



PIXELPLUS SDK Documentation

Version 0.9
Last updated: 2025-10-23

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 AlgEntry Struct Reference	5
3.1.1 Detailed Description	5
3.2 CConverter Class Reference	5
3.2.1 Detailed Description	6
3.3 CCpuParaConverter Class Reference	6
3.3.1 Detailed Description	6
3.4 CCpuSerialConverter Class Reference	6
3.4.1 Detailed Description	7
3.5 CImageDisplayer Class Reference	7
3.5.1 Detailed Description	7
3.6 cimage::CImageDisplayerCPP Class Reference	7
3.6.1 Detailed Description	9
3.6.2 Member Function Documentation	9
3.6.2.1 setImage()	9
3.6.2.2 setImageRaw()	10
3.6.2.3 wheelScroll()	11
3.7 CImageUploadDesc Struct Reference	11
3.7.1 Detailed Description	12
3.8 ipm::ClpmCpuEnv Class Reference	12
3.8.1 Detailed Description	12
3.8.2 Member Function Documentation	12
3.8.2.1 bestSimdFor()	12
3.9 ipm::ClpmEnv Class Reference	13
3.9.1 Detailed Description	13
3.10 ipmcommon::ClpmFuncTable Class Reference	13
3.10.1 Detailed Description	14
3.10.2 Member Function Documentation	14
3.10.2.1 getAlgorithmList()	14
3.10.2.2 process()	15
3.11 ipm::ClpmGpuEnv Class Reference	15
3.11.1 Detailed Description	16
3.11.2 Member Function Documentation	16
3.11.2.1 selectByNameSubstring()	16
3.12 csh_img::CSH_Image Class Reference	16
3.12.1 Detailed Description	18
3.12.2 Constructor & Destructor Documentation	19

3.12.2.1 CSH_Image()	19
3.12.3 Member Function Documentation	19
3.12.3.1 allocateBuffer()	19
3.12.3.2 copy()	20
3.12.3.3 copyBufferPointer() [1/2]	20
3.12.3.4 copyBufferPointer() [2/2]	21
3.12.3.5 data() [1/2]	21
3.12.3.6 data() [2/2]	21
3.12.3.7 getBufferSize()	22
3.12.3.8 getCameraId()	22
3.12.3.9 getFormat()	22
3.12.3.10 getHeight()	22
3.12.3.11 getImageCount()	23
3.12.3.12 getImagePtr() [1/2]	23
3.12.3.13 getImagePtr() [2/2]	23
3.12.3.14 getMemoryAlign()	24
3.12.3.15 getMemoryBit()	24
3.12.3.16 getOriginalBit()	24
3.12.3.17 getPattern()	25
3.12.3.18 getSelectedImage()	25
3.12.3.19 getWidth()	25
3.12.3.20 isEnabled()	25
3.12.3.21 loadImage()	25
3.12.3.22 recomputeBufferSize()	26
3.12.3.23 saveImage()	26
3.12.3.24 setSelectedImage()	26
3.12.3.25 totalBytes()	27
3.13 CSH_Log Class Reference	27
3.13.1 Detailed Description	27
3.14 CWatchTime Class Reference	28
3.14.1 Detailed Description	28
3.14.2 Constructor & Destructor Documentation	29
3.14.2.1 CWatchTime()	29
3.14.3 Member Function Documentation	29
3.14.3.1 GetCurrentTimeString()	29
3.14.3.2 GetCurrentTimeStringA()	29
3.14.3.3 GetMicroSecond()	29
3.14.3.4 GetMilliSecond()	30
3.14.3.5 GetSecond()	30
3.14.3.6 getString()	30
3.14.3.7 start()	30
3.14.3.8 stop()	31

3.15 FuncInfo Struct Reference	31
3.15.1 Detailed Description	31
3.16 ipm::GpuInfo Struct Reference	31
3.16.1 Detailed Description	32
3.17 cimage::Mat4 Struct Reference	32
3.17.1 Detailed Description	32
3.18 cimage::Quat Struct Reference	33
3.18.1 Detailed Description	33
3.19 cimage::UploadDescriptor Struct Reference	33
3.19.1 Detailed Description	34
3.20 ipm_internal::UserCustomLoader Class Reference	34
3.20.1 Detailed Description	34
3.20.2 Member Function Documentation	35
3.20.2.1 loadOnce()	35
3.21 cimage::Vec2 Struct Reference	35
3.21.1 Detailed Description	35
3.22 cimage::Vec3 Struct Reference	35
3.22.1 Detailed Description	35
4 File Documentation	37
4.1 CFrameGrabber.h	37
4.2 CFrameGrabber/CGrabberConfig.h File Reference	38
4.2.1 Detailed Description	38
4.3 CGrabberConfig.h	38
4.4 IFrameGrabImpl.h	39
4.5 CImageDisplay/CImageDisplay.h File Reference	40
4.5.1 Detailed Description	41
4.5.2 Enumeration Type Documentation	41
4.5.2.1 PixelLayout	41
4.5.3 Function Documentation	42
4.5.3.1 __attribute__()	42
4.6 CImageDisplay.h	46
4.7 CImageDisplay/CImageDisplayC.h File Reference	49
4.7.1 Detailed Description	50
4.7.2 Function Documentation	50
4.7.2.1 __attribute__()	50
4.8 CImageDisplayC.h	52
4.9 CImageDisplay/CImageDisplayCPP.h File Reference	54
4.9.1 Detailed Description	55
4.10 CImageDisplayCPP.h	55
4.11 ImageProcessorManager/CImageProcessMng.h File Reference	57
4.11.1 Detailed Description	58

4.11.2 Typedef Documentation	58
4.11.2.1 DisplayCallback	58
4.11.3 Function Documentation	59
4.11.3.1 __attribute__()	59
4.12 ClImageProcessMng.h	62
4.13 ImageProcessorManager/ClpmCpuEnv.h File Reference	63
4.13.1 Detailed Description	63
4.13.2 Enumeration Type Documentation	63
4.13.2.1 En_OpProfile	63
4.13.2.2 En_SimdKind	64
4.14 ClpmCpuEnv.h	64
4.15 ImageProcessorManager/ClpmEnv.h File Reference	66
4.15.1 Detailed Description	66
4.16 ClpmEnv.h	66
4.17 ImageProcessorManager/ClpmFuncTable.h File Reference	67
4.17.1 Detailed Description	68
4.17.2 Function Documentation	68
4.17.2.1 ClpmFuncTable_RegisterDummyForDev()	68
4.18 ClpmFuncTable.h	69
4.19 ImageProcessorManager/ClpmGpuEnv.h File Reference	69
4.19.1 Detailed Description	70
4.20 ClpmGpuEnv.h	70
4.21 ImageProcessorManager/ClpmUserCustomLoader.h File Reference	71
4.21.1 Detailed Description	72
4.22 ClpmUserCustomLoader.h	72
4.23 CConverter.h	73
4.24 CCpuParaConverter.h	74
4.25 CCpuSerialConverter.h	74
4.26 CGpuClConverter.h	75
4.27 CGpuCudaConverter.h	75
4.28 CGpuGIComputeConverter.h	75
4.29 ImageProcessorManager/IpmpClamp.h File Reference	75
4.29.1 Detailed Description	75
4.29.2 Function Documentation	76
4.29.2.1 __attribute__()	76
4.30 IpmpClamp.h	76
4.31 ImageProcessorManager/IpmpStringUtils.h File Reference	77
4.31.1 Detailed Description	78
4.31.2 Function Documentation	78
4.31.2.1 u8_to_w()	78
4.31.2.2 w_to_u8()	78
4.32 IpmpStringUtils.h	79

4.33 ImageProcessorManager/lpmTypes.h File Reference	80
4.33.1 Detailed Description	81
4.33.2 Typedef Documentation	81
4.33.2.1 lpmFn	81
4.33.3 Enumeration Type Documentation	82
4.33.3.1 EnlpmModule	82
4.33.3.2 EnProcessBackend	82
4.33.3.3 lpmStatus	83
4.34 lpmTypes.h	84
4.35 ClpmUserCustom.h	84
4.36 utility/CSH_Image/CSH_Image.h File Reference	85
4.36.1 Detailed Description	86
4.36.2 Enumeration Type Documentation	86
4.36.2.1 CopyMode	86
4.36.2.2 En_ImageFormat	86
4.36.2.3 En_ImageMemoryAlign	87
4.36.2.4 En_ImagePattern	88
4.37 CSH_Image.h	88
4.38 utility/CWatchTime/CWatchTime.h File Reference	91
4.38.1 Detailed Description	91
4.39 CWatchTime.h	92
4.40 utility/SH_Log/CSH_Log.h File Reference	92
4.40.1 Detailed Description	93
4.40.2 Macro Definition Documentation	93
4.40.2.1 CSH_FUNC_SIG	93
4.40.2.2 LOG_WRITE	93
4.40.2.3 LOG_WRITE_MSG	94
4.40.3 Enumeration Type Documentation	94
4.40.3.1 LogLevel	94
4.40.4 Function Documentation	95
4.40.4.1 __attribute__()	95
4.41 CSH_Log.h	102
Index	105

