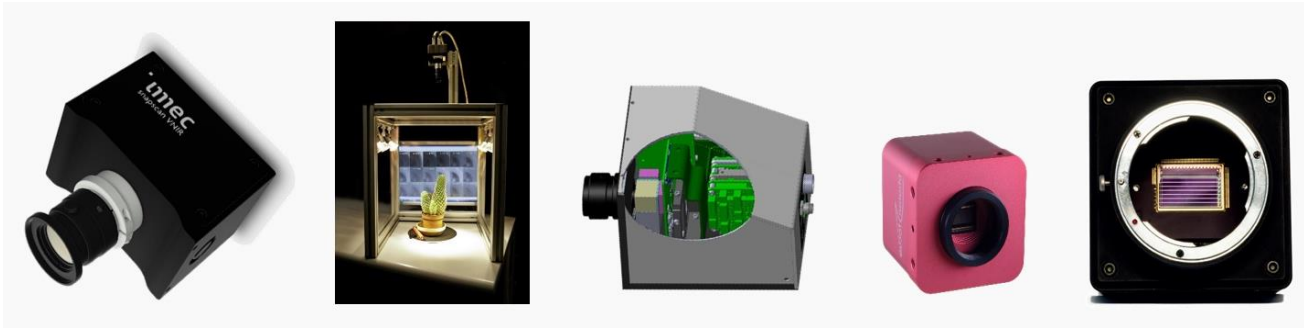
imec

Perform Hyperspectral Imaging using our user-friendly API(s)

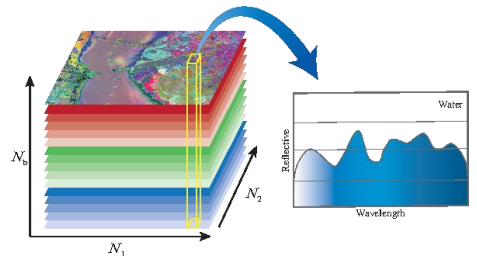
Introduction

Imec HSI is a whole habitat of cameras, tools and software

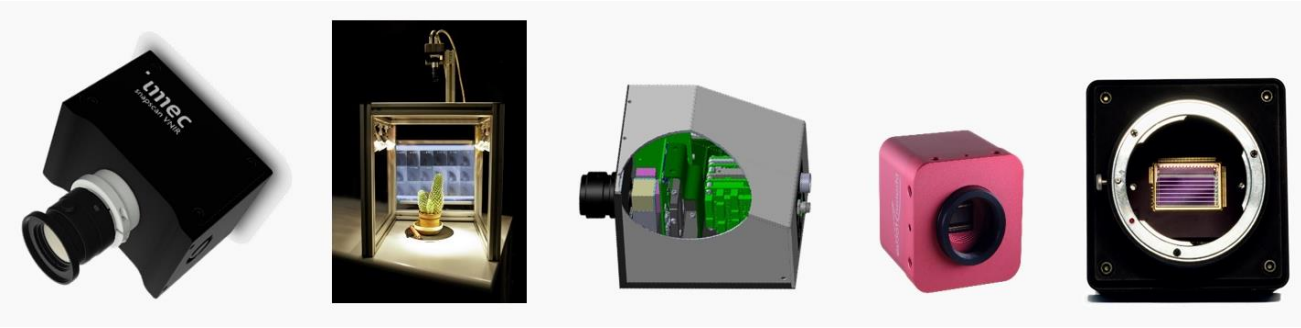
We build Hyperspectral Cameras ...



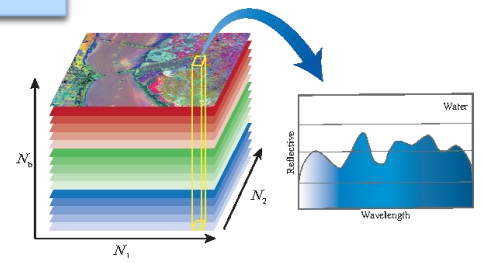
... for you to obtain high-quality Hyperspectral Cubes



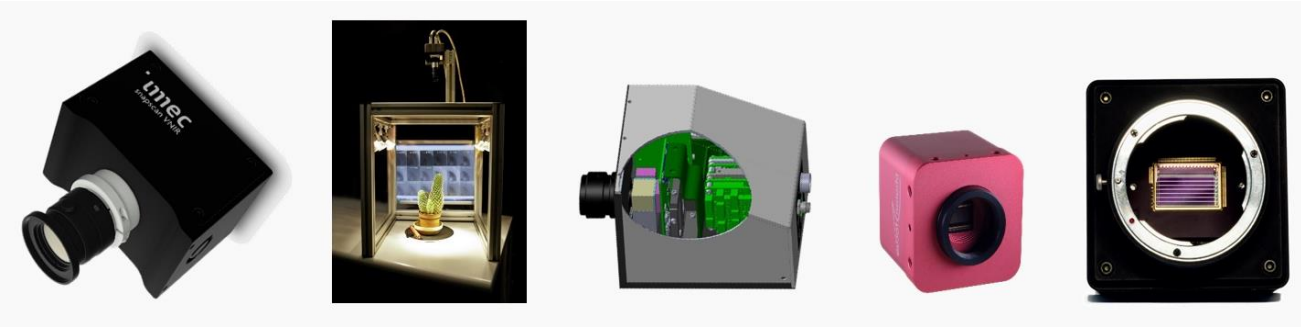
Imec HSI is a whole habitat of cameras, tools and software

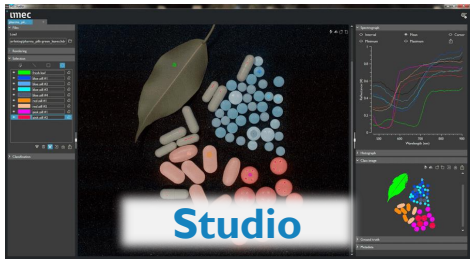


We also provide Software to operate our different types of imec HSI Cameras

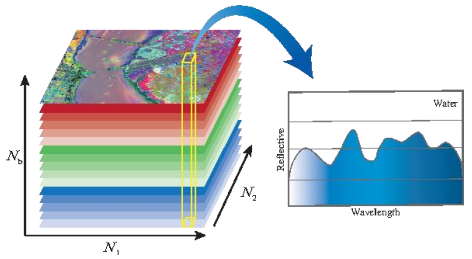


Imec HSI is a whole habitat of cameras, tools and software

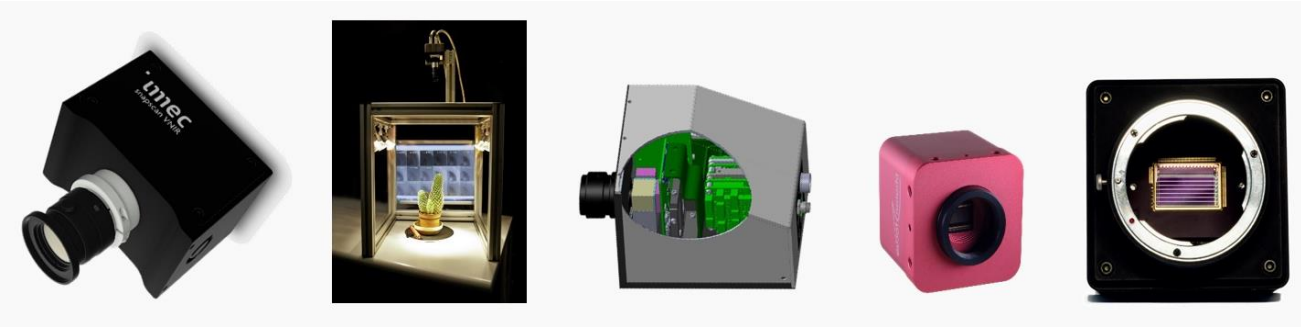




... and we provide you with a Hyperspectral Cube Data Visualization and Analysis Tool



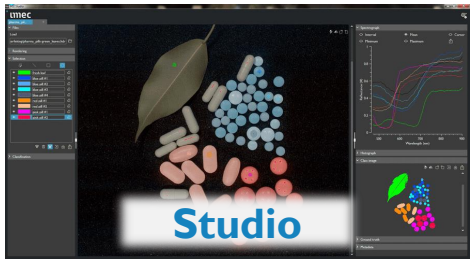
Imec HSI is a whole habitat of cameras, tools and software

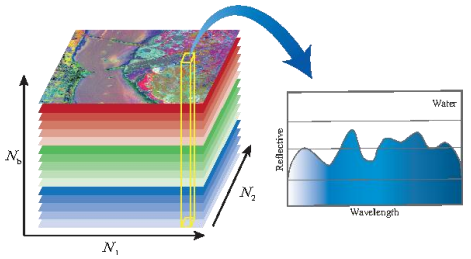


API Camera ⊕ API Mosaic

API SnapScan

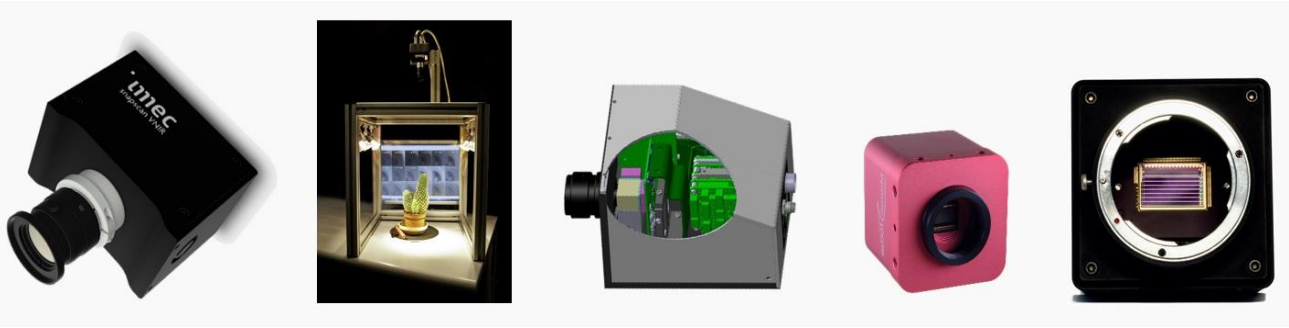
Now we also provide APIs for all these components





The API is in C-language with python-wrappers

Imec HSI is a whole habitat of cameras, tools and software



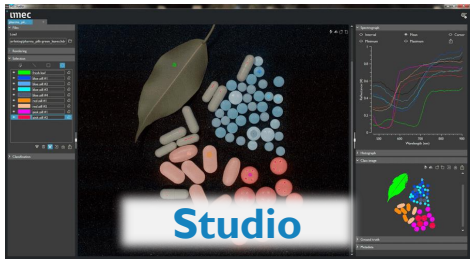
Linked by a common API to ensure inter-operability between all APIs / Tools.

