json &j) override

Additional Inherited Members

3.14.1 Detailed Description

Camera specialization for O3R

3.14.2 Member Function Documentation

3.14.2.1 FactoryReset()

```
virtual void ifm3d::O3RCamera::FactoryReset (
            bool keepNetworkSettings )  [virtual]
```

Sets the camera configuration back to the state in which it shipped from the ifm factory.

Parameters

in	*keepNetworkSettings*	a bool indicating wether to keep the current network settings

3.14.2.2 FromJSON()

```
void ifm3d::O3RCamera::FromJSON (
            const json & j )  [override], [virtual]
```

Configures the camera based on the parameter values of the passed in JSON. This function is *the* way to tune the camera/application/imager/etc. parameters.

Parameters

in	*json*	A json object encoding a camera configuration to apply to the hardware.

- Device parameters are processed and saved persistently

Exceptions

ifm3d::error_t	upon error - if this throws an exception, you are encouraged to check the log file as a best effort is made to be as descriptive as possible as to the specific error that has occured. Equivalent to Set() followed by SaveInit()

Reimplemented from ifm3d::CameraBase.

3.14.2.3 Get()

```
json ifm3d::O3RCamera::Get (
            const std::vector< std::string > & path = std::vector< std::string >() )
```

Returns the configuration formatted as JSON based on a path. If the path is empty, returns the whole configuration.

Parameters

in	*path*	A List of JSON path fragments to retrieve the information for

Returns

The JSON configuration for the list of object path fragments

3.14.2.4 GetInit()

```
json ifm3d::O3RCamera::GetInit ( )
```

Return the initial JSON configuration.

Returns

The initial JSON configuration

3.14.2.5 GetInitStatus()

```
std::string ifm3d::O3RCamera::GetInitStatus ( )
```

Returns the init status of the device

Returns

The init status of the device

3.14.2.6 GetSchema()

```
json ifm3d::O3RCamera::GetSchema ( )
```

Return the current JSON schema configuration

Returns

The current JSON schema configuration

3.14.2.7 Lock()

```
void ifm3d::O3RCamera::Lock (
            const std::string & password )
```

Release the lock from the Device

Parameters

in	*password*	the password used to unlock the device

3.14.2.8 Port()

```
PortInfo ifm3d::O3RCamera::Port (
            const std::string & port )
```

Returns information about a given physical port

Parameters

in	*port*	the port for which to get the information

Returns

the port information

3.14.2.9 Ports()

```
std::vector<PortInfo> ifm3d::O3RCamera::Ports ( )
```

Returns a list containing information about all connected physical ports

Returns

the list of ports

3.14.2.10 Reboot()

```
void ifm3d::O3RCamera::Reboot (
            const boot_mode & mode = ifm3d::CameraBase::boot_mode::PRODUCTIVE )  [override],
[virtual]
```

Reboot the sensor

Parameters

in	*mode*	The system mode to boot into upon restart of the sensor

Exceptions

ifm3d::error_t	upon error

Reimplemented from ifm3d::CameraBase.

3.14.2.11 ResolveConfig()

```
json ifm3d::O3RCamera::ResolveConfig (
            const json::json_pointer & ptr )
```

Returns a part of the configuration formatted as JSON based on a JSON pointer.

Parameters

in	*ptr*	A JSON pointer to retrieve the information for

Returns

The partial JSON configuration for the given JSON pointer

3.14.2.12 SaveInit()

```
void ifm3d::O3RCamera::SaveInit ( )
```

Save to current temporary JSON configuration as initial JSON configuration

3.14.2.13 Set()

```
void ifm3d::O3RCamera::Set (
            const json & j )
```

Overwrites parts of the temporary JSON configuration which is achieved by merging the provided JSON fragment with the current temporary JSON.

Parameters

in	*j*	The new temporay JSON configuration of the device.

3.14.2.14 SwUpdateVersion()

```
ifm3d::CameraBase::swu_version ifm3d::O3RCamera::SwUpdateVersion ( )  [override], [virtual]
```

Checks the swupdater version supported by device

Returns

sw_version supported by device

Reimplemented from ifm3d::CameraBase.