Chapter 1

Class Index

1.1 Class List

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

AlgEntry	Entry stored in a module catalog	5
CConverter	Singleton Converter Module	5
CCpuParaConverter	CPU parallel converter (thread-safe, stateless)	6
CCpuSerialConverter	CPU Serial Converter (thread-safe, stateless)	6
CImageDisplayer	Owns/aliases an image and exposes 2D/3D transforms and upload metadata	7
cimage::CImageDisplayerCPP	RAII C++ wrapper around the C ABI. No rendering; forwards to C API	7
CImageUploadDesc	Upload descriptor (C layout) matching the C++ UploadDescriptor	11
ipm::ClpmCpuEnv	CPU capability probe (non-owning, POD-like; call Detect once)	12
ipm::ClpmEnv	Process-wide IPM environment (CPU + GPU)	13
ipmcommon::ClpmFuncTable	Function Table Registry (singleton)	13
ipm::ClpmGpuEnv	GPU environment probe and selection (non-singleton)	15
csh_img::CSH_Image	Image container with explicit format metadata and flexible buffer ownership	16
CSH_Log	Thread-safe, Unicode-aware singleton logger	27
CWatchTime	RAII-less stopwatch using high_resolution_clock with simple elapsed queries	28
FuncInfo	Metadata for a registered algorithm	31
ipm::GpuInfo	One GPU record with runtime capability flags	31
cimage::Mat4	Column-major 4x4 matrix (OpenGL style)	32
cimage::Quat	Quaternion (w + xyz). Identity by default	33

cimage::UploadDescriptor	
Description of the memory block to upload to a GPU/renderer	33
ipm_internal::UserCustomLoader	
Singleton user plug-in loader for the User_Custom module	34
cimage::Vec2	
2D vector	35
cimage::Vec3	
3D vector	35

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

CFrameGrabber/ CFrameGrabber.h	37
CFrameGrabber/ CGrabberConfig.h Basic runtime configuration for a frame grabber backend	38
CFrameGrabber/ IFrameGrabImpl.h	39
CImageDisplay/ CImageDisplay.h Math-only image “displayer” that owns an image buffer and exposes 2D/3D transform state and upload descriptors. No rendering is performed	40
CImageDisplay/ CImageDisplayC.h C ABI wrapper for <code>cimage::CImageDisplay</code>	49
CImageDisplay/ CImageDisplayCPP.h Thin C++ wrapper around the pure-C CImageDisplay ABI	54
ImageProcessorManager/ CImageProcessMng.h High-level image processing pipeline manager (frame handoff + staged processing + display callback)	57
ImageProcessorManager/ ClpmCpuEnv.h CPU feature probing (x86/x86_64 AVX2/AVX-512/AMX, ARM NEON/SVE/SVE2) and best-SIMD selection	63
ImageProcessorManager/ ClpmEnv.h Orchestrator singleton wrapping CPU and GPU environment probes and selection	66
ImageProcessorManager/ ClpmFuncTable.h Lazy-initialized registry mapping (backend,module,algIndex) -> callable lpmFn + UI name . . .	67
ImageProcessorManager/ ClpmGpuEnv.h GPU runtime/environment probe (OS adapters, CUDA/OpenCL presence, optional OpenGL probing) and selection	69
ImageProcessorManager/ ClpmUserCustomLoader.h Loader for User_Custom plug-ins and exposure of their registered algorithms	71
ImageProcessorManager/ lpmClamp.h Tiny header-only clamping and saturating-cast utilities optimized for image pipelines	75
ImageProcessorManager/ lpmStringUtils.h Lightweight, dependency-minimal UTF-8/UTF-16 helpers and enum stringifiers for IPM	77
ImageProcessorManager/ lpmTypes.h Core IPM type aliases, enums, and small PODs used across the image processing modules .	80
ImageProcessorManager/Converter/ CCConverter.h	73
ImageProcessorManager/Converter/ CCpuParaConverter.h	74
ImageProcessorManager/Converter/ CCpuSerialConverter.h	74

ImageProcessorManager/Converter/ CGpuCIConverter.h	75
ImageProcessorManager/Converter/ CGpuCudaConverter.h	75
ImageProcessorManager/Converter/ CGpuGIComputeConverter.h	75
plugins/ClpmUserCustom/ ClpmUserCustom.h	84
utility/CSH_Image/ CSH_Image.h	
Cross-platform C++17 image container class for camera/vision pipelines	85
utility/CWatchTime/ CWatchTime.h	
Lightweight wall-clock stopwatch and timestamp formatting utilities	91
utility/SH_Log/ CSH_Log.h	
Simple thread-safe logging class with printf-style and complete message logging	92

Chapter 3

Class Documentation

3.1 AlgEntry Struct Reference

Entry stored in a module catalog.

```
#include <IpmTypes.h>
```

Public Attributes

- **int alg**
Algorithm index/key within the module.
- **FuncInfo func**
Function + localized UI name.

3.1.1 Detailed Description

Entry stored in a module catalog.

Combines an integer algorithm ID (stable within module/backend) and [FuncInfo](#).

Definition at line [109](#) of file [IpmTypes.h](#).

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

- ImageProcessorManager/[IpmTypes.h](#)

3.2 CConverter Class Reference

Singleton Converter Module.

```
#include <CConverter.h>
```

3.2.1 Detailed Description

Singleton Converter Module.

- Wraps conversion functions of each backend (CPU Serial/Parallel, GL Compute, OpenCL, CUDA) and offers them as catalogs (lists of [AlgEntry](#)).
- The function table (CipmFuncTable) reads this catalog as is and calls registerFunc.

Definition at line 22 of file [CConverter.h](#).

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

- ImageProcessorManager/Converter/CConverter.h

3.3 CCpuParaConverter Class Reference

CPU parallel converter (thread-safe, stateless).

```
#include <CCpuParaConverter.h>
```

3.3.1 Detailed Description

CPU parallel converter (thread-safe, stateless).

- YUV422 8bit -> RGB888/BGR888 (automatic ordering based on output format)
- RGB888/BGR888 -> Gray8 (automatic input format detection)

Definition at line 10 of file [CCpuParaConverter.h](#).

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

- ImageProcessorManager/Converter/CCpuParaConverter.h

3.4 CCpuSerialConverter Class Reference

CPU Serial Converter (thread-safe, stateless).

```
#include <CCpuSerialConverter.h>
```

3.4.1 Detailed Description

CPU Serial Converter (thread-safe, stateless).

- YUV422 8bit -> RGB888/BGR888 (automatic ordering based on the output format)
- RGB888/BGR888 -> Gray8 (automatic input format detection)

Definition at line 10 of file [CCpuSerialConverter.h](#).

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

- ImageProcessorManager/Converter/CCpuSerialConverter.h

3.5 CImageDisplay Class Reference

Owns/aliases an image and exposes 2D/3D transforms and upload metadata.

```
#include <CImageDisplay.h>
```

3.5.1 Detailed Description

Owns/aliases an image and exposes 2D/3D transforms and upload metadata.

The class is rendering-framework agnostic. Use uploadDesc to obtain the pixel/stride/format info and the various matrix getters for your shader pipeline. Input hooks are provided to drive interaction from a host UI.