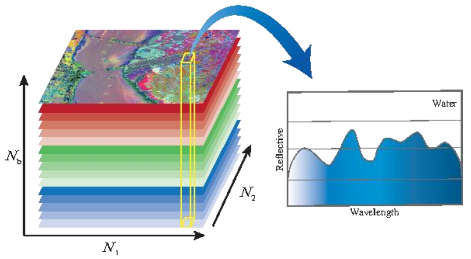
API Camera ⊕ API Mosaic

API SnapScan

API common

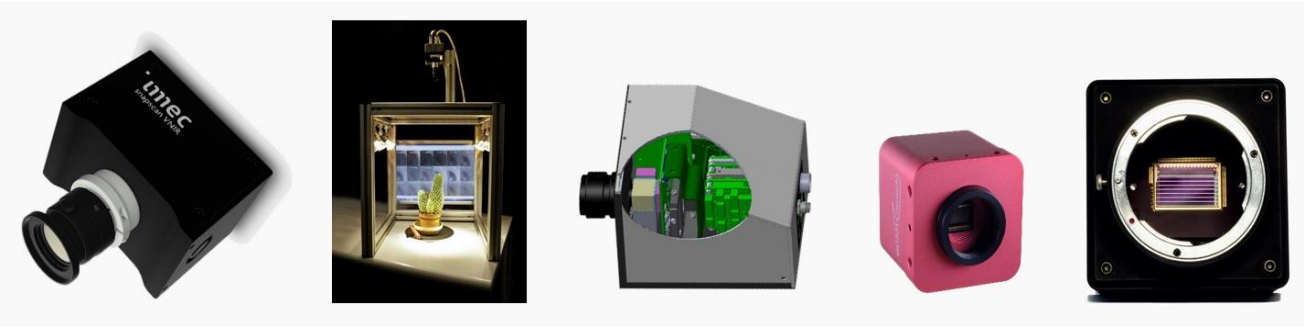
- *Data-types: Frame, Cube, Spectrum, ...*
- *Load-/Save functions*
- *Allocate/Deallocate functions*





The COMMON - API

First we focus on the Common API : one API to rule them all ...

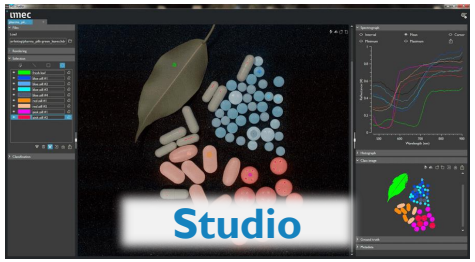


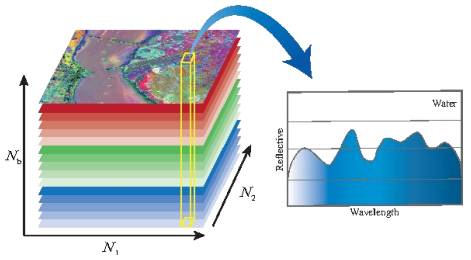
API Camera ⊕ API Mosaic

API SnapScan

API common

- *Data-types: Frame, Cube, Spectrum, ...*
- *Load-/Save functions*
- *Allocate/Deallocate functions*



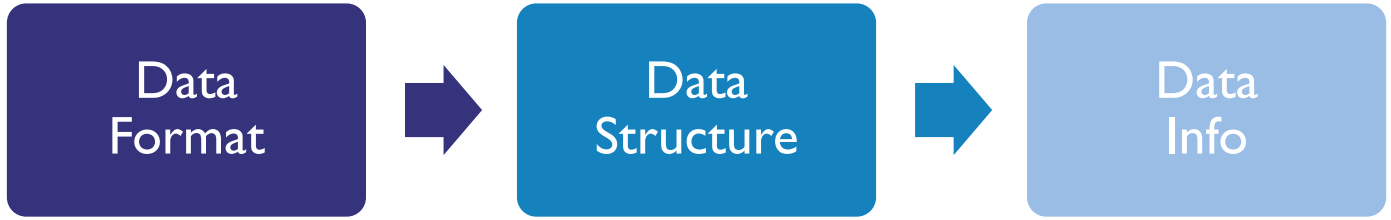


imec

Common - API

Making all APIs inter-operable

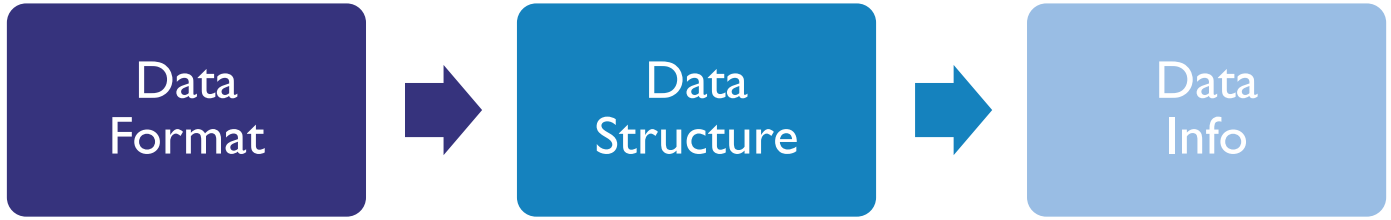
- Our Data-Type Principle →
  - Every datatype consist of
    - One Data format
    - Data Values (structure)
    - One Data Info

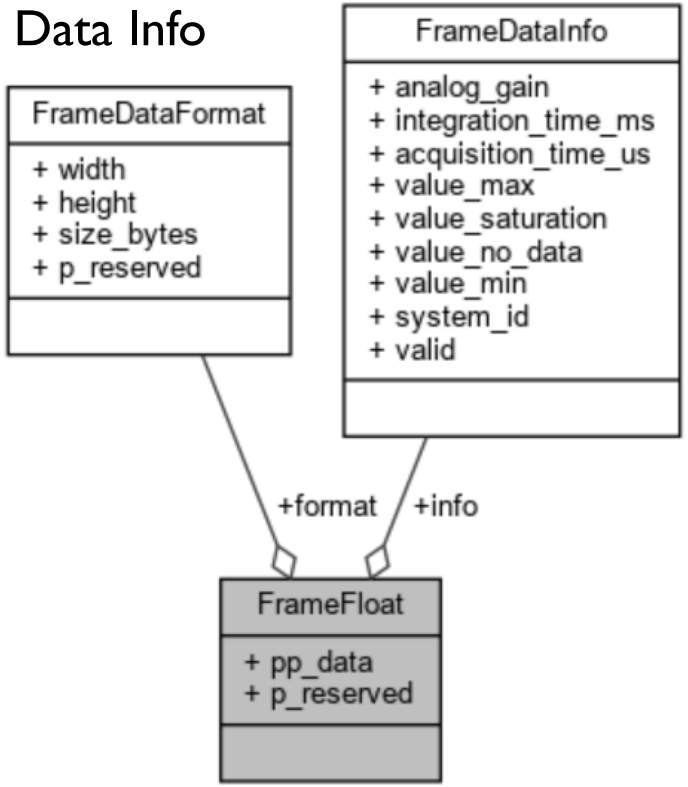


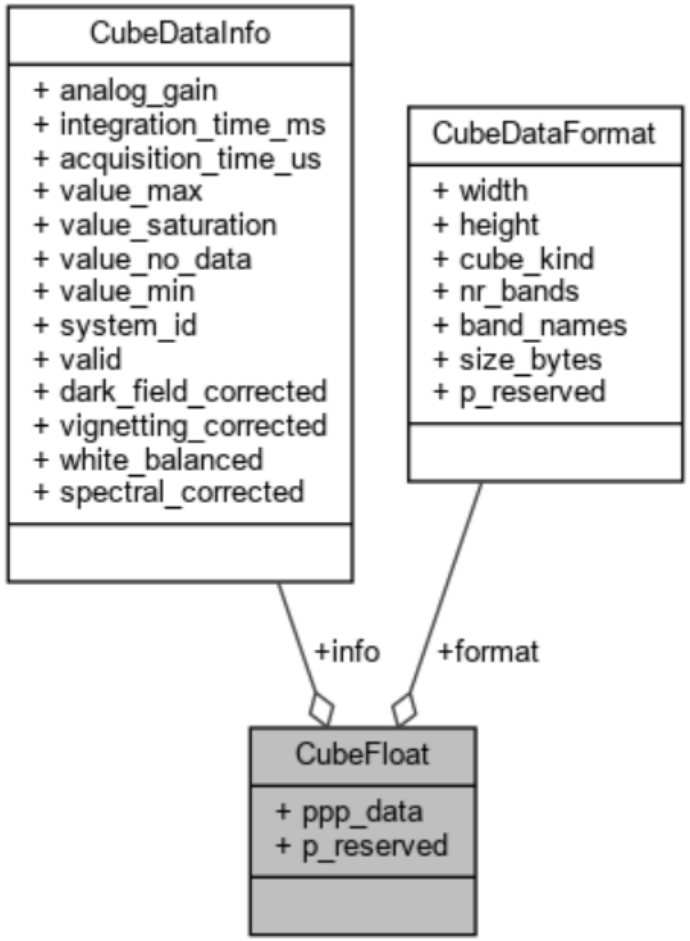
Common - API

Making all APIs inter-operable

- Our Data-Type Principle →
  - Every datatype consist of
    - One Data format
    - Data Values (structure)
    - One Data Info

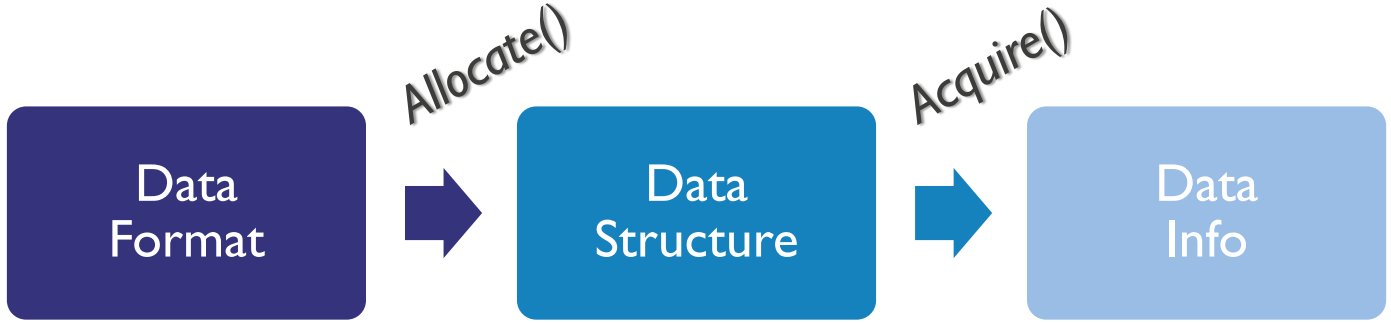






Common - API

Making all APIs inter-operable



- Our Data-Type Principle →
  - Every datatype consist of
    - One Data format
    - Data Values (structure)
    - One Data Info
  - The (internal) memory allocation is done *based* on the Data-format
    - data-format is always required *first* (!)
    - memory allocations are done inside the data-structure
  - Data-info is populated *at the end* of an acquisition
    - Data Acquisition => data values are stored in allocated space
    - Data-info fields are populated => acquisition time, exposure time, min/max values, ...