3.14.2.15 ToJSON()

```
json ifm3d::O3RCamera::ToJSON ( )  [override], [virtual]
```

Serializes the state of the camera to JSON.

The JSON interface returned here is the excellent JSON for Modern C++.

This function (along with its `std::string` equivalent `ToJSONStr()`) provides the primary gateway into obtaining the current parameter settings for the camera and PMD imager. Data returned from this function can be manipulated as a `json` object, then fed into `FromJSON(...)` to mutate parameter settings on the camera.

Returns

A JSON object representation of the current state of the hardware.

Exceptions

ifm3d::error_t	upon error Equivalent to the Get() method

Reimplemented from ifm3d::CameraBase.

3.14.2.16 Unlock()

```
void ifm3d::O3RCamera::Unlock (
            const std::string & password )
```

Locks the device until it is unlocked. If the device is unlocked and an empty password is provided the password protection is removed.

Parameters

in	*password*	the password used to lock the device

3.14.2.17 WhoAmI()

```
device_family ifm3d::O3RCamera::WhoAmI ( )  [override], [virtual]
```

This function can be used to retrieve the family of the connected device

Returns

the device family of the connected device.

Reimplemented from ifm3d::CameraBase.

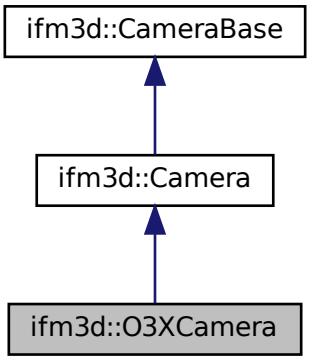
The documentation for this class was generated from the following file:

- /home/usmasslo/gitlab/ifm3d/modules/camera/include/ifm3d/camera/camera_o3r.h

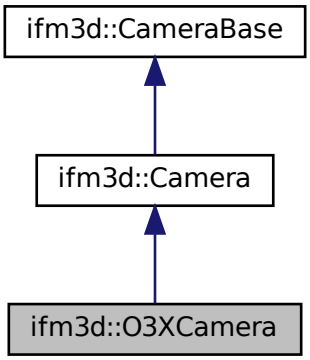
3.15 ifm3d::O3XCamera Class Reference

`#include <camera_o3x.h>`

Inheritance diagram for ifm3d::O3XCamera:



Collaboration diagram for ifm3d::O3XCamera:



Public Types

- using **Ptr** = std::shared_ptr< O3XCamera >

Public Member Functions

- **O3XCamera** (const std::string &ip=ifm3d::DEFAULT_IP, const std::uint16_t xmlrpc_port=ifm3d::DEFAULT↩ _XMLRPC_PORT, const std::string &password=ifm3d::DEFAULT_PASSWORD)
- **O3XCamera** (O3XCamera &&)=delete
- O3XCamera & **operator=** (O3XCamera &&)=delete
- **O3XCamera** (O3XCamera &)=delete
- O3XCamera & **operator=** (O3XCamera &)=delete
- device_family WhoAmI () override

Additional Inherited Members

3.15.1 Detailed Description

Camera specialization for O3X

3.15.2 Member Function Documentation

3.15.2.1 WhoAmI()

```
device_family ifm3d::O3XCamera::WhoAmI ( )  [override], [virtual]
```

This function can be used to retrieve the family of the connected device

Returns

the device family of the connected device.

Reimplemented from ifm3d::CameraBase.

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

- /home/usmasslo/gitlab/ifm3d/modules/camera/include/ifm3d/camera/camera_o3x.h

3.16 ifm3d::point< T, n > Struct Template Reference

Struct for 3D space point.

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

Public Types

- using **value_type** = T

Public Attributes

- T **val** [n]

3.16.1 Detailed Description

template<typename T, int n>
struct ifm3d::point< T, n >

Struct for 3D space point.