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

- CImageDisplay/[CImageDisplay.h](#)

3.6 cimage::CImageDisplayCPP Class Reference

RAII C++ wrapper around the C ABI. No rendering; forwards to C API.

```
#include <CImageDisplayCPP.h>
```

Public Member Functions

- **CImageDisplayerCPP** ()
Construct and allocate an instance. Throws `std::bad_alloc` on failure.
- **~CImageDisplayerCPP** ()
Destroy the owned instance.
- **CImageDisplayerCPP** (CImageDisplayerCPP &&o) noexcept
Move construct, transferring ownership.
- **CImageDisplayerCPP** & **operator=** (CImageDisplayerCPP &&o) noexcept
Move assign, destroying any currently owned handle.
- void **setImage** (const **CSH_Image** &img, **CSH_Image::CopyMode** mode=CSH_Image::CopyMode::Shallow)
*Set image from **CSH_Image** with copy semantics.*
- void **setImageRaw** (uint32_t w, uint32_t h, **CImgFormat** fmt, **CImgPattern** pat, **CImgAlign** align, const void *pixels, size_t bytes, **CImgCopyMode** mode)
*Set image from a raw pointer (when not using **CSH_Image**).*
- void **setViewport** (int w, int h)
Set viewport size in pixels.
- void **setFitMode** (**FitMode** m)
Set fit strategy (None/Fit/Fill/Stretch).
- void **setDimensionality** (**Dimensionality** d)
Switch between 2D and 3D modes.
- void **set2DAnchor** (float ax, float ay)
Set normalized anchor inside image rect $[0..1]^2$.
- void **set2DTranslation** (float tx, float ty)
Set 2D translation in viewport pixels.
- void **set2DScale** (float sx, float sy)
Set 2D scale factors.
- void **set2DRotationDeg** (float deg)
Set rotation in degrees (CCW).
- void **reset2D** ()
Reset 2D transform to identity.
- void **set3DModelTranslate** (float x, float y, float z)
Set model translation.
- void **set3DModelScale** (float x, float y, float z)
Set model scale.
- void **set3DModelRotationQuat** (float w, float x, float y, float z)
Set model rotation (quaternion).
- void **reset3DModel** ()
Reset model transform to identity.
- void **set3DTarget** (float x, float y, float z)
Set look-at target.
- void **set3DEye** (float x, float y, float z)
Set camera eye.
- void **set3DUp** (float x, float y, float z)
Set camera up vector.
- void **set3DOrbitStyle** (CImgOrbitStyle s)
Set orbit interaction style.
- void **setOrtho** (float l, float r, float b, float t, float n, float f)
Set orthographic projection.
- void **setPerspective** (float fovyDeg, float aspect, float zNear, float zFar)

-
- *Set perspective projection (fovy in degrees).*
void **model2D_3x3** (float outRowMajor3x3[9]) const
 - *Fetch current 2D model (row-major 3x3).*
void **model3D_4x4** (float outColMajor4x4[16]) const
 - *Fetch current 3D model (column-major 4x4).*
void **view3D_4x4** (float outColMajor4x4[16]) const
 - *Fetch current 3D view (column-major 4x4).*
void **proj_4x4** (float outColMajor4x4[16]) const
 - *Fetch current projection (column-major 4x4).*
void **mvp3D_4x4** (float outColMajor4x4[16]) const
 - *Fetch current MVP = P*V*M (column-major 4x4).*
void **triStrip2D_XYUV** (float out4x4[16]) const
 - *Transformed quad as 2D tri-strip with UVs ({x,y,u,v} per vertex).*
CImageUploadDesc **uploadDesc** () const
 - *Current upload descriptor (pointer/size/stride/layout).*
void **beginPointer** (float x, float y, MouseButton btn, KeyMod keyMods)
 - *Begin a pointer interaction.*
void **updatePointer** (float x, float y)
 - *Update pointer position during active interaction.*
void **endPointer** ()
 - *End current pointer interaction.*
void **wheelScroll** (float delta, float cx, float cy)
 - *Mouse wheel / trackpad zoom or dolly.*
void **keyPan2D** (float dx, float dy)
 - *Keyboard panning in 2D mode (pixels).*
void **keyDolly3D** (float amount)
 - *Keyboard dolly in 3D mode (world units along view).*
CImageDisplayHandle **raw** () const noexcept
 - *Expose raw C handle for low-level interop.*

Static Public Member Functions

- static void **triStrip3D_XYUV_ObjectSpace** (float out4x4[16])
Unit quad in object space for 3D pipelines ({x,y,u,v} per vertex).

3.6.1 Detailed Description

RAII C++ wrapper around the C ABI. No rendering; forwards to C API.

This wrapper creates/destroys the underlying C instance and offers typed helpers. ABI remains stable because the boundary stays in C.

Definition at line 98 of file [CImageDisplayerCPP.h](#).

3.6.2 Member Function Documentation

3.6.2.1 setImage()

```
void cimage::CImageDisplayerCPP::setImage (
    const csh_img::CSH_Image & img,
    csh_img::CopyMode mode = csh_img::CopyMode::Shallow) [inline]
```

Set image from [csh_img::CSH_Image](#) with copy semantics.

Parameters

<i>img</i>	Source image.
<i>mode</i>	Copy mode (MetaOnly/Shallow/Deep). Default Shallow.

Definition at line 125 of file [CImageDisplayerCPP.h](#).

```

00125                                     {
00126         const auto fmt = static_cast<CImgFormat>(static_cast<uint32_t>(img.getFormat()));
00127         const auto pat = static_cast<CImgPattern>(static_cast<uint32_t>(img.getPattern()));
00128         const auto alg = static_cast<CImgAlign>(static_cast<uint32_t>(img.getMemoryAlign()));
00129
00130         const void* data = nullptr;
00131         size_t bytes = 0;
00132         if (mode != csh_img::CopyMode::MetaOnly) {
00133             data = img.data();
00134             bytes = img.getBufferSize();
00135         }
00136         cimgSetImageRaw(h_, img.getWidth(), img.getHeight(), fmt, pat, alg, data, bytes,
00137             static_cast<CImgCopyMode>(static_cast<uint32_t>(mode)));
00138     }

```

References [csh_img::CSH_Image::data\(\)](#), [csh_img::CSH_Image::getBufferSize\(\)](#), [csh_img::CSH_Image::getFormat\(\)](#), [csh_img::CSH_Image::getHeight\(\)](#), [csh_img::CSH_Image::getMemoryAlign\(\)](#), [csh_img::CSH_Image::getPattern\(\)](#), and [csh_img::CSH_Image::getWidth\(\)](#).

3.6.2.2 setImageRaw()

```

void cimage::CImageDisplayerCPP::setImageRaw (
    uint32_t w,
    uint32_t h,
    CImgFormat fmt,
    CImgPattern pat,
    CImgAlign align,
    const void * pixels,
    size_t bytes,
    CImgCopyMode mode) [inline]

```

Set image from a raw pointer (when not using CSH_Image).

Parameters

<i>w,h</i>	Dimensions in pixels.
<i>fmt</i>	Format.
<i>pat</i>	Pattern / channel order.
<i>align</i>	Memory packing/alignment.
<i>pixels</i>	Pointer to buffer (nullable for MetaOnly).
<i>bytes</i>	Total buffer size.
<i>mode</i>	Copy semantics.

Definition at line 150 of file [CImageDisplayerCPP.h](#).

```

00152                                     {
00153         cimgSetImageRaw(h_, w, h, fmt, pat, align, pixels, bytes, mode);
00154     }

```

3.6.2.3 wheelScroll()

```
void cimage::CImageDisplayerCPP::wheelScroll (
    float delta,
    float cx,
    float cy) [inline]
```

Mouse wheel / trackpad zoom or dolly.

Parameters

<i>delta</i>	Positive for zoom/dolly in.
<i>cx,cy</i>	Cursor at event time (pixels).

Definition at line 240 of file [CImageDisplayerCPP.h](#).

```
00240 { cimgWheelScroll(h_, delta, cx, cy); }
```

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

- CImageDisplayer/[CImageDisplayerCPP.h](#)

3.7 CImageUploadDesc Struct Reference

Upload descriptor (C layout) matching the C++ UploadDescriptor.

```
#include <CImageDisplayerC.h>
```

Public Attributes

- `const uint8_t * data`
Pointer to buffer start (nullable).
- `size_t sizeBytes`
Total buffer size in bytes.
- `int32_t height`
Dimensions in pixels.
- `int32_t bytesPerPixel`
Bytes per pixel if tightly packed.
- `int32_t strideBytes`
Row pitch in bytes (0 if tightly packed).
- `int32_t layout`
0=Unknown, 1=Gray8, 2=RGB888, 3=BGR888, 4=YUV422, 5=RGB565, 6=Gray16, 7=Bayer16
- `int32_t yuv422Pattern`
0=YUYV, 1=UYVY, 2=VYVU, 3=VYUY
- `int32_t isPacked`
Boolean (0/1).
- `int32_t isLittleEndian16`
Boolean (0/1).

3.7.1 Detailed Description

Upload descriptor (C layout) matching the C++ UploadDescriptor.

Note

`data` may be NULL for MetaOnly copies.

Definition at line 92 of file [CImageDisplayC.h](#).