Common – API – FrameFloat / CubeFloat

To Allocate() is also to deallocate()

◆ AllocateFrame()

HSI_API **HSI_RETURN** AllocateFrame (**FrameFloat** * o_p_frame,
FrameDataFormat i_data_format
)

Allocates a data frame from the given data format.

Parameters

[out] **o_p_frame** Pointer to the frame structure in which to allocate the data.

[in] **i_data_format** Data format of the data to allocate.

Returns

ReturnValue indicating the result of the call.

◆ AllocateCube()

HSI_API **HSI_RETURN** AllocateCube (**CubeFloat** * o_p_cube,
CubeDataFormat i_data_format
)

Allocates a data cube from the given data format.

Parameters

[out] **o_p_cube** Pointer to the cube structure in which to allocate the data.

[in] **i_data_format** Data format of the data to allocate.

Returns

ReturnValue indicating the result of the call.

◆ DeallocateFrame()

HSI_API **HSI_RETURN** DeallocateFrame (**FrameFloat** * o_p_frame)

Deallocates the memory behind the given frame. Does not delete the pointer to the frame.

Parameters

[out] **o_p_frame** The frame of which to deallocate the data.

Returns

ReturnValue indicating the result of the call.

◆ DeallocateCube()

HSI_API **HSI_RETURN** DeallocateCube (**CubeFloat** * o_p_cube)

Deallocates the memory behind the given cube. Does not delete the pointer to the cube.

Parameters

[out] **o_p_cube** The cube of which to deallocate the data.

Returns

ReturnValue indicating the result of the call.

Common – API – FrameFloat / CubeFloat

Loading and Saving support is build in

◆ LoadFrame()

```
HSI_API HSI_RETURN LoadFrame ( FrameFloat *     o_p_frame,
                               wchar_t const *  i_in_file_path
                             )
```

Loads a data frame from disk.

Parameters

- [in] **o_p_frame** The frame that will contain the loaded data.
- [in] **i_in_file_path** The system file's path to the desired frame.

Returns

ReturnValue indicating the result of the call.

◆ LoadCube()

```
HSI_API HSI_RETURN LoadCube ( CubeFloat *     o_p_cube,
                              wchar_t const *  i_in_file_path
                            )
```

Loads a data cube from disk.

Parameters

- [in] **o_p_cube** The cube that will contain the loaded data.
- [in] **i_in_file_path** The system file's path to the desired cube. Path must refer to the header and the raw must be standing aside.

Returns

ReturnValue indicating the result of the call.

◆ SaveFrame()

```
HSI_API HSI_RETURN SaveFrame ( FrameFloat      i_frame,
                               wchar_t const *  i_out_dir_path,
                               wchar_t const *  i_out_file_name,
                               FileFormat       i_out_file_format
                             )
```

Stores a data frame to disk in the standard TIFF file format.

Parameters

- [in] **i_frame** The frame data that must be stored to disk.
- [in] **i_out_dir_path** Path to a directory in which the data must be stored. May be both relative or absolute. Non-ex
- [in] **i_out_file_name** Filename for the output cube without extension.
- [in] **i_out_file_format** File format in which the image must be stored on disk.

◆ SaveCube()

```
HSI_API HSI_RETURN SaveCube ( CubeFloat      i_cube,
                              wchar_t const *  i_out_dir_path,
                              wchar_t const *  i_out_file_name,
                              FileFormat       i_out_file_format
                            )
```

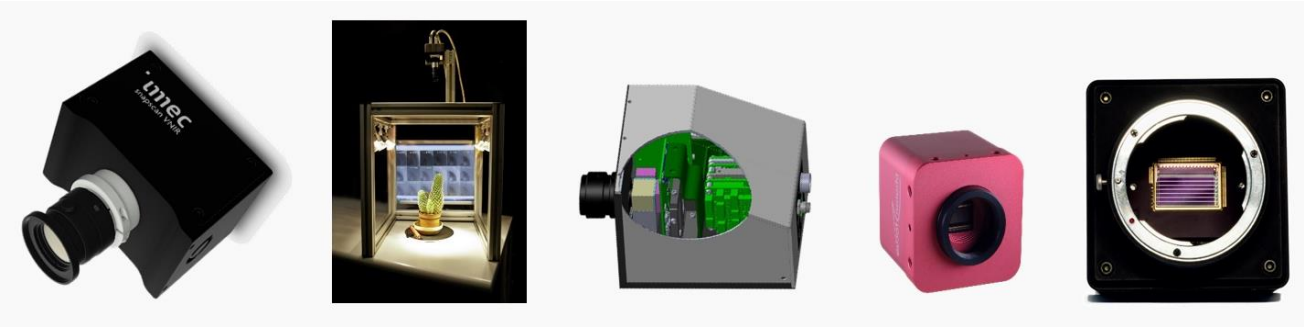
Stores a data cube to disk in the standard ENVI file format.

Parameters

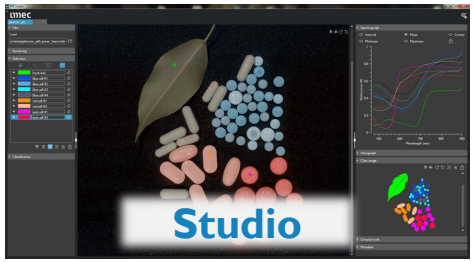
- [in] **i_cube** The cube data that must be stored to disk.
- [in] **i_out_dir_path** Path to a directory in which the data must be stored. May be both relative or absolute. Non-existing paths will be created recursively.
- [in] **i_out_file_name** Filename for the output cube without extension. Will be used to name both the ENVI header and ENVI data file.
- [in] **i_out_file_format** File format in which the image must be stored on disk.

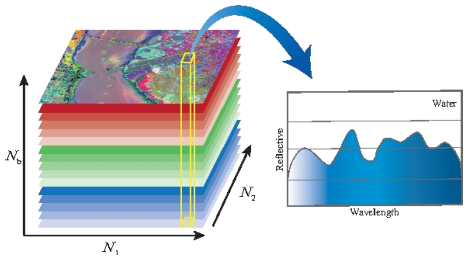
The CAMERA - API

Camera API is used to acquire RAW data from a HSI Camera

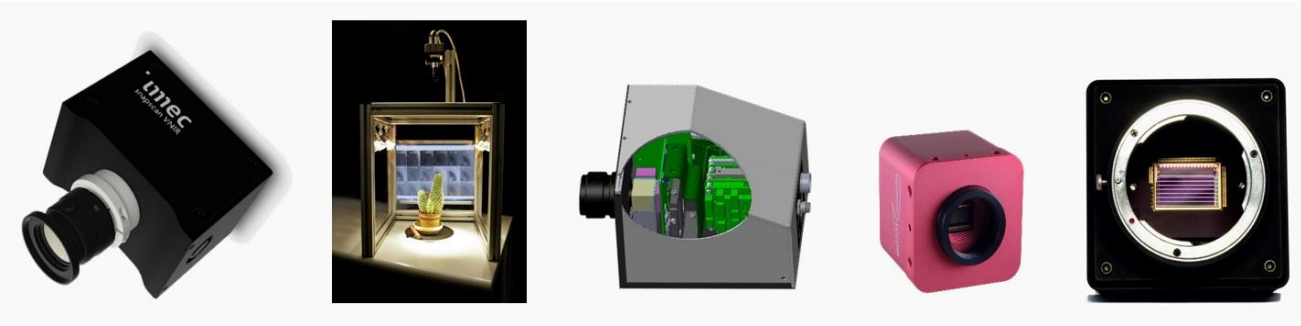


- *Data-types: Frame, Cube, Spectrum, ...*
- *Load-/Save functions*
- *Allocate/Deallocate functions*





Camera API is used to acquire RAW data from a HSI Camera

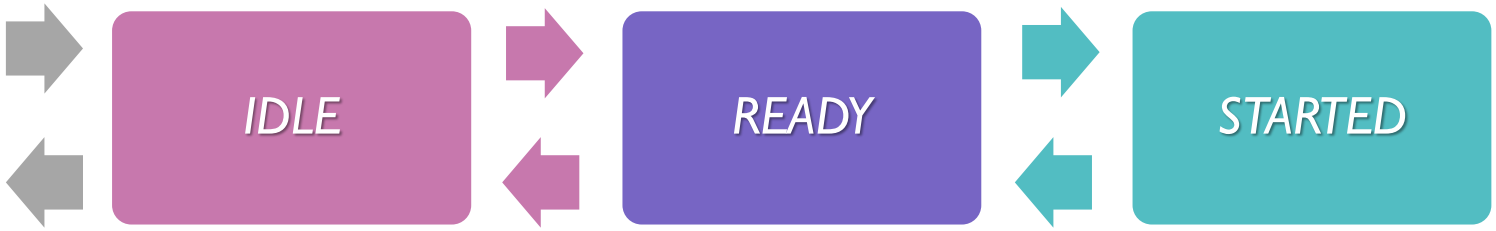


The Camera API will:
- Communicate with the camera device(s)
- Acquire Data from the camera