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

- /home/usmasslo/gitlab/ifm3d/modules/stlimage/include/ifm3d/stlimage/image.h

3.17 ifm3d::PortInfo Struct Reference

Public Attributes

- std::string **port**
- uint16_t **pcic_port**
- std::string **type**

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

- /home/usmasslo/gitlab/ifm3d/modules/camera/include/ifm3d/camera/camera_o3r.h

3.18 ifm3d::SemVer Struct Reference

Public Member Functions

- **SemVer** (size_t major, size_t minor, size_t patch, const std::optional< std::string > prerelease=std::nullopt, const std::optional< std::string > build_meta=std::nullopt)
- constexpr bool **operator<** (const SemVer &rhs) const
- constexpr bool **operator==** (const SemVer &rhs) const
- constexpr bool **operator!=** (const SemVer &rhs) const
- constexpr bool **operator>=** (const SemVer &rhs) const
- constexpr bool **operator>** (const SemVer &rhs) const
- constexpr bool **operator<=** (const SemVer &rhs) const

Static Public Member Functions

- static std::optional< SemVer > **Parse** (const std::string &version_string)

Public Attributes

- const size_t **major_num**
- const size_t **minor_num**
- const size_t **patch_num**
- const std::optional< std::string > **prerelease**
- const std::optional< std::string > **build_meta**

Friends

- std::ostream & **operator**<< (std::ostream &os, const SemVer &version)

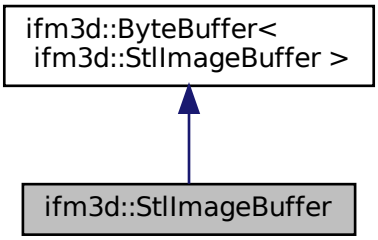
The documentation for this struct was generated from the following file:

- /home/usmasslo/gitlab/ifm3d/modules/camera/include/ifm3d/camera/semver.h

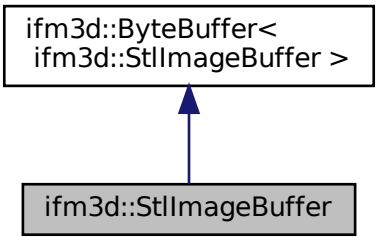
3.19 ifm3d::StlImageBuffer Class Reference

`#include <stl_image_buffer.h>`

Inheritance diagram for ifm3d::StlImageBuffer:



Collaboration diagram for ifm3d::StlImageBuffer:



Public Types

- using **Ptr** = std::shared_ptr< StlImageBuffer >

Public Member Functions

- StlImageBuffer ()
- ~StlImageBuffer () override
- **StlImageBuffer** (StlImageBuffer &&)
- StlImageBuffer & **operator=** (StlImageBuffer &&)
- **StlImageBuffer** (const StlImageBuffer &src_buff)
- StlImageBuffer & **operator=** (const StlImageBuffer &src_buff)
- ifm3d::Image DistanceImage ()
- ifm3d::Image UnitVectors ()
- ifm3d::Image GrayImage ()
- ifm3d::Image AmplitudeImage ()
- ifm3d::Image RawAmplitudeImage ()
- ifm3d::Image ConfidenceImage ()
- ifm3d::Image XYZImage ()
- ifm3d::Image JPEGImage ()
- ifm3d::Image DistanceNoiseImage ()

Protected Member Functions

- template<typename T >
  void ImCreate (ifm3d::image_chunk im, std::uint32_t fmt, std::size_t idx, std::uint32_t width, std::uint32_t height, int nchan, std::uint32_t npts, const std::vector< std::uint8_t > &bytes)
- template<typename T >
  void CloudCreate (std::uint32_t fmt, std::size_t xidx, std::size_t yidx, std::size_t zidx, std::uint32_t width, std::uint32_t height, std::uint32_t npts, const std::vector< std::uint8_t > &bytes)

Friends

- class **ifm3d::ByteBuffer**< **ifm3d::StlImageBuffer** >

Additional Inherited Members

3.19.1 Detailed Description

The StlImageBuffer class is a composite data structure used to hold time-synchronized images from the sensor. It organizes a single byte buffer read from the sensor into its component parts.