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

- CImageDisplay/[CImageDisplayC.h](#)

3.8 ipm::CipmCpuEnv Class Reference

CPU capability probe (non-owning, POD-like; call [Detect](#) once).

```
#include <CipmCpuEnv.h>
```

Public Member Functions

- void **Detect** ()
Probe CPU family and SIMD features (idempotent per instance).
- int **simdMaxBits** () const
Maximum generic SIMD vector width in bits (AMX excluded).
- int **sveVectorBits** () const
SVE vector length in bits (0 if unknown/not applicable).
- [En_SimdKind](#) **bestSimdGeneric** () const
Best generic SIMD candidate independent of workload.
- [En_SimdKind](#) **bestSimdFor** ([En_OpProfile](#) prof) const
Choose best SIMD for a given workload profile.

3.8.1 Detailed Description

CPU capability probe (non-owning, POD-like; call [Detect](#) once).

The object caches the detected state; accessors are cheap and thread-safe after [Detect](#) completes.

Definition at line 56 of file [CipmCpuEnv.h](#).

3.8.2 Member Function Documentation

3.8.2.1 bestSimdFor()

```
En\_SimdKind ipm::CipmCpuEnv::bestSimdFor (  
    En\_OpProfile prof) const
```

Choose best SIMD for a given workload profile.

Parameters

<i>prof</i>	Operation profile.
-------------	--------------------

Returns

Preferred SIMD kind (or #En_SimdKind::None).

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

- ImageProcessorManager/[ClpmCpuEnv.h](#)

3.9 ipm::ClpmEnv Class Reference

Process-wide IPM environment (CPU + GPU).

```
#include <ClpmEnv.h>
```

Public Member Functions

- void **Initialize** ()
Initialize CPU/GPU probes and write diagnostic summary (idempotent).
- void **Refresh** ()
Refresh GPU list and runtime states (CPU rarely changes; not re-probed by default).

Static Public Member Functions

- static ClpmEnv & **Instance** ()
Get the singleton instance (calls [Initialize](#) once on first use).

3.9.1 Detailed Description

Process-wide IPM environment (CPU + GPU).

Definition at line [39](#) of file [ClpmEnv.h](#).

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

- ImageProcessorManager/[ClpmEnv.h](#)

3.10 ipmcommon::ClpmFuncTable Class Reference

Function Table Registry (singleton).

```
#include <ClpmFuncTable.h>
```

Public Member Functions

- [IpmStatus process](#) ([ipmcommon::EnProcessBackend](#) backend, [ipmcommon::EnIpmModule](#) module, int alg↵ Index, const [csh_img::CSH_Image](#) *in, [csh_img::CSH_Image](#) *out, void *param1, void *param2) const
Dispatch a processing call to the registered function.
- `std::vector< std::pair< int, std::wstring > >` [getAlgorithmList](#) ([ipmcommon::EnProcessBackend](#) backend, [ipmcommon::EnIpmModule](#) module) const
Enumerate algorithms for (backend,module) for UI population.

Static Public Member Functions

- static `CipmFuncTable &` **Instance** ()
Get the singleton (initializes on first use).
- static const `std::vector< std::wstring > &` **getBackendNames** ()
Localized backend names in enum order (for UI).
- static const `std::vector< std::wstring > &` **getModuleNames** ()
Localized module names in enum order (for UI).
- static bool **TryParseBackend** (const std::wstring &name, [ipmcommon::EnProcessBackend](#) &out)
Parse backend name into enum (exact match).
- static bool **TryParseModule** (const std::wstring &name, [ipmcommon::EnIpmModule](#) &out)
Parse module name into enum (exact match).

3.10.1 Detailed Description

Function Table Registry (singleton).

Usage:

- Call [Instance\(\)](#) anywhere; the first call triggers `InitFuncTable()` once.
- [process](#) dispatches to the registered function and returns an [IpmStatus](#).
- [getAlgorithmList](#) exposes (algIndex, uiName) for UI population.

Threading:

- Registration is protected by a mutex.
- Lazy initialization is guarded by `std::once_flag`.
- Read access is lock-free after initialization.

Definition at line 42 of file [CipmFuncTable.h](#).

3.10.2 Member Function Documentation

3.10.2.1 [getAlgorithmList\(\)](#)

```
std::vector< std::pair< int, std::wstring > > ipmcommon::CipmFuncTable::getAlgorithmList (  
    ipmcommon::EnProcessBackend backend,  
    ipmcommon::EnIpmModule module) const
```

Enumerate algorithms for (backend,module) for UI population.

Returns

Vector of (algIndex, uiName).

3.10.2.2 process()

```
IpmStatus ipmcommon::CIpmFuncTable::process (
    ipmcommon::EnProcessBackend backend,
    ipmcommon::EnIpmModule module,
    int algIndex,
    const csh_img::CSH_Image * in,
    csh_img::CSH_Image * out,
    void * param1,
    void * param2) const
```

Dispatch a processing call to the registered function.

Parameters

<i>backend</i>	Execution backend.
<i>module</i>	Module (Converter/Scaler/Splitter/User_Custom).
<i>algIndex</i>	Algorithm index/key within the module.
<i>in</i>	Input image (nullable for algorithms that don't need it).
<i>out</i>	Output image (must not be null).
<i>param1</i>	Opaque parameter 1.
<i>param2</i>	Opaque parameter 2.

Returns

[IpmStatus](#) result of the function call (or an error mapped by the table).

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

- ImageProcessorManager/[CIpmFuncTable.h](#)

3.11 ipm::CIpmGpuEnv Class Reference

GPU environment probe and selection (non-singleton).

```
#include <CIpmGpuEnv.h>
```

Public Member Functions

- void **Refresh** ()
Full refresh: enumerate OS adapters, guess active display GPU, probe GL/CUDA/OpenCL.
- bool [selectByNameSubstring](#) (const std::string &substr, bool preferCUDA=true)
Select the first GPU whose name or vendor contains the substring (case-insensitive).
- bool **selectByCudaIndex** (int cudaIndex)
Select by CUDA device ordinal (as discovered during probing).
- bool **selectByOpenCL** (int platformIndex, int deviceIndex)
Select by OpenCL (platformIndex, deviceIndex) pair.

- void **clearSelection** ()
Clear selection to "none".
- void **setSelectedOpenGLVersion** (const std::string &glVersion)
Override OpenGL version string for the selected GPU (when you probe in your own GL context).
- std::string **getSelectedOpenGLVersion** () const
Get OpenGL version string for the selected GPU (empty if unknown).
- [SupportState](#) **selectedCudaState** () const
Summaries for the selected GPU (Unknown if none).

3.11.1 Detailed Description

GPU environment probe and selection (non-singleton).

Call [Refresh](#) to fully rescan the system and update states. Selection APIs set a best-effort `selected_` index for convenience.

Definition at line 70 of file [ClpmGpuEnv.h](#).

3.11.2 Member Function Documentation

3.11.2.1 selectByNameSubstring()

```
bool ipm::ClpmGpuEnv::selectByNameSubstring (
    const std::string & substr,
    bool preferCUDA = true)
```

Select the first GPU whose name or vendor contains the substring (case-insensitive).

Parameters

<i>substr</i>	Key to search in name or vendor.
<i>preferCUDA</i>	If true, prefer a CUDA-capable candidate when multiple match.

Returns

true if a selection was made.

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

- ImageProcessorManager/[ClpmGpuEnv.h](#)

3.12 csh_img::CSH_Image Class Reference

Image container with explicit format metadata and flexible buffer ownership.

```
#include <CSH_Image.h>
```

Public Types

- using **byte** = uint8_t
Unsigned 8-bit byte alias for buffer access.