The (Mosaic) Camera API

Respect for Internal States

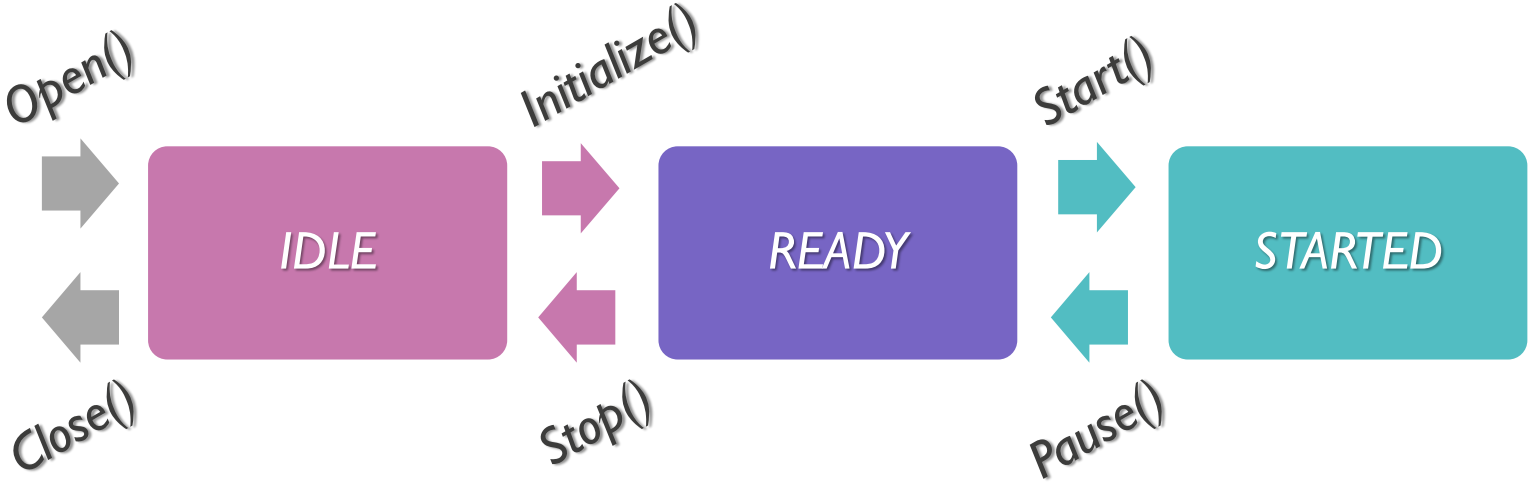
- A Camera has 3 states :



The (Mosaic) Camera API

Respect for Internal States

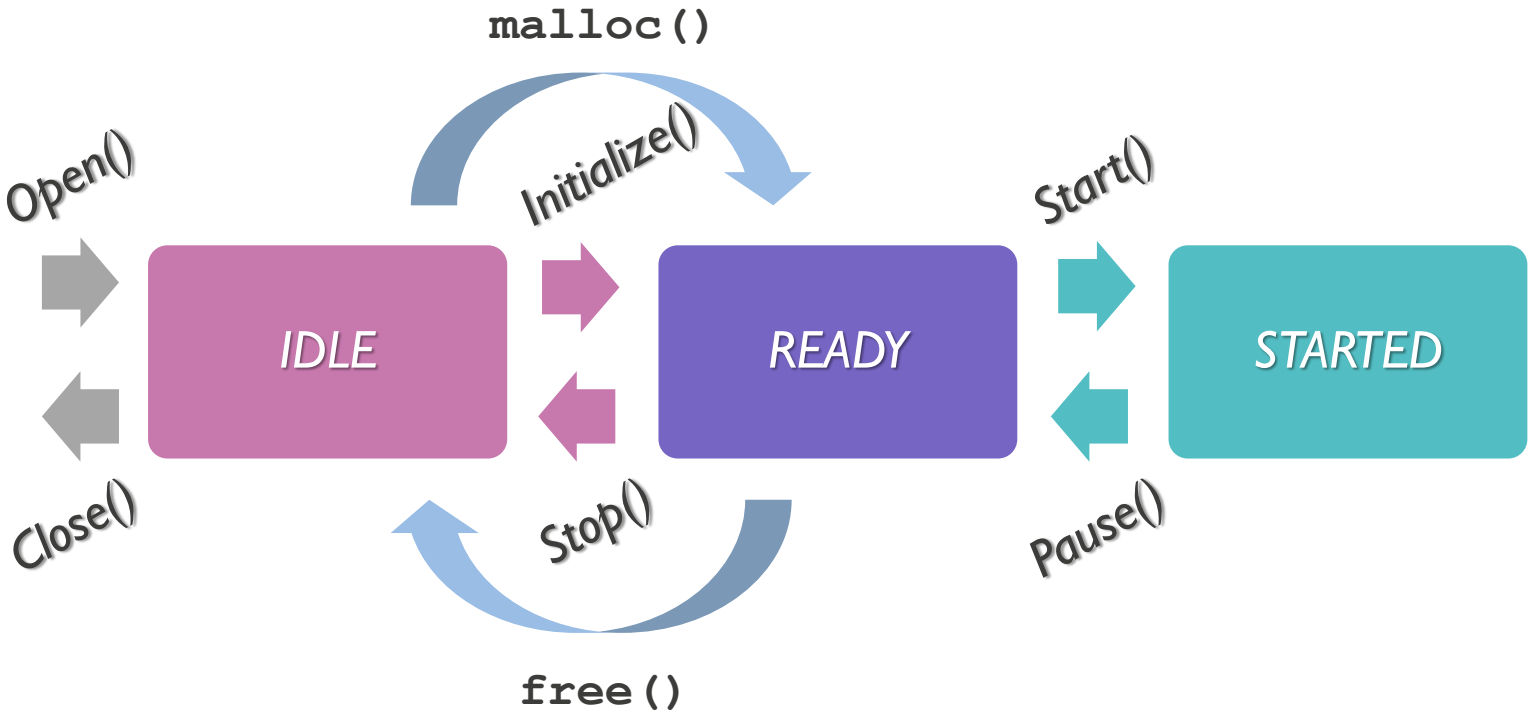
- A Camera has 3 states :
- Calls can modify state



The (Mosaic) Camera API

Respect for Internal States

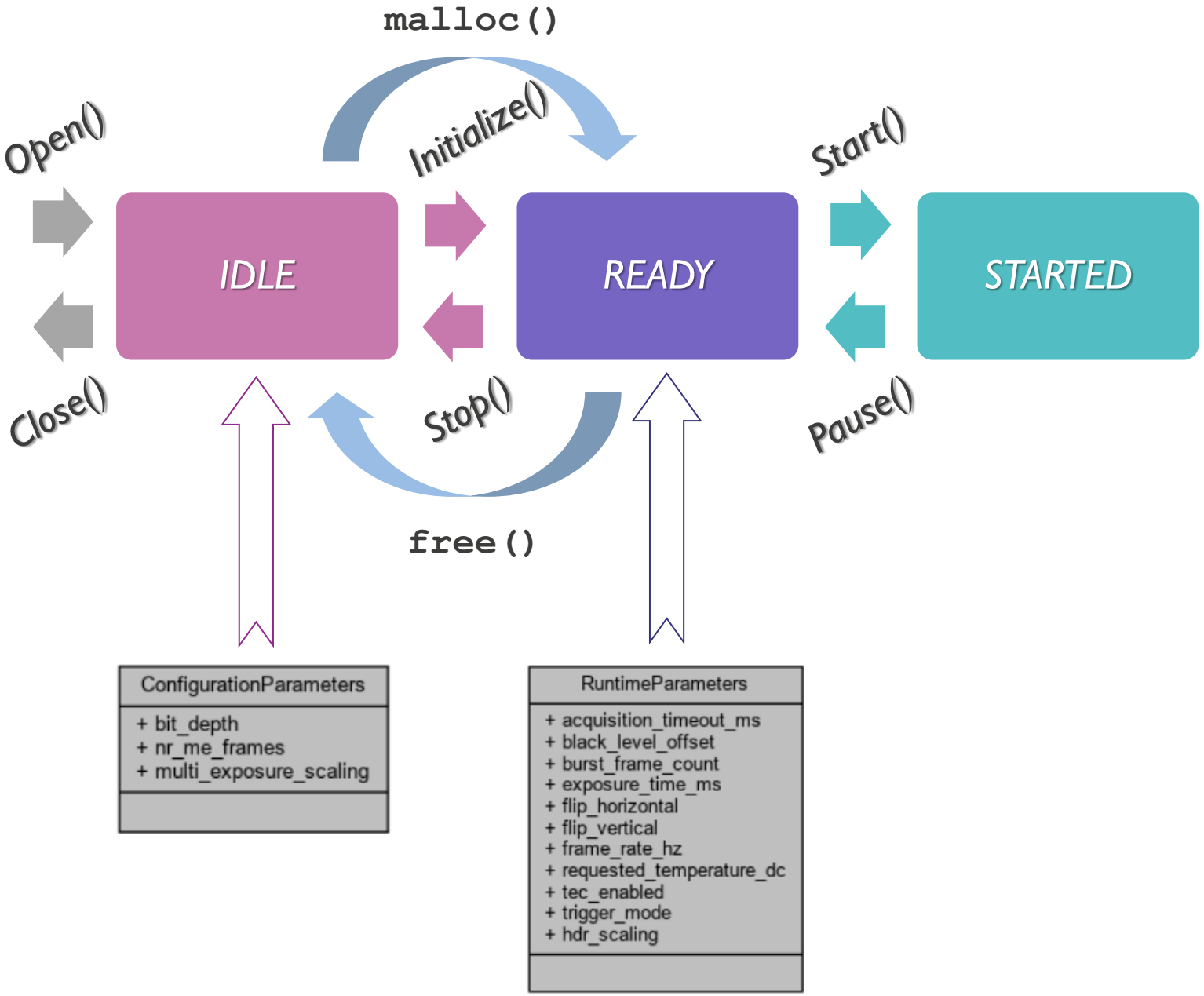
- A Camera has 3 states :
- Calls can modify state
- Memory Allocations are function of state



The (Mosaic) Camera API

Respect for Internal States

- A Camera has 3 states :
- Calls can modify state
- Memory Allocations are function of state
- Different Parameter set in function of state



CAMERA – API
The EXAMPLE

The (Mosaic) Camera API

Let's build up an example

- Let's include the API header file.
- All API functions return a result, indicating the success/failure of its actions.
- HSI has a build-in Logger tool. It takes a directory and a log-level as arguments.

```
#include "hsi_camera_api/hsi_camera_api.h"

    HSI_RETURN result = HSI_OK;

    result = InitializeLogger (L"./logs/", LV_DEBUG);
    print ("InitializeLogger", result);
```

The (Mosaic) Camera API

Let's build up an example

- The API will look for connected devices, and return a list of them, and the number of devices it found.

```
#include "hsi_camera_api/hsi_camera_api.h"

    HSI_RETURN result = HSI_OK;

    result = InitializeLogger (L"./logs/", LV_DEBUG);
    print ("InitializeLogger", result);

    CameraInfo camera_info = {};
    int nr_devices;
    result = EnumerateConnectedDevices (&camera_info, &nr_devices, 100, EM_IMEC);

    if (HSI_OK != result || 0 == nr_devices)
    {
        return -1;
    }
```