This class is not thread safe

3.19.2 Constructor & Destructor Documentation

3.19.2.1 StlImageBuffer()

```
ifm3d::StlImageBuffer::StlImageBuffer ( )
```

Allocates space for the individual component images.

3.19.2.2 ~StlImageBuffer()

```
ifm3d::StlImageBuffer::~StlImageBuffer ( )  [override]
```

RAII deallocations

3.19.3 Member Function Documentation

3.19.3.1 AmplitudeImage()

```
ifm3d::Image ifm3d::StlImageBuffer::AmplitudeImage ( )
```

Accessor for the normalized amplitude image

3.19.3.2 CloudCreate()

```
template<typename T >
void ifm3d::StlImageBuffer::CloudCreate (
            std::uint32_t fmt,
            std::size_t xidx,
            std::size_t yidx,
            std::size_t zidx,
            std::uint32_t width,
            std::uint32_t height,
            std::uint32_t npts,
            const std::vector< std::uint8_t > & bytes )  [inline], [protected]
```

Hook called by the base class to populate the point cloud containers.

3.19.3.3 ConfidenceImage()

```
ifm3d::Image ifm3d::StlImageBuffer::ConfidenceImage ( )
```

Accessor for the confidence image

3.19.3.4 DistanceImage()

```
ifm3d::Image ifm3d::StlImageBuffer::DistanceImage ( )
```

Accessor for the wrapped radial distance image

3.19.3.5 DistanceNoiseImage()

```
ifm3d::Image ifm3d::StlImageBuffer::DistanceNoiseImage ( )
```

Accessor for the OpenCV encoding of the DistanceNoiseImage

3.19.3.6 GrayImage()

```
ifm3d::Image ifm3d::StlImageBuffer::GrayImage ( )
```

Accessor the the wrapped ambient light image

3.19.3.7 ImCreate()

```
template<typename T >
void ifm3d::StlImageBuffer::ImCreate (
            ifm3d::image_chunk im,
            std::uint32_t fmt,
            std::size_t idx,
            std::uint32_t width,
            std::uint32_t height,
            int nchan,
            std::uint32_t npts,
            const std::vector< std::uint8_t > & bytes )  [inline], [protected]
```

Hook called by the base class to populate the image containers.

3.19.3.8 JPEGImage()

```
ifm3d::Image ifm3d::StlImageBuffer::JPEGImage ( )
```

Accessor for the jpeg encoded 2D image

3.19.3.9 RawAmplitudeImage()

```
ifm3d::Image ifm3d::StlImageBuffer::RawAmplitudeImage ( )
```

Accessor for the raw amplitude image

3.19.3.10 UnitVectors()

```
ifm3d::Image ifm3d::StlImageBuffer::UnitVectors ( )
```

Accessor for the wrapped unit vectors

3.19.3.11 XYZImage()

```
ifm3d::Image ifm3d::StlImageBuffer::XYZImage ( )
```

Accessor for the OpenCV encoding of the point cloud

3-channel image of spatial planes X, Y, Z

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

- /home/usmasslo/gitlab/ifm3d/modules/stlimage/include/ifm3d/stlimage/stl_image_buffer.h

3.20 ifm3d::SWUpdater Class Reference

Public Types

- using **Ptr** = std::shared_ptr< SWUpdater >
- using FlashStatusCb = std::function< void(float, const std::string &)>

Public Member Functions

- SWUpdater (ifm3d::CameraBase::Ptr cam, const ifm3d::SWUpdater::FlashStatusCb &cb={}, const std::uint16_t swupdate_recovery_port=ifm3d::SWUPDATER_RECOVERY_PORT)
- **SWUpdater** (SWUpdater &&)=delete
- SWUpdater & **operator=** (SWUpdater &&)=delete
- **SWUpdater** (SWUpdater &)=delete
- SWUpdater & **operator=** (const SWUpdater &)=delete
- void RebootToRecovery ()
- bool WaitForRecovery (long timeout_millis=0)
- void RebootToProductive ()
- bool WaitForProductive (long timeout_millis=0)
- bool FlashFirmware (const std::vector< std::uint8_t > &bytes, long timeout_millis=0)

3.20.1 Member Typedef Documentation

3.20.1.1 FlashStatusCb

```
using ifm3d::SWUpdater::FlashStatusCb = std::function<void(float, const std::string&)>
```

Signature for user callback to receive status information about firmware flashing.