Public Member Functions

- **CSH_Image** ()
Constructs an empty, disabled image (no buffer).
- **CSH_Image** (uint32_t width, uint32_t height, En_ImageFormat format, bool alloc_mem=true, uint32_t image_count=1)
Constructs an image with metadata and optional allocation.
- **CSH_Image** (const CSH_Image &)=default
Shallow copy (shared buffer).
- **CSH_Image** (CSH_Image &&) noexcept=default
Move constructor.
- **CSH_Image** & **operator=** (const CSH_Image &)=default
Shallow assignment (shared buffer).
- **CSH_Image** & **operator=** (CSH_Image &&) noexcept=default
Move assignment.
- void **copy** (const CSH_Image &src, CopyMode mode)
Copies metadata and optionally buffer depending on mode.
- void **copyBufferPointer** (const CSH_Image &src)
Shares the underlying buffer pointer/view with src (shallow ownership).
- void **copyBufferPointer** (byte *pFrame)
Adopts an external raw pointer as the buffer (shallow, no delete).
- void **saveImage** (const std::filesystem::path &filepath) const
Saves the image (header + fields + optional bytes) to a file.
- void **loadImage** (const std::filesystem::path &filepath)
Loads an image from a TLV file written by saveImage().
- uint32_t **getWidth** () const
- uint32_t **getHeight** () const
- bool **isEnabled** () const
- uint32_t **getCameraId** () const
- En_ImageFormat **getFormat** () const
- uint32_t **getMemoryBit** () const
- uint32_t **getOriginalBit** () const
- En_ImagePattern **getPattern** () const
- En_ImageMemoryAlign **getMemoryAlign** () const
- std::size_t **getBufferSize** () const
- uint32_t **getImageCount** () const
- uint32_t **getSelectedImage** () const
- byte * **data** ()
Returns a pointer to the current view (selected image).
- const byte * **data** () const
Returns a const pointer to the current view (selected image).
- byte * **getImagePtr** (uint32_t n)
Returns the base pointer to the n-th image (0-based) without changing state.
- const byte * **getImagePtr** (uint32_t n) const
Returns the base pointer to the n-th image (0-based) without changing state.
- void **setSelectedImage** (uint32_t idx)

-
- Selects the active image index for the view.*
 - `std::size_t totalBytes () const`
Logical total bytes = per-frame bytes image count.
 - `void recomputeBufferSize ()`
Recomputes `buffer_size` based on format, width, height, and bit depth.
 - `void allocateBuffer ()`
Allocates exactly `totalBytes()` and updates the current view.

Backward-compatible overloads

Public Attributes

- `uint32_t width = 0`
Image width in pixels.
- `uint32_t height = 0`
Image height in pixels.
- `bool bEnable = false`
Logical enabled flag.
- `uint32_t camera_id = 0`
User-defined camera identifier.
- `En_ImageFormat format = En_ImageFormat::Gray8`
Current image format.
- `uint32_t memory_bit = 8`
Memory container bit depth.
- `uint32_t original_bit = 8`
Original sensor bit depth.
- `En_ImagePattern pattern = En_ImagePattern::RGGB`
Pixel/component layout.
- `En_ImageMemoryAlign memory_align = En_ImageMemoryAlign::Packed`
Memory layout.
- `std::size_t buffer_size = 0`
Per-frame byte size.
- `uint32_t image_count = 1`
Number of images in allocation.
- `uint32_t sel_image = 0`
Currently selected image index.
- `std::shared_ptr< byte[] > buffer`
Base shared buffer pointer (may be null).

3.12.1 Detailed Description

Image container with explicit format metadata and flexible buffer ownership.

3.12.1.0.1 Key properties

- **Ownership:** buffer is stored as `std::shared_ptr<byte[]>`. Shallow copies share the buffer; deep copies require a pre-allocated destination.
- **View:** `buffer_offset` + `sel_image` define the current view inside a multi-image allocation (e.g., frame arrays).
- **Persistence:** TLV-based binary I/O (`saveImage/loadImage`) with a stable header.
- **Interoperability:** Optional zero-copy interop with OpenCV (when `CSH_IMAGE_WITH_OPENCV`).

Warning

Deep copies require the destination to have an allocated buffer with enough writable bytes from the current view; otherwise an exception is thrown.

Definition at line 129 of file `CSH_Image.h`.

3.12.2 Constructor & Destructor Documentation

3.12.2.1 CSH_Image()

```
csh_img::CSH_Image::CSH_Image (  
    uint32_t width,  
    uint32_t height,  
    En_ImageFormat format,  
    bool alloc_mem = true,  
    uint32_t image_count = 1)
```

Constructs an image with metadata and optional allocation.

Parameters

<i>width</i>	Image width in pixels.
<i>height</i>	Image height in pixels.
<i>format</i>	Pixel/container format.
<i>alloc_mem</i>	If true, allocate a buffer sized to <code>width*height*bpp*image_count</code> .
<i>image_count</i>	Number of images (frames) in the allocation.

Note

The per-frame byte size is computed by `recomputeBufferSize()`.

References `format`, `height`, `image_count`, and `width`.

3.12.3 Member Function Documentation

3.12.3.1 allocateBuffer()

```
void csh_img::CSH_Image::allocateBuffer ()
```

Allocates exactly `totalBytes()` and updates the current view.

Exceptions

<code>std::runtime_error</code>	If <code>buffer_size</code> or <code>image_count</code> is zero.
---------------------------------	------------------------------------------------------------------

On success, sets `buffer_capacity_bytes`, ensures `sel_image` is in range, and updates `buffer_offset` accordingly.

3.12.3.2 `copy()`

```
void csh_img::CSH_Image::copy (  
    const CSH_Image & src,  
    CopyMode mode)
```

Copies metadata and optionally buffer depending on `mode`.

- `CopyMode::MetaOnly` : copies metadata; resets buffer.
- `CopyMode::Shallow` : shares `src` buffer; adopts capacity info if known.
- `CopyMode::Deep` : copies bytes into *existing* destination buffer.

Parameters

<code>src</code>	Source image.
<code>mode</code>	Copy behavior.

Exceptions

<code>std::runtime_error</code>	If <code>mode</code> is <code>Deep</code> and the destination has no buffer or insufficient writable bytes from current view.
<code>std::invalid_argument</code>	If <code>mode</code> is unknown.

References `copy()`, and `CSH_Image()`.

Referenced by `copy()`.

3.12.3.3 `copyBufferPointer()` [1/2]

```
void csh_img::CSH_Image::copyBufferPointer (  
    byte * pFrame)
```

Adopts an external raw pointer as the buffer (shallow, no delete).

Parameters

<code>pFrame</code>	Raw pointer to externally-managed memory.
---------------------	-------------------------------------------

Exceptions

<code>std::invalid_argument</code>	If <code>pFrame</code> is <code>nullptr</code> .
------------------------------------	--------------------------------------------------

Warning

The memory is **not** freed by this class. Ensure the lifetime of `pFrame` outlives this image or any shared copies.

References [copyBufferPointer\(\)](#).

3.12.3.4 copyBufferPointer() [2/2]

```
void csh_img::CSH_Image::copyBufferPointer (
    const CSH_Image & src)
```

Shares the underlying buffer pointer/view with `src` (shallow ownership).

Parameters

<code>src</code>	Source image (its buffer/shared_ptr is adopted).
------------------	--------------------------------------------------

References [copyBufferPointer\(\)](#), and [CSH_Image\(\)](#).

Referenced by [copyBufferPointer\(\)](#), and [copyBufferPointer\(\)](#).

3.12.3.5 data() [1/2]

```
byte * csh_img::CSH_Image::data () [inline]
```

Returns a pointer to the current view (selected image).

Returns

Writable pointer or `nullptr` if no buffer.

Definition at line 260 of file [CSH_Image.h](#).

```
00260 { return buffer ? (buffer.get() + buffer_offset) : nullptr; }
```

References [buffer](#).

Referenced by [cimage::CImageDisplayerCPP::setImage\(\)](#).

3.12.3.6 data() [2/2]

```
const byte * csh_img::CSH_Image::data () const [inline]
```

Returns a const pointer to the current view (selected image).

Returns

Read-only pointer or `nullptr` if no buffer.

Definition at line 266 of file [CSH_Image.h](#).

```
00266 { return buffer ? (buffer.get() + buffer_offset) : nullptr; }
```

References [buffer](#).

3.12.3.7 `getBufferSize()`

```
std::size_t csh_img::CSH_Image::getBufferSize () const [inline]
```

Returns

Per-frame byte size.

Definition at line 250 of file [CSH_Image.h](#).

```
00250 { return buffer_size; }
```

References [buffer_size](#).

Referenced by [cimage::CImageDisplayerCPP::setImage\(\)](#).

3.12.3.8 `getCameraId()`

```
uint32_t csh_img::CSH_Image::getCameraId () const [inline]
```

Returns

Associated camera identifier (user-defined).

Definition at line 238 of file [CSH_Image.h](#).

```
00238 { return camera_id; }
```

References [camera_id](#).

3.12.3.9 `getFormat()`

```
En_ImageFormat csh_img::CSH_Image::getFormat () const [inline]
```

Returns

Current image format.

Definition at line 240 of file [CSH_Image.h](#).

```
00240 { return format; }
```

References [format](#).

Referenced by [cimage::CImageDisplayerCPP::setImage\(\)](#).