The (Mosaic) Camera API

Let's build up an example

- Assume there is only a single camera connected, and the API has found it, let's open the device.

```
#include "hsi_camera_api/hsi_camera_api.h"

    HSI_RETURN result = HSI_OK;

    result = InitializeLogger (L"./logs/", LV_DEBUG);
    print ("InitializeLogger", result);

    CameraInfo camera_info = {};
    int nr_devices;
    result = EnumerateConnectedDevices (&camera_info, &nr_devices, 100, EM_IMEC);

    HANDLE camera = {};
    result = OpenDevice (&camera, camera_info);

    if (HSI_OK != result)
    {
        return -1;
    }
```

The (Mosaic) Camera API

Let's build up an example

- Now we can iterate through the camera's internal states:

```
#include "hsi_camera_api/hsi_camera_api.h"

    HSI_RETURN result = HSI_OK;

    result = InitializeLogger (L"./logs/", LV_DEBUG);
    print ("InitializeLogger", result);

    CameraInfo camera_info = {};
    int nr_devices;
    result = EnumerateConnectedDevices (&camera_info, &nr_devices, 100, EM_IMEC);

    HANDLE camera = {};
    result = OpenDevice (&camera, camera_info);

    result = Initialize (camera);

    result = Start (camera);

    result = Pause (camera);

    result = Stop (camera);

    CloseDevice (&camera);
```

The (Mosaic) Camera API

Let's build up an example

- Before Initialize() we can get/set the configuration parameters if needed.
- After Initialize() we can get/set the runtime parameters if needed.

```
#include "hsi_camera_api/hsi_camera_api.h"

    ...

    HANDLE camera = {};
    result = OpenDevice (&camera, camera_info);

    ConfigurationParameters config_parameters = {};
    GetConfigurationParameters (camera, &config_parameters);
    config_parameters.bit_depth = 10;
    result = SetConfigurationParameters (camera, config_parameters);

    result = Initialize (camera);

    RuntimeParameters runtime_parameters = {};
    GetRuntimeParameters (camera, &runtime_parameters);
    runtime_parameters.frame_rate_hz = 25;
    runtime_parameters.exposure_time_ms = 5.0;
    runtime_parameters.trigger_mode = TM_NoTriggering;
    result = SetRuntimeParameters (camera, runtime_parameters);

    result = Start (camera);
    result = Pause (camera);
    result = Stop (camera);
    CloseDevice (&camera);
```

The (Mosaic) Camera API

Let's build up an example

- Ask the Camera which output-format it is configured for ...
- ... and use that to allocate a Frame(Float).
- And don't forget to de-allocate both !

```
#include "hsi_camera_api/hsi_camera_api.h"

    ...

    HANDLE camera = {};
    result = OpenDevice (&camera, camera_info);
    result = Initialize (camera);

    FrameDataFormat data_format = {};
    result = GetOutputFrameDataFormat (camera, &data_format);

    FrameFloat frame = {};
    result = AllocateFrame (&frame, data_format);

    result = Start (camera);









    result = Pause (camera);
    result = Stop (camera);

    DeallocateFrame (&frame);
    DeallocateFrameDataFormat (&data_format);
    CloseDevice (&camera);
```

The (Mosaic) Camera API

Let's build up an example

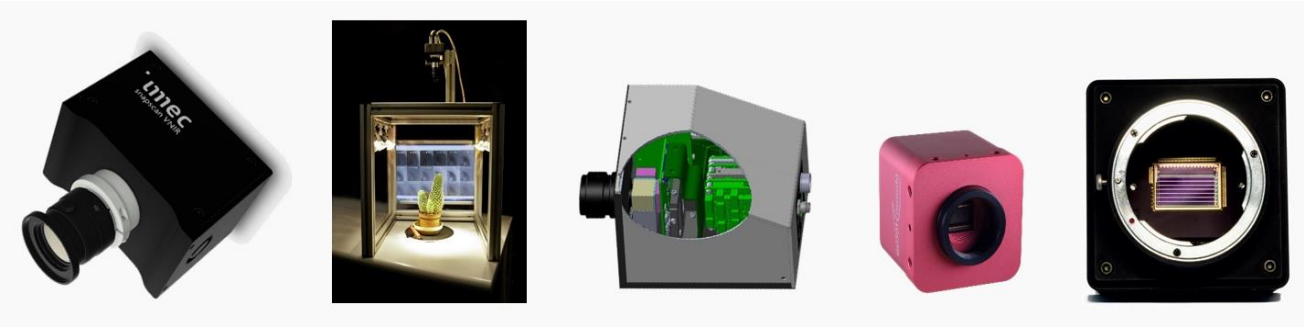
- Let's (software) trigger the camera ...
- ... and acquire a frame using the allocated buffer.

- And use the Save function to write it to disk.

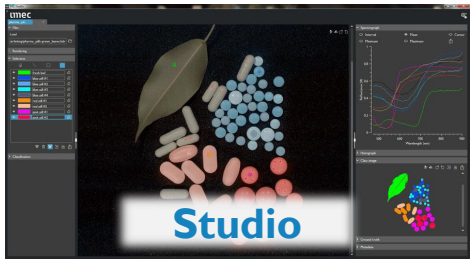
```
#include "hsi_camera_api/hsi_camera_api.h"

    ...

    HANDLE camera = {};
    result = OpenDevice (&camera, camera_info);
    result = Initialize (camera);

    FrameDataFormat data_format = {};
    result = GetOutputFrameDataFormat (camera, &data_format);

    FrameFloat frame = {};
    result = AllocateFrame (&frame, data_format);

    result = Start (camera);

    result = Trigger (camera);
    result = AcquireFrame (camera, &frame);

    result = SaveFrame (frame, L"./output/", L"frame0", FF_PNG);

    result = Pause (camera);
    result = Stop (camera);

    DeallocateFrame (&frame);
    DeallocateFrameDataFormat (&data_format);
    CloseDevice (&camera);
```

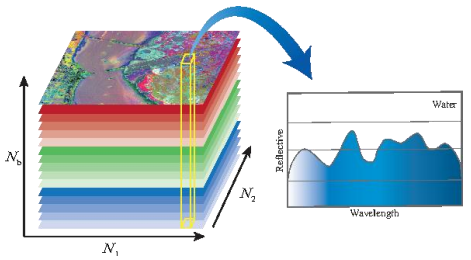
The MOSAIC – API

Mosaic API will convert RAW frames into HSI Cubes



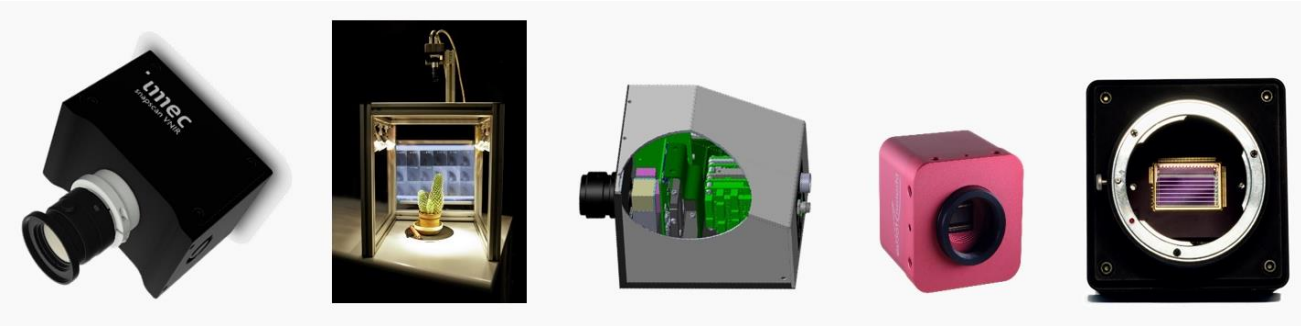
- *Data-types: Frame, Cube, Spectrum, ...*
- *Load-/Save functions*
- *Allocate/Deallocate functions*





imec

Mosaic API will convert RAW frames into HSI Cubes

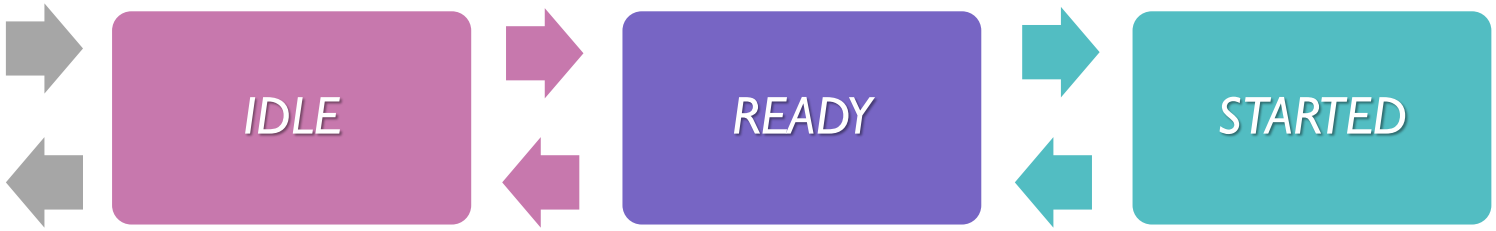


The Mosaic API can:
- Convert frames into Hyperspectral Cubes
- This based on a "context" (i.e. calibration file, ...)
- Extract spectral information from a cube

The Mosaic (processing-engine) API

Respect for Internal States

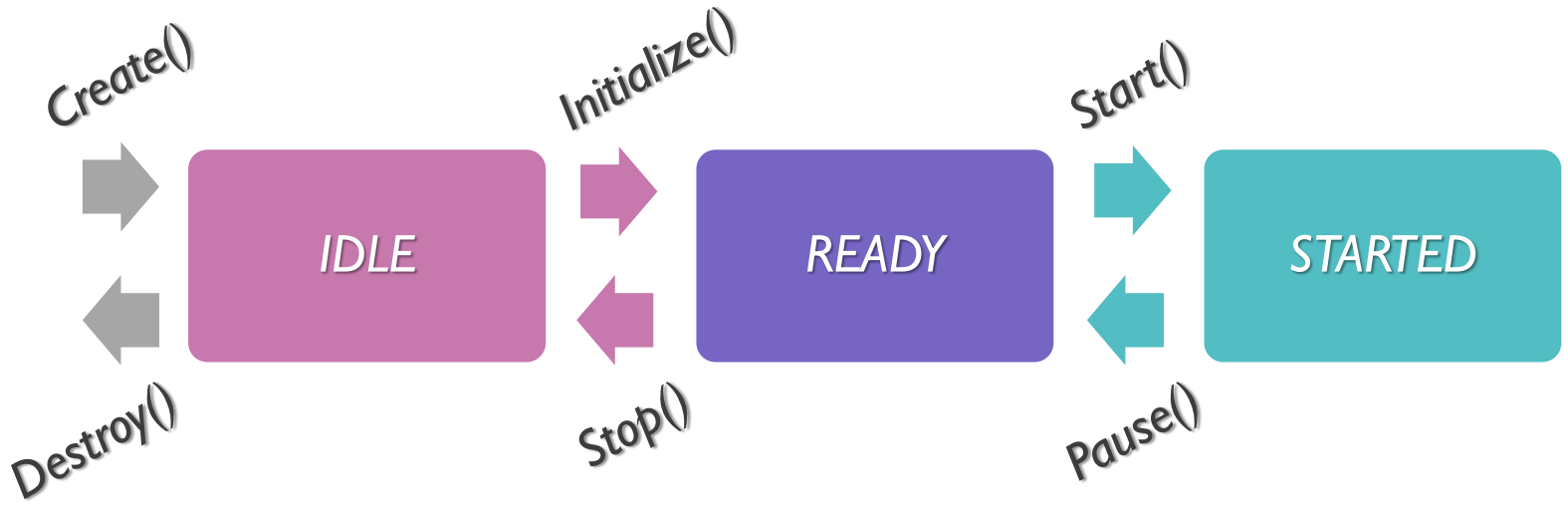
- A Pipeline has 3 states :