The first parameter is a percentage (0.0-1.0) indicating the status of uploading the file to the device.

The second parameter is a status message from the camera during install.

3.20.2 Constructor & Destructor Documentation

3.20.2.1 SWUpdater()

```
ifm3d::SWUpdater::SWUpdater (
            ifm3d::CameraBase::Ptr cam,
            const ifm3d::SWUpdater::FlashStatusCb & cb = {},
            const std::uint16_t swupdate_recovery_port = ifm3d::SWUPDATER_RECOVERY_PORT )
```

Ctor

Parameters

cam	Camera object to manipulate
cb	Optional user-defined callback to handle status updates
swupdate_recovery_port	swupate recovery port for the device

3.20.3 Member Function Documentation

3.20.3.1 FlashFirmware()

```
bool ifm3d::SWUpdater::FlashFirmware (
            const std::vector< std::uint8_t > & bytes,
            long timeout_millis = 0 )
```

Uploads a firmware image to the camera's recovery system. Assumes device has already been rebooted to recovery mode.

Parameters

in	*bytes*	The firmware image data to flash to the camera.
in	*timeout_millis*	Timeout in millis to wait for the firmware upload to complete. If `timeout_millis` is set to 0, this function will block indefinitely.

NOTE: Firmware uploading and flashing typically takes several minutes. The blocking version of the API (timeout_millis = 0) is recommended in most cases. If a timeout is truly required, it is recommended to use a value of at least 300000 (5 minutes).

Exceptions

ifm3d::error_t	on error

3.20.3.2 RebootToProductive()

```
void ifm3d::SWUpdater::RebootToProductive ( )
```

Reboots the camera from recovery to productive. The function returns immediately, but the reboot process takes some time. The function `WaitForProductive` may be used to poll for completion.

Exceptions

ifm3d::error_t	on error

3.20.3.3 RebootToRecovery()

```
void ifm3d::SWUpdater::RebootToRecovery ( )
```

Reboots the camera from productive to recovery. The function returns immediately, but the reboot process takes some time. The function `WaitForRecovery` may be used to poll for completion.

Exceptions

ifm3d::error_t	on error

3.20.3.4 WaitForProductive()

```
bool ifm3d::SWUpdater::WaitForProductive (
            long timeout_millis = 0 )
```

Polls on status of the camera, waiting for it to present in productive mode. Should be used following a call to RebootToProductive().

Parameters

in	*timeout_millis*	Timeout in millis to wait for the device to become available. If `timeout_millis` is set to 0, this function will block indefinitely. If `timeout_millis` is set to -1, this function will check once and return immediately.

Returns

true if the device became available w/in `timeout_millis`, false otherwise.

Exceptions

ifm3d::error_t	on error

3.20.3.5 WaitForRecovery()

```
bool ifm3d::SWUpdater::WaitForRecovery (
            long timeout_millis = 0 )
```

Polls on status of the camera, waiting for it to present in recovery mode. Should be used following a call to RebootToRecovery().

Parameters

in	*timeout_millis*	Timeout in millis to wait for the recovery system to become available. If `timeout_millis` is set to 0, this function will block indefinitely. If `timeout_millis` is set to -1, this function will check once and return immediately.

Returns

true if the recovery system became available w/in `timeout_millis`, false otherwise.

Exceptions

ifm3d::error_t	on error

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

- /home/usmasslo/gitlab/ifm3d/modules/swupdater/include/ifm3d/swupdater/swupdater.h

Index