3.12.3.10 `getHeight()`

```
uint32_t csh_img::CSH_Image::getHeight () const [inline]
```

Returns

Image height in pixels.

Definition at line 234 of file [CSH_Image.h](#).

```
00234 { return height; }
```

References [height](#).

Referenced by [cimage::CImageDisplayerCPP::setImage\(\)](#).

3.12.3.11 getImageCount()

```
uint32_t csh_img::CSH_Image::getImageCount () const [inline]
```

Returns

Number of images in the allocation.

Definition at line 252 of file [CSH_Image.h](#).

```
00252 { return image_count; }
```

References [image_count](#).

3.12.3.12 getImagePtr() [1/2]

```
byte * csh_img::CSH_Image::getImagePtr (
    uint32_t n)
```

Returns the base pointer to the n-th image (0-based) without changing state.

Parameters

<i>n</i>	Image index.
----------	--------------

Returns

Pointer to the start of image *n* or `nullptr` if no buffer.

Exceptions

<code>std::out_of_range</code>	If <i>n</i> >= image_count .
--------------------------------	----------------------------------------------

3.12.3.13 getImagePtr() [2/2]

```
const byte * csh_img::CSH_Image::getImagePtr (
    uint32_t n) const
```

Returns the base pointer to the n-th image (0-based) without changing state.

Parameters

<i>n</i>	Image index.
----------	--------------

Returns

Pointer to the start of image *n* or `nullptr` if no buffer.

Exceptions

<code>std::out_of_range</code>	<code>if n >= image_count.</code>
--------------------------------	--------------------------------------

3.12.3.14 `getMemoryAlign()`

```
En_ImageMemoryAlign csh_img::CSH_Image::getMemoryAlign () const [inline]
```

Returns

Memory alignment/plane arrangement.

Definition at line 248 of file [CSH_Image.h](#).

```
00248 { return memory_align; }
```

References [memory_align](#).

Referenced by [cimage::CImageDisplayerCPP::setImage\(\)](#).

3.12.3.15 `getMemoryBit()`

```
uint32_t csh_img::CSH_Image::getMemoryBit () const [inline]
```

Returns

Container bit depth (memory).

Definition at line 242 of file [CSH_Image.h](#).

```
00242 { return memory_bit; }
```

References [memory_bit](#).

3.12.3.16 `getOriginalBit()`

```
uint32_t csh_img::CSH_Image::getOriginalBit () const [inline]
```

Returns

Original sensor bit depth (semantic).

Definition at line 244 of file [CSH_Image.h](#).

```
00244 { return original_bit; }
```

References [original_bit](#).

3.12.3.17 `getPattern()`

```
En_ImagePattern csh_img::CSH_Image::getPattern () const [inline]
```

Returns

Pixel pattern (CFA/YUV order/channel order).

Definition at line 246 of file [CSH_Image.h](#).

```
00246 { return pattern; }
```

References [pattern](#).

Referenced by [cimage::CImageDisplayerCPP::setImage\(\)](#).

3.12.3.18 `getSelectedImage()`

```
uint32_t csh_img::CSH_Image::getSelectedImage () const [inline]
```

Returns

Currently selected image index (0-based).

Definition at line 254 of file [CSH_Image.h](#).

```
00254 { return sel_image; }
```

References [sel_image](#).

3.12.3.19 `getWidth()`

```
uint32_t csh_img::CSH_Image::getWidth () const [inline]
```

Returns

Image width in pixels.

Definition at line 232 of file [CSH_Image.h](#).

```
00232 { return width; }
```

References [width](#).

Referenced by [cimage::CImageDisplayerCPP::setImage\(\)](#).

3.12.3.20 `isEnabled()`

```
bool csh_img::CSH_Image::isEnabled () const [inline]
```

Returns

Whether the image is logically enabled/valid.

Definition at line 236 of file [CSH_Image.h](#).

```
00236 { return bEnable; }
```

References [bEnable](#).

3.12.3.21 `loadImage()`

```
void csh_img::CSH_Image::loadImage (  
    const std::filesystem::path & filepath)
```

Loads an image from a TLV file written by [saveImage\(\)](#).

Parameters

<i>filepath</i>	Source path.
-----------------	--------------

Exceptions

<i>std::runtime_error</i>	On I/O errors, bad magic/version, or allocation failures.
---------------------------	-----------------------------------------------------------

References [loadImage\(\)](#), and [saveImage\(\)](#).

Referenced by [loadImage\(\)](#).

3.12.3.22 **recomputeBufferSize()**

```
void csh_img::CSH_Image::recomputeBufferSize ()
```

Recomputes [buffer_size](#) based on format, width, height, and bit depth.

Uses bytesPerPixelForFormat() when known; otherwise falls back to `ceil(memory_bit/8)` bytes per pixel.

3.12.3.23 **saveImage()**

```
void csh_img::CSH_Image::saveImage (
    const std::filesystem::path & filepath) const
```

Saves the image (header + fields + optional bytes) to a file.

Parameters

<i>filepath</i>	Destination path.
-----------------	-------------------

Exceptions

<i>std::runtime_error</i>	On I/O errors or when the buffer is missing but required.
---------------------------	-----------------------------------------------------------

Format: Magic (CHSI) + Version + field count + {TLV...}. The buffer is written as a single field when present. All integers are little-endian.

References [saveImage\(\)](#).

Referenced by [loadImage\(\)](#), and [saveImage\(\)](#).

3.12.3.24 **setSelectedImage()**

```
void csh_img::CSH_Image::setSelectedImage (
    uint32_t idx)
```

Selects the active image index for the view.

Parameters

<i>idx</i>	New index (0-based).
------------	----------------------

Exceptions

<i>std::out_of_range</i>	If <code>idx >= image_count</code> or if the computed offset exceeds capacity.
--------------------------	-----------------------------------------------------------------------------------

3.12.3.25 totalBytes()

```
std::size_t csh_img::CSH_Image::totalBytes () const [inline]
```

Logical total bytes = per-frame bytes * image count.

Returns

Total logical size of the allocation.

Definition at line 293 of file [CSH_Image.h](#).

```
00293 { return buffer_size * static_cast<std::size_t>(image_count); }
```

References [buffer_size](#), and [image_count](#).

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

- [utility/CSH_Image/CSH_Image.h](#)

3.13 CSH_Log Class Reference

Thread-safe, Unicode-aware singleton logger.

```
#include <CSH_Log.h>
```

3.13.1 Detailed Description

Thread-safe, Unicode-aware singleton logger.

The logger writes UTF-8 encoded lines to a rolling log file whose name is derived from the current timestamp (e.g., `YYYYMMDD_HHMMSS.log`).

Thread safety

All public member functions are thread-safe. Internally, file writes are serialized with a mutex; configuration flags are stored in atomics.

Encoding

Public logging APIs accept wide strings (`std::wstring`) or printf-style wide format strings (`const wchar_t*`). Lines are written to disk as UTF-8.

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

- [utility/SH_Log/CSH_Log.h](#)
-

3.14 CWatchTime Class Reference

RAII-less stopwatch using `high_resolution_clock` with simple elapsed queries.

```
#include <CWatchTime.h>
```

Public Types

- using **Clock** = `std::chrono::high_resolution_clock`
High-resolution clock type used for measurements.
- using **TimePoint** = `Clock::time_point`
Time point type from [Clock](#).

Public Member Functions

- [CWatchTime](#) ()
Construct an idle stopwatch.
- void [start](#) ()
Begin (or restart) timing.
- void [stop](#) ()
Stop timing.
- double [GetSecond](#) ()
Get elapsed time in seconds.
- double [GetMilliSecond](#) ()
Get elapsed time in milliseconds.
- double [GetMicroSecond](#) ()
Get elapsed time in microseconds.
- const `wchar_t*` [getString](#) () const
Format the current elapsed time as a wide string like `L"123.456ms"`.
- const `wchar_t*` [GetCurrentTimeString](#) ()
Get the current local time as a wide string in the form `YYYY-MM-DD HH:MM:SS`.
- const `char*` [GetCurrentTimeStringA](#) ()
Get the current local time as a narrow string in the form `YYYY-MM-DD HH:MM:SS.mmm`.

3.14.1 Detailed Description

RAII-less stopwatch using `high_resolution_clock` with simple elapsed queries.

Typical usage:

```
CWatchTime wt;  
wt.start();  
// ... work ...  
wt.stop();  
double ms = wt.GetMilliSecond();           // elapsed milliseconds  
const wchar_t* label = wt.getString();      // L"123.456ms"  
const char* now = wt.GetCurrentTimeStringA(); // "YYYY-MM-DD HH:MM:SS.mmm"
```

Warning

Not thread-safe. Do not share a single instance across threads without external synchronization.