The Mosaic (processing-engine) API

Respect for Internal States

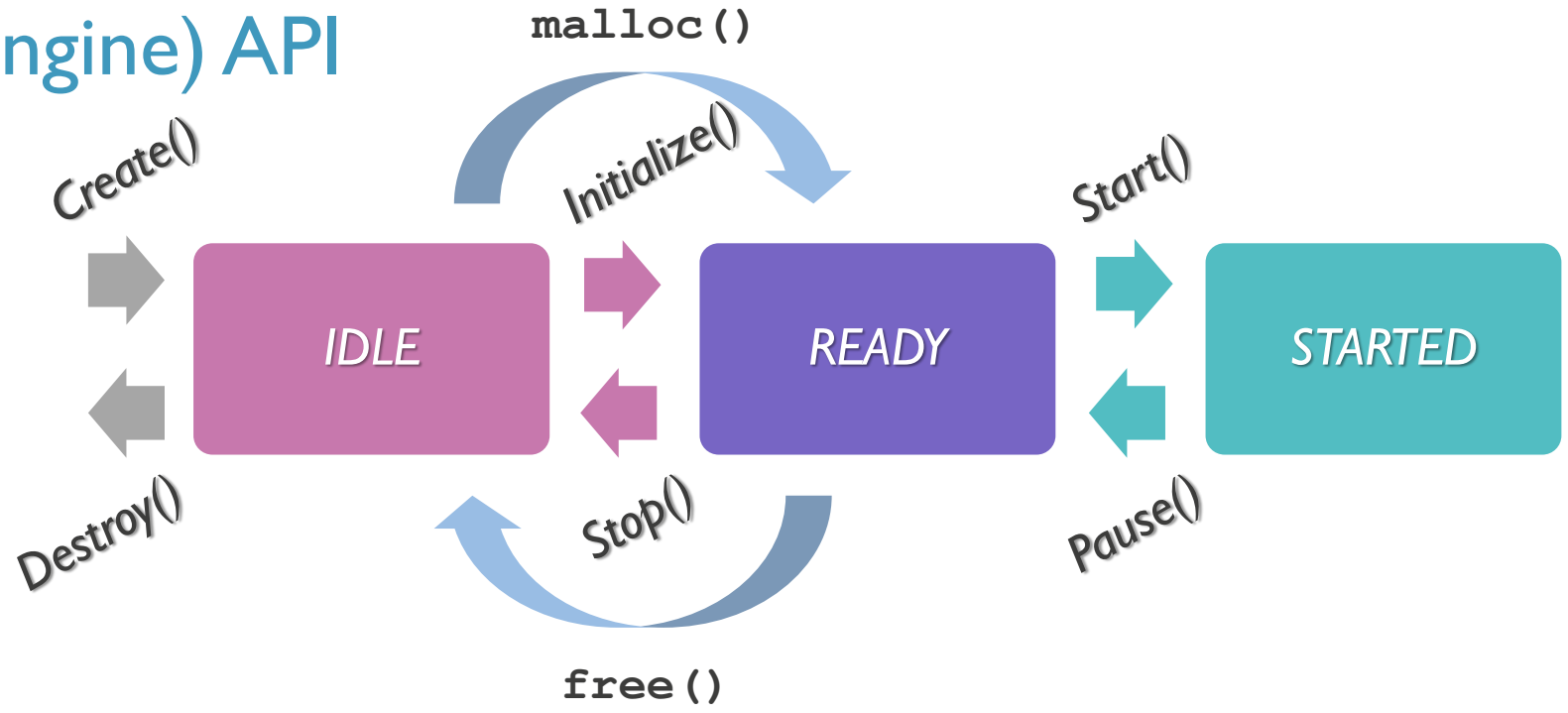
- A Pipeline has 3 states :
- API Calls can modify state



The Mosaic (processing-engine) API

Respect for Internal States

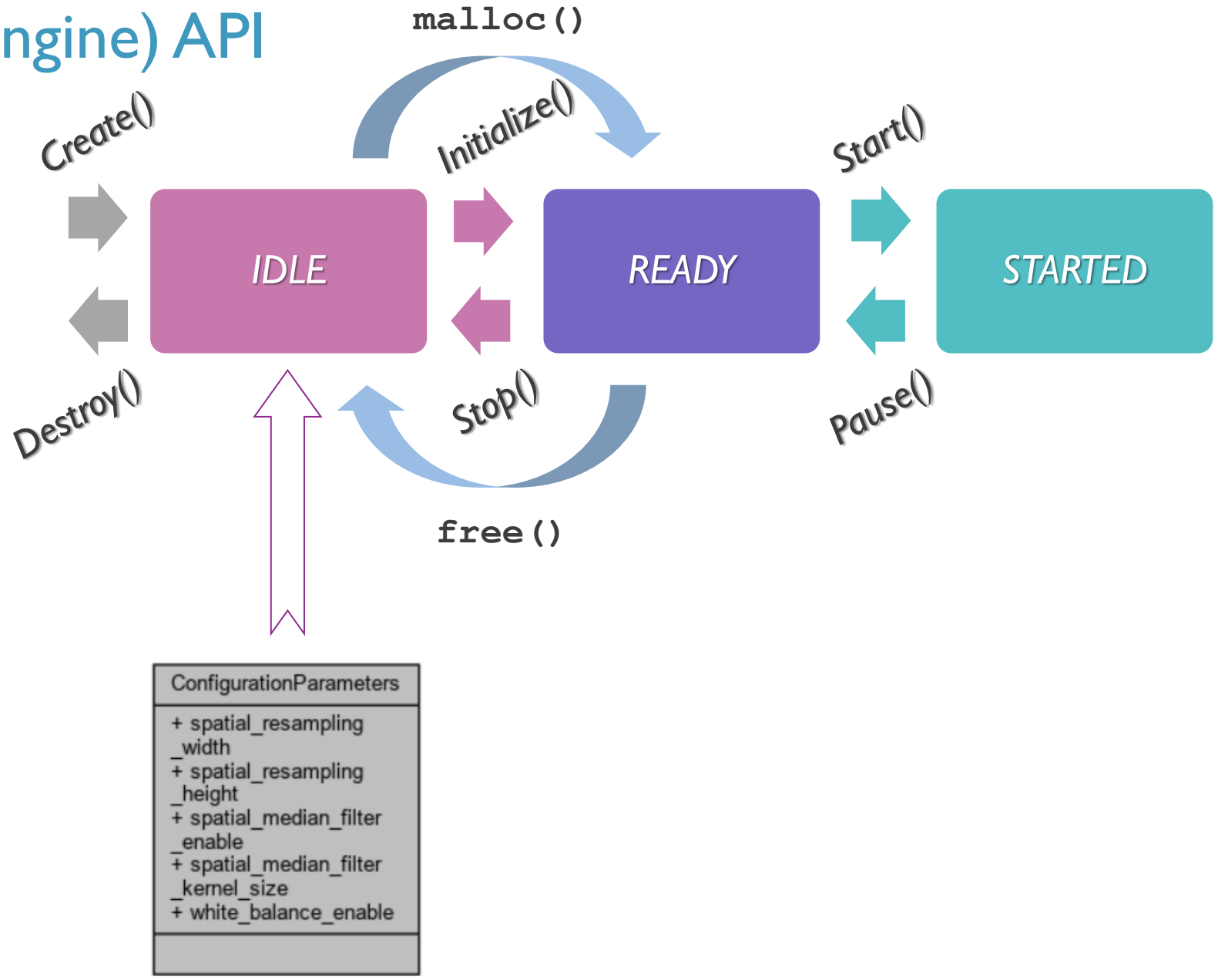
- A Pipeline has 3 states :
- API Calls can modify state
- Memory Allocations are function of state (for every pipeline stage)



The Mosaic (processing-engine) API

Respect for Internal States

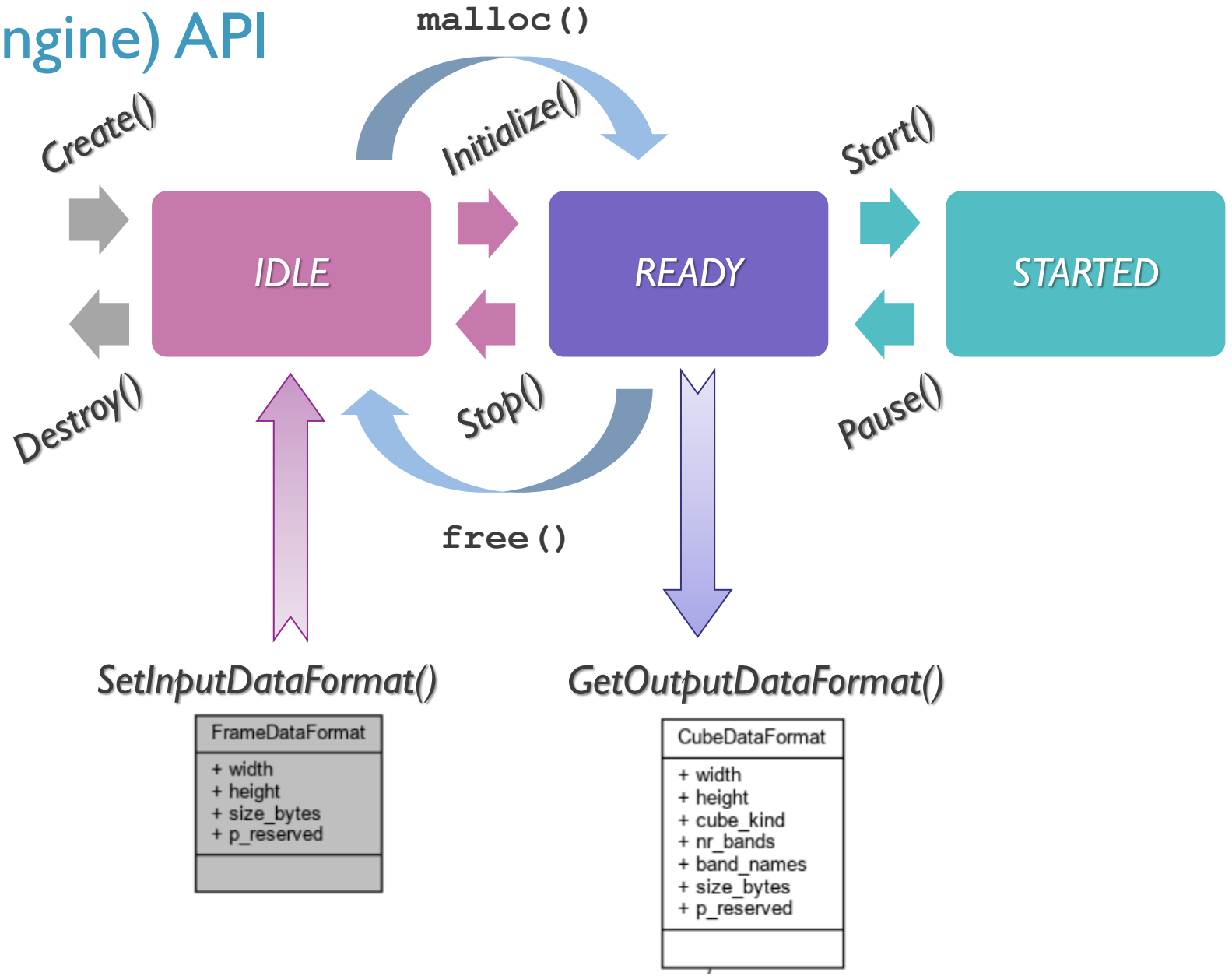
- A Pipeline has 3 states :
- API Calls can modify state
- Memory Allocations are function of state (for every pipeline stage)
- Different Parameter set in function of state



The Mosaic (processing-engine) API

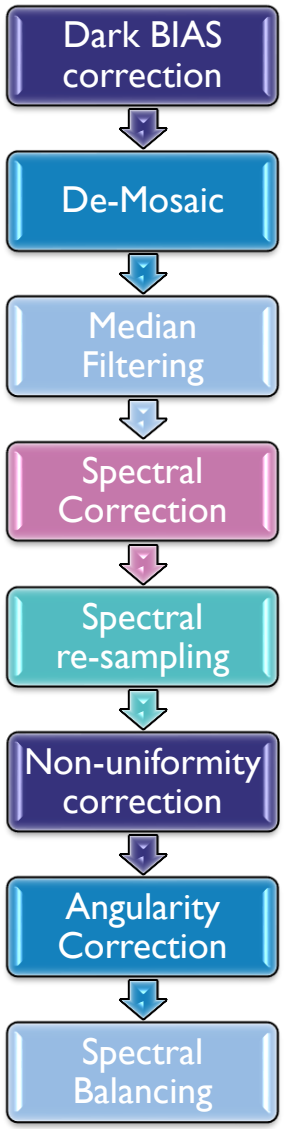
Respect for Internal States

- A Pipeline has 3 states :
- API Calls can modify state
- Memory Allocations are function of state (for every pipeline stage)
- Configure Input-format
  Get Output-format



The Mosaic (processing-engine) contains multiple "stages"

All stages can be configured and/or bypassed



- Based on dark reference frames
- De-Mosaic a frame into a cube
- Median filtering (to reduce noise) : available kernels: none, 3x3 and 5x5
- Apply Spectral correction : based on a sensor calibration file
- Spectral re-sampling : based on optical system (lens parameters)
- Correction for non-uniformity in the sensor
- Compensate spectra for angles introduced by the camera lens
- Make output spectra relative to a reference spectrum

The Mosaic (processing-engine) API

Creation is based on a file/folder-structure called : Context

- A context (folder) is a structured file/folder ensemble
- It holds all the necessary parts to configure a Mosaic Processing Pipeline:
  - A sensor calibration file
  - Dark-reference image(s)
  - White-reference image(s)
  - Non-Uniformity data frames
  - An Optical Setup File

◆ LoadContext()

```
HSI_MOSAIC_API HSI_RETURN LoadContext ( HANDLE *       o_p_context_handle,
                                        wchar_t const * i_pathname
                                      )
```

This Function will load a HSI Mosaic pipeline context from disc into its according memory object.

The organisation of how a context is saved to disc is documented in the user manual. The names of files and the folder structure is fixed by the tool, both for saving as well as for loading from disk.

Parameters

[out]	**o_p_context_handle**	the handle to the internal context structure (in memory)
[in]	**i_pathname**	the string from where to load the pipeline context, i.e. the name of the context folder.

NOTE : More explanations and API calls related to "context" – see separate section at the end.

The Mosaic (processing-engine) API

Creation is based on a file/folder-structure called : Context

- A context (folder) is a structured file/folder ensemble
- It holds all the necessary parts to configure a Mosaic Processing Pipeline:
  - A sensor calibration file
  - Dark-reference image(s)
  - White-reference image(s)
  - Non-Uniformity data frames
  - An Optical Setup File

- The Creation of a Mosaic Processing Pipeline is done based on such a Context-folder:

◆ Create()

```
HSI_MOSAIC_API HSI_RETURN Create ( HANDLE *  o_p_pipeline_handle,
                                   HANDLE    i_context_handle
                                 )
```

Function to create new pipeline object that will be able to process raw HSI Mosaic images.

Parameters

[out] **o_p_pipeline_handle** Handle to the HSI Mosaic pipeline object.

[in] **i_context_handle** Handle to the context object on how to build and configure the HSI Mosaic Pipelineapply

The MOSAIC – API
The EXAMPLE

The Mosaic (processing-engine) API

There is a Push/Pull mechanism to process Mosaic Data

◆ PushFrame()

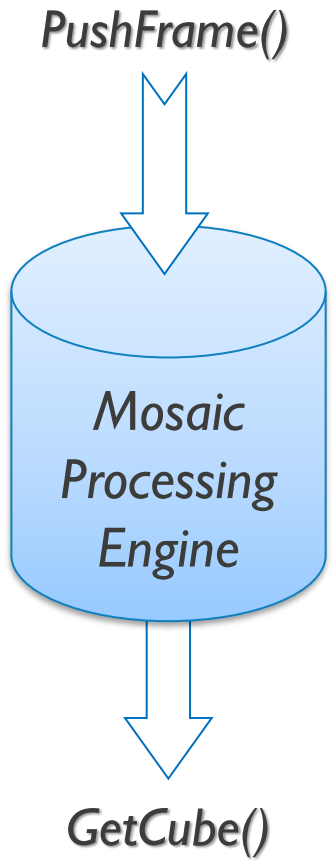
```
HSI_MOSAIC_API HSI_RETURN PushFrame ( HANDLE      i_pipeline_handle,
                                      FrameFloat  i_frame
                                    )
```

Function to push an input (raw) frame into a pipeline object to convert it into a processed HSI Mosaic cube.

◆ GetCube()

```
HSI_MOSAIC_API HSI_RETURN GetCube ( HANDLE      i_pipeline_handle,
                                    CubeFloat * o_p_cube,
                                    int         i_timeout_ms
                                  )
```

Function to retrieve a processed output HSI Mosaic Cube from the pipeline object. If several images are pushed into the pipeline object, this will act like a First-In-First-Out queue. Processed cubes will come out in the same order they were pushed in.



The Mosaic (processing-engine) API Example

Let's build up an example

- Let's include the API header file.
- All API functions return a result, indicating the success/failure of its actions.
- HSI has a build-in Logger tool. It takes a directory and a log-level as arguments.

```
#include "hsi_mosaic_api/hsi_mosaic_api.h"

    HSI_RETURN result;
    result = InitializeLogger (L"./logs/", LV_DEBUG);
```

The Mosaic (processing-engine) API Example

Let's build up an example

- We need a handle to the "context"
- We can Load a complete "context" from disk...

```
#include "hsi_mosaic_api/hsi_mosaic_api.h"

    HSI_RETURN result;
    result = InitializeLogger (L"./logs/", LV_DEBUG);

    HANDLE context = 0x0;
    result = LoadContext (&context, L"./resources/context/");
```

The Mosaic (processing-engine) API Example

Let's build up an example

- We need a handle to the data-processing pipeline
- Which we can create/configure from a complete "context"

```
#include "hsi_mosaic_api/hsi_mosaic_api.h"

    HSI_RETURN result;
    result = InitializeLogger (L"./logs/", LV_DEBUG);

    HANDLE context = 0x0;
    result = LoadContext (&context, L"./resources/context/");

    HANDLE pipeline = 0x0;
    result = Create (&pipeline, context);
```

The Mosaic (processing-engine) API Example

Let's build up an example

- When loading a frame from disk...
- ... we can use it to set the input-data-format

```
#include "hsi_mosaic_api/hsi_mosaic_api.h"

    HSI_RETURN result;
    result = InitializeLogger (L"./logs/", LV_DEBUG);

    HANDLE context = 0x0;
    result = LoadContext (&context, L"./resources/context/");

    HANDLE pipeline = 0x0;
    result = Create (&pipeline, context);

    FrameFloat data_frame = {};
    result = LoadFrame (&data_frame, L"./resources/leaves/frame.raw");
    result = SetInputDataFormat (pipeline, data_frame.format);
```

The Mosaic (processing-engine) API Example

Let's build up an example

- Now the data-processing pipeline can be Initialized. It will be internally fully configured, and all buffers will be allocated.
- And hence we can ask the pipeline what the output-data-format will be ...
- So that we can allocate an output cube of the correct size.

```
#include "hsi_mosaic_api/hsi_mosaic_api.h"

    HSI_RETURN result;
    result = InitializeLogger (L"./logs/", LV_DEBUG);

    HANDLE context = 0x0;
    result = LoadContext (&context, L"./resources/context/");

    HANDLE pipeline = 0x0;
    result = Create (&pipeline, context);

    FrameFloat data_frame = {};
    result = LoadFrame (&data_frame, L"./resources/leaves/frame.raw");
    result = SetInputDataFormat (pipeline, data_frame.format);

    result = Initialize (pipeline);
    CubeDataFormat output_data_format {};
    CubeFloat data_cube = {};
    result = GetOutputDataFormat (pipeline, &output_data_format);
    result = AllocateCube (&data_cube, output_data_format);
```