String-returning methods return pointers to internal buffers whose contents are overwritten on subsequent calls.

Definition at line 46 of file [CWatchTime.h](#).

3.14.2 Constructor & Destructor Documentation

3.14.2.1 CWatchTime()

```
CWatchTime::CWatchTime ()
```

Construct an idle stopwatch.

The stopwatch starts in a non-running state. Call [start\(\)](#) to begin measuring.

3.14.3 Member Function Documentation

3.14.3.1 GetCurrentTimeString()

```
const wchar_t * CWatchTime::GetCurrentTimeString ()
```

Get the current local time as a wide string in the form YYYY-MM-DD HH:MM:SS.

Returns

Pointer to an internal static wide buffer.

Note

Uses a process-wide static buffer. Subsequent calls overwrite the buffer.

Thread-safety: The buffer is shared across all instances; concurrent calls from multiple threads will race.

3.14.3.2 GetCurrentTimeStringA()

```
const char * CWatchTime::GetCurrentTimeStringA ()
```

Get the current local time as a narrow string in the form YYYY-MM-DD HH:MM:SS.mmm.

Returns

Pointer to an internal static char buffer.

Note

Uses a process-wide static buffer. Subsequent calls overwrite the buffer.

Thread-safety: The buffer is shared across all instances; concurrent calls from multiple threads will race.

3.14.3.3 GetMicroSecond()

```
double CWatchTime::GetMicroSecond ()
```

Get elapsed time in microseconds.

Returns

Elapsed microseconds as a floating-point value.

If the stopwatch is running, computes duration from start to "now"; otherwise, uses the captured end timestamp.

3.14.3.4 GetMilliSecond()

```
double CWatchTime::GetMilliSecond ()
```

Get elapsed time in milliseconds.

Returns

Elapsed milliseconds as a floating-point value.

If the stopwatch is running, computes duration from start to "now"; otherwise, uses the captured end timestamp.

3.14.3.5 GetSecond()

```
double CWatchTime::GetSecond ()
```

Get elapsed time in seconds.

Returns

Elapsed seconds as a floating-point value.

If the stopwatch is running, computes duration from start to "now"; otherwise, uses the captured end timestamp.

3.14.3.6 getString()

```
const wchar_t * CWatchTime::getString () const
```

Format the current elapsed time as a wide string like `L"123.456ms"`.

Returns

Pointer to an internal wide buffer containing the formatted string.

Note

The returned pointer remains valid until the next call to [getString\(\)](#) on this instance. Copy the contents if you need to keep it.

3.14.3.7 start()

```
void CWatchTime::start ()
```

Begin (or restart) timing.

Sets the start timestamp to now and marks the stopwatch as running. Calling [start\(\)](#) again restarts the measurement.

3.14.3.8 stop()

```
void CWatchTime::stop ()
```

Stop timing.

Captures the end timestamp and marks the stopwatch as not running. Elapsed queries will use the captured end time until [start\(\)](#) is called again.

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

- [utility/CWatchTime/CWatchTime.h](#)

3.15 FuncInfo Struct Reference

Metadata for a registered algorithm.

```
#include <IpmTypes.h>
```

Public Attributes

- [IpmFn](#) **fn**
Callable entry point (may be empty prior to registration).
- `std::wstring` **uiName**
Display name for UI lists (UTF-16).

3.15.1 Detailed Description

Metadata for a registered algorithm.

See also

[ClpmFuncTable](#)

Definition at line 99 of file [IpmTypes.h](#).

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

- [ImageProcessorManager/IpmTypes.h](#)

3.16 ipm::GpuInfo Struct Reference

One GPU record with runtime capability flags.

```
#include <CIpmGpuEnv.h>
```

Public Attributes

- `int id = -1`
Internal index within the enumerated list.
- `std::string name`
Adapter/device name.
- `std::string vendor`
"NVIDIA", "Intel", "AMD", or "Unknown".

3.16.1 Detailed Description

One GPU record with runtime capability flags.

Notes:

- `cudaVersion` is a driver version string (e.g., "Driver 12070") if CUDA Driver API is present.
- `openglVersion` is the platform version string for the enumerated ICD.
- `openglVersion` is set when the optional GL probe succeeds.

Definition at line 45 of file [ClpmGpuEnv.h](#).

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

- ImageProcessorManager/[ClpmGpuEnv.h](#)

3.17 cimage::Mat4 Struct Reference

Column-major 4x4 matrix (OpenGL style).

```
#include <CImageDisplay.h>
```

Static Public Member Functions

- static [Mat4](#) `identity` ()
Identity matrix.

3.17.1 Detailed Description

Column-major 4x4 matrix (OpenGL style).

Definition at line 92 of file [CImageDisplay.h](#).

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

- CImageDisplay/[CImageDisplay.h](#)

3.18 cimage::Quat Struct Reference

Quaternion (w + xyz). Identity by default.

```
#include <CImageDisplay.h>
```

3.18.1 Detailed Description

Quaternion (w + xyz). Identity by default.

Definition at line 87 of file [CImageDisplay.h](#).

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

- [CImageDisplay/CImageDisplay.h](#)

3.19 cimage::UploadDescriptor Struct Reference

Description of the memory block to upload to a GPU/renderer.

```
#include <CImageDisplay.h>
```

Public Attributes

- `const std::uint8_t * data = nullptr`
Pointer to first byte (may be null for MetaOnly).
- `std::size_t sizeBytes = 0`
Total buffer size in bytes.
- `int height = 0`
Image dimensions in pixels.
- `int bytesPerPixel = 0`
Bytes per pixel for tightly packed data.
- `int strideBytes = 0`
*Row stride in bytes (0 => tightly packed width*bpp).*
- `PixelFormat layout = PixelLayout::Unknown`
Pixel layout.
- `Yuv422Pattern yuv422Pattern = Yuv422Pattern::YUYV`
YUV422 order if applicable.
- `bool isPacked = true`
True if contiguous packed memory.
- `bool isLittleEndian16 = true`
True if 16-bit samples are little-endian.

3.19.1 Detailed Description

Description of the memory block to upload to a GPU/renderer.

Note

Values are derived from `csd_img::CSH_Image` via `ClImageDisplayer::uploadDesc`.

Definition at line 68 of file `ClImageDisplayer.h`.

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

- `ClImageDisplayer/ClImageDisplayer.h`

3.20 ipm_internal::UserCustomLoader Class Reference

Singleton user plug-in loader for the User_Custom module.

```
#include <CIpmUserCustomLoader.h>
```

Public Member Functions

- `int loadOnce ()`
Attempt to load and register plug-ins exactly once.
- `const std::vector< AlgEntry > & entries () const`
Access registered entries (valid after successful `loadOnce`).
- `void unload ()`
Unregister and unload the plug-in module (safe during process teardown).

Static Public Member Functions

- `static UserCustomLoader & instance ()`
Get the singleton instance.

3.20.1 Detailed Description

Singleton user plug-in loader for the User_Custom module.

After a successful load, call `entries` to retrieve the list of plug-in algorithms. The function table merges these entries into its own registry.

See also

`ipmcommon::ClpmFuncTable`

Definition at line 36 of file `ClpmUserCustomLoader.h`.

3.20.2 Member Function Documentation

3.20.2.1 loadOnce()

```
int ipm_internal::UserCustomLoader::loadOnce ()
```

Attempt to load and register plug-ins exactly once.

Returns

The number of entries registered (≥ 0).

On success, the internal `entries_` vector becomes populated and remains valid until [unload](#).

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

- ImageProcessorManager/[ClpmUserCustomLoader.h](#)

3.21 cimage::Vec2 Struct Reference

2D vector.

```
#include <CImageDisplay.h>
```

3.21.1 Detailed Description

2D vector.

Definition at line 83 of file [CImageDisplay.h](#).

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

- CImageDisplay/[CImageDisplay.h](#)

3.22 cimage::Vec3 Struct Reference

3D vector.

```
#include <CImageDisplay.h>
```

3.22.1 Detailed Description

3D vector.

Definition at line 85 of file [CImageDisplay.h](#).