The Mosaic (processing-engine) API Example

Let's build up an example

```
#include "hsi_mosaic_api/hsi_mosaic_api.h"

    HSI_RETURN result;
    result = InitializeLogger (L"./logs/", LV_DEBUG);

    HANDLE context = 0x0;
    result = LoadContext (&context, L"./resources/context/");

    HANDLE pipeline = 0x0;
    result = Create (&pipeline, context);

    FrameFloat data_frame = {};
    result = LoadFrame (&data_frame, L"./resources/leaves/frame.raw");
    result = SetInputDataFormat (pipeline, data_frame.format);

    result = Initialize (pipeline);
    CubeDataFormat output_data_format {};
    CubeFloat data_cube = {};
    result = GetOutputDataFormat (pipeline, &output_data_format);
    result = AllocateCube (&data_cube, output_data_format);

    result = Start (pipeline);

    result = PushFrame (pipeline, data_frame);
    result = GetCube (pipeline, &data_cube, 8000);
```

- Now we can Start the data processing pipeline
- We can push a data-frame in ...
- ... and get the resulting output cube

The Mosaic (processing-engine) API Example

Let's build up an example

```
#include "hsi_mosaic_api/hsi_mosaic_api.h"

    HSI_RETURN result;
    result = InitializeLogger (L"./logs/", LV_DEBUG);
    HANDLE context = 0x0;
    result = LoadContext (&context, L"./resources/context/");
    HANDLE pipeline = 0x0;
    result = Create (&pipeline, context);
    FrameFloat data_frame = {};
    ret_val = LoadFrame (&data_frame, L"./resources/leaves/frame.raw");
    result = SetInputDataFormat (pipeline, data_frame.format);
    result = Initialize (pipeline);
    CubeDataFormat output_data_format {};
    CubeFloat data_cube = {};
    result = GetOutputDataFormat (pipeline, &output_data_format);
    result = AllocateCube (&data_cube, output_data_format);
    result = Start (pipeline);
    result = PushFrame (pipeline, data_frame);
    result = GetCube (pipeline, &data_cube, 8000);

    result = Pause (pipeline);
    result = Stop (pipeline);

    result = DeallocateCube (&data_cube);
    result = DeallocateCubeDataFormat (&output_data_format);
    result = DeallocateFrame (&data_frame);
    result = DeallocateContext (&context);
    result = Destroy (&pipeline);
```

- And now we need to do cleanup
  - Pause and Stop the pipeline
  - De-allocate cube, cube-data-format and frame
  - De-allocate the "context"
  - And Destroy the pipeline (free all internally allocated buffers)

imec

The Mosaic (processing-engine) API Example

Let's build up an example – apply a reference spectrum

- You can extract a "reference spectrum" from a cube => {*offset_x, offset_y, width, height*} ...
- ... and apply it to the data-processing pipeline (the coefficient here means a 95% white-tile used as reference)

```
#include "hsi_mosaic_api/hsi_mosaic_api.h"

    HSI_RETURN result;
    result = InitializeLogger (L"./logs/", LV_DEBUG);
    HANDLE context = 0x0;
    result = LoadContext (&context, L"./resources/context/");
    HANDLE pipeline = 0x0;
    result = Create (&pipeline, context);
    FrameFloat data_frame = {};
    ...
    result = Initialize (pipeline);
    ...
    result = Start (pipeline);
    result = PushFrame (pipeline, data_frame);
    result = GetCube (pipeline, &data_cube, 8000);

    SpectrumFloat reference_spectrum = {};
    result = ExtractSpectrumFromCube (&reference_spectrum, data_cube, {10,10,10,10});
    result = SetReferenceSpectrum (pipeline, reference_spectrum, 0.95);
```

The Mosaic (processing-engine) API Example

Let's build up an example – apply a reference spectrum

```
#include "hsi_mosaic_api/hsi_mosaic_api.h"

    HSI_RETURN result;
    result = InitializeLogger (L"./logs/", LV_DEBUG);
    HANDLE context = 0x0;
    result = LoadContext (&context, L"./resources/context/");
    HANDLE pipeline = 0x0;
    result = Create (&pipeline, context);
    FrameFloat data_frame = {};
    ...
    result = Initialize (pipeline);
    ...
    result = Start (pipeline);
    result = PushFrame (pipeline, data_frame);
    result = GetCube (pipeline, &data_cube, 8000);

    SpectrumFloat reference_spectrum = {};
    result = ExtractSpectrumFromCube (&reference_spectrum, data_cube, {10,10,10,10});
    result = SetReferenceSpectrum (pipeline, reference_spectrum, 0.95);

    result = Initialize (pipeline);
    result = DeallocateCubeDataFormat (&output_data_format); //-- clean up previous
    result = GetOutputDataFormat (pipeline, &output_data_format);
```

- We need to re-initialize the pipeline
- And ask for the new output data format

imec

The Mosaic (processing-engine) API Example

Let's build up an example – apply a reference spectrum

```
#include "hsi_mosaic_api/hsi_mosaic_api.h"

    HSI_RETURN result;
    result = InitializeLogger (L"./logs/", LV_DEBUG);
    HANDLE context = 0x0;
    result = LoadContext (&context, L"./resources/context/");
    HANDLE pipeline = 0x0;
    result = Create (&pipeline, context);
    FrameFloat data_frame = {};
    ...
    result = Initialize (pipeline);
    ...
    result = Start (pipeline);
    result = PushFrame (pipeline, data_frame);
    result = GetCube (pipeline, &data_cube, 8000);

    SpectrumFloat reference_spectrum = {};
    result = ExtractSpectrumFromCube (&reference_spectrum, data_cube, {10,10,10,10});
    result = SetReferenceSpectrum (pipeline, reference_spectrum, 0.95);

    result = Initialize (pipeline);
    result = DeallocateCubeDataFormat (&output_data_format); //-- clean up previous
    result = GetOutputDataFormat (pipeline, &output_data_format);
    result = AllocateCube (&data_cube, output_data_format);
    result = Start (pipeline);
    result = PushFrame (pipeline, data_frame);
    result = GetCube (pipeline, &data_cube, 8000);
    ...
    result = DeallocateSpectrum (&reference_spectrum);
```

- So that we can allocate an output cube ...
  push an input frame ...
  and get the processed (balanced) output cube.

- (don't forget to do cleanup!)

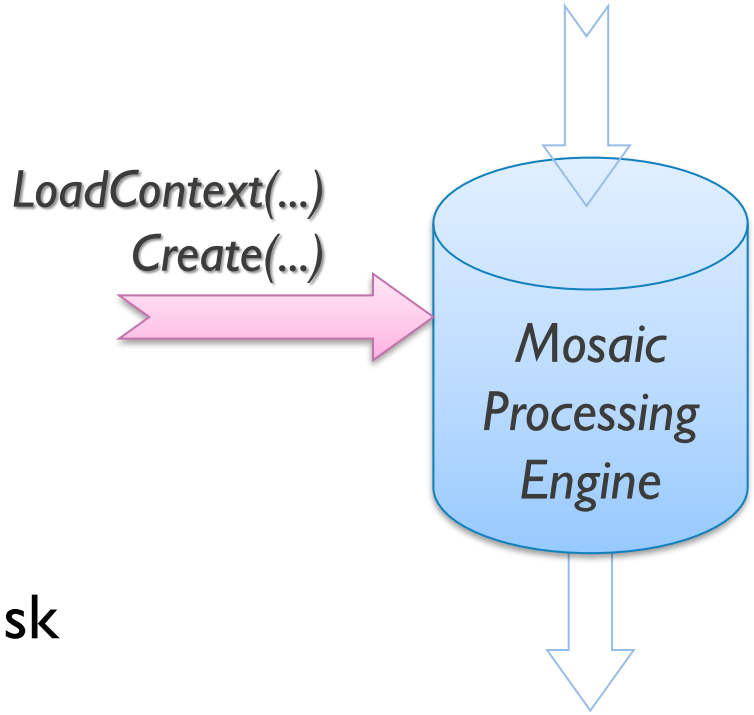
imec

The MOSAIC – API
The CONTEXT

The Mosaic (processing-engine) API - Context

A Context and how to build one yourself ...

- A "Context" contains ...
  - A sensor calibration file
  - Dark-reference frame(s) <= f(exposure_ms)
  - White-reference frame(s)
  - Optical Setup file : lens, f-number, ...
  - Non-uniformity frames : dark(s) + white(s)



- Fixed File/Folder structure/naming-scheme
- API can do LoadContext(...)/SaveContext(...) to/from disk
- Can be saved from the HSI-Mosaic GUI
- Can also be "build-up" by the user ...

The Mosaic (processing-engine) API - Context

A Context and how to build one yourself ...

- A "Context" can be "build-up" by the user ...

```
HSI_MOSAIC_API HSI_RETURN AllocateContext ( HANDLE * o_p_context_handle )
```

```
HSI_MOSAIC_API HSI_RETURN DeallocateContext ( HANDLE * io_p_context_handle )
```

```
HSI_MOSAIC_API HSI_RETURN ContextSetCalibrationFile ( HANDLE          i_context_handle,
                                                      wchar_t const * i_in_file_path
                                                    )
```

```
HSI_MOSAIC_API HSI_RETURN ContextSetDarkFieldReferences ( HANDLE       i_context_handle,
                                                          FrameFloat * i_p_dark_field_array,
                                                          int          i_array_length
                                                        )
```

```
HSI_MOSAIC_API HSI_RETURN ContextSetBrightFieldReferences ( HANDLE     i_context_handle,
                                                            FrameFloat i_bright_field,
                                                            FrameFloat i_dark_field
                                                          )
```

```
HSI_MOSAIC_API HSI_RETURN ContextSetOpticalSetup ( HANDLE       i_context_handle,
                                                   OpticalSetup  i_optical_setup
                                                 )
```

OpticalSetup
+ lens_f_number + focal_length_mm + exit_pupil_mm + manufacturer + optical_range

ContextStatus
+ context_ready + has_dark_reference + has_sensor_calibration + has_nonuniformity + has_optical_setup

```
HSI_MOSAIC_API HSI_RETURN ContextGetStatus ( HANDLE        i_context_handle,
                                             ContextStatus * o_p_status_flags
                                           )
```

- ... and written to / read from disk

```
HSI_MOSAIC_API HSI_RETURN SaveContext ( HANDLE          i_context_handle,
                                        wchar_t const * i_pathname
                                      )
```

```
HSI_MOSAIC_API HSI_RETURN LoadContext ( HANDLE *        o_p_context_handle,
                                        wchar_t const * i_pathname
                                      )
```

embracing a better life