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

- CImageDisplay/[CImageDisplay.h](#)

Chapter 4

File Documentation

4.1 CFrameGrabber.h

```
00001 #pragma once
00002 #include <memory>
00003 #include <vector>
00004 #include <string>
00005 #include <mutex>
00006 #include "IFrameGrabImpl.h"
00007
00008 // ----- Export macro -----
00009 #pragma once
00010 #if defined(_WIN32) || defined(_WIN64)
00011 #ifdef FRAMEGRAB_BUILD
00012 #define FRAMEGRAB_API __declspec(dllexport)
00013 #else
00014 #define FRAMEGRAB_API __declspec(dllimport)
00015 #endif
00016 #else
00017 #define FRAMEGRAB_API __attribute__((visibility("default")))
00018 #endif
00019
00035 class FRAMEGRAB_API CFrameGrabber {
00036 public:
00041     enum class En_GrabberBackend { UVC = 0, V4L2, gStreamer, Count };
00042
00044     CFrameGrabber();
00045
00047     ~CFrameGrabber();
00048
00055     bool SetBackend(En_GrabberBackend be);
00056
00063     bool GetConnected(int& outDeviceCount, std::vector<std::string>& outModelNames);
00064
00070     bool Connect();
00071
00076     void Disconnect();
00077
00085     bool SetConfig(const CGrabberConfig* cfg);
00086
00092     bool GrabFrames();
00093
00098     void StopGrabbing();
00099
00105     void RegisterCallbackProcessor(FrameGrabCallbackProc cb);
00106
00112     void RegisterCallbackDisplayer(FrameGrabCallbackDisp cb);
00113
00119     bool SetSensorRegister(uint32_t address, uint32_t value);
00120
00126     bool GetSensorRegister(uint32_t address, uint32_t& outValue);
00127
00129     int deviceCount() const { return deviceCount_; }
00130
00132     const std::vector<std::string>& modelNames() const { return modelNames_; }
00133
00135     bool IsConnecting() const { return isConnecting_; }
00136
00138     bool IsGrabbing() const { return isGrabbing_; }
00139
```

```

00140 private:
00142     std::unique_ptr<IFrameGrabImpl> impl_;
00143
00144     // Cached probe results
00145     int deviceCount_ = 0;
00146     std::vector<std::string> modelNames_;
00147
00148     // Simple state flags (best-effort)
00149     bool isConnecting_ = false;
00150     bool isGrabbing_ = false;
00151
00152     // Serialize public API; backends have their own internal sync.
00153     std::mutex mtx_;
00154 };

```

4.2 CFrameGrabber/CGrabberConfig.h File Reference

Basic runtime configuration for a frame grabber backend.

```

#include <string>
#include <cstdint>
#include <CSH_Image.h>

```

4.2.1 Detailed Description

Basic runtime configuration for a frame grabber backend.

This struct carries the desired device selection (video/subdev), frame geometry, frame rate, and pixel format. Backends should interpret these as *requests* and may clamp/adjust to the closest supported mode.

Note

Defaults are safe 640x480 @ 30fps RGB24.

See also

`IFrameGrabImpl::SetConfig`

Definition in file [CGrabberConfig.h](#).

4.3 CGrabberConfig.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include <string>
00003 #include <cstdint>
00004 #include <CSH_Image.h>
00005
00017 struct CGrabberConfig {
00025     enum class PixelFormat : uint32_t {
00026         UNKNOWN = 0,
00027         GRAY8,
00028         RGB24,
00029         BGR24,
00030         YUYV422,
00031         UYVY422
00032     };
00033 };

```

```

00039     int    video_id = -1;
00040
00045     int    subdev_id = -1;
00046
00051     std::string strVideo;
00052
00057     std::string strSubdev;
00058
00063     std::string strGrabberName;
00064
00066     uint32_t width = 640;
00067
00069     uint32_t height = 480;
00070
00072     uint32_t fps = 30;
00073
00075     PixelFormat pixel_format = PixelFormat::RGB24;
00076 };

```

4.4 IFrameGrabImpl.h

```

00001 #pragma once
00002 #include <functional>
00003 #include <vector>
00004 #include <string>
00005 #include <atomic>
00006 #include <thread>
00007 #include <memory>
00008 #include "CSH_Image.h"
00009 #include "CGrabberConfig.h"
00010
00011 // ----- Export macro -----
00012 #pragma once
00013 #if defined(_WIN32) || defined(_WIN64)
00014 #ifdef FRAMEGRAB_BUILD
00015 #define FRAMEGRAB_API __declspec(dllexport)
00016 #else
00017 #define FRAMEGRAB_API __declspec(dllimport)
00018 #endif
00019 #else
00020 #define FRAMEGRAB_API __attribute__((visibility("default")))
00021 #endif
00022
00028 using FrameGrabCallbackProc = std::function<void(const csh_img::CSH_Image&);>;
00029
00035 using FrameGrabCallbackDisp = std::function<void(const csh_img::CSH_Image&);>;
00036
00052 class FRAMEGRAB_API IFrameGrabImpl {
00053 public:
00054     virtual ~IFrameGrabImpl() = default;
00055
00063     virtual bool GetConnected(int& outDeviceCount, std::vector<std::string>& outModelNames) = 0;
00064
00071     virtual bool Connect() = 0;
00072
00077     virtual void Disconnect() = 0;
00078
00087     virtual bool SetConfig(const CGrabberConfig* cfg) = 0;
00088
00094     virtual bool GrabFrames() = 0;
00095
00100     virtual void StopGrabbing() = 0;
00101
00107     virtual void RegisterCallbackProcessor(FrameGrabCallbackProc cb) = 0;
00108
00114     virtual void RegisterCallbackDisplayer(FrameGrabCallbackDisp cb) = 0;
00115
00123     virtual bool SetSensorRegister(uint32_t address, uint32_t value) = 0;
00124
00132     virtual bool GetSensorRegister(uint32_t address, uint32_t& outValue) = 0;
00133
00138     const CGrabberConfig& Config() const { return grabberConfig; }
00139
00140 protected:
00142     CGrabberConfig grabberConfig{};
00143 };

```

4.5 CImageDisplay/CImageDisplay.h File Reference

Math-only image “displayer” that owns an image buffer and exposes 2D/3D transform state and upload descriptors. No rendering is performed.

```
#include <array>
#include <stdint>
#include <stddef>
#include "CSH_Image.h"
```

Classes

- struct [cimage::UploadDescriptor](#)
Description of the memory block to upload to a GPU/renderer.
- struct [cimage::Vec2](#)
2D vector.
- struct [cimage::Vec3](#)
3D vector.
- struct [cimage::Quat](#)
Quaternion (w + xyz). Identity by default.
- struct [cimage::Mat4](#)
Column-major 4x4 matrix (OpenGL style).

Typedefs

- using **cimage::Mat3** = std::array<float, 9>
Row-major 3x3 (for 2D APIs).

Enumerations

- enum class [cimage::PixelFormat](#) : uint32_t {
 [Unknown](#) = 0 , [Gray8](#) , [RGB888](#) , [BGR888](#) ,
 [YUV422Packed](#) , [RGB565](#) , [Gray16](#) , [Bayer16](#) }
High-level pixel layout of the buffer to upload.
- enum class [cimage::Yuv422Pattern](#) : uint32_t
YUV422 packing order for packed layouts.
- enum class [cimage::Dimensionality](#) : uint32_t
Viewing dimensionality.
- enum class [cimage::FitMode](#) : uint32_t
Fit/Fill strategy into viewport.
- enum class [cimage::OrbitStyle](#) : uint32_t
Orbit interaction style for 3D rotation.
- enum class [cimage::MouseButton](#) : uint32_t
Mouse button bitmask (can be combined).
- enum class [cimage::KeyMod](#) : uint32_t
Keyboard modifier bitmask (can be combined).

Functions

- `KeyMod cimage::operator| (KeyMod a, KeyMod b)`
Bitwise OR for key modifiers.
- class `cimage::__attribute__` ((visibility("default"))) `CImageDisplayer`

Variables

- constexpr `Mat3 cimage::kMat3Identity { 1,0,0, 0,1,0, 0,0,1 }`
Identity 3x3.

4.5.1 Detailed Description

Math-only image “displayer” that owns an image buffer and exposes 2D/3D transform state and upload descriptors. No rendering is performed.

Typical use: 1) Provide image data via `cimage::CImageDisplayer::setImage` or `setImageRaw`. 2) Configure view-port/fit and 2D/3D transforms. 3) Fetch matrices, geometry and `uploadDesc` for your renderer.

Threading: this type is not thread-safe. Call from a single UI/render thread.

Definition in file [CImageDisplayer.h](#).

4.5.2 Enumeration Type Documentation

4.5.2.1 PixelLayout

```
enum class cimage::PixelLayout : uint32_t [strong]
```

High-level pixel layout of the buffer to upload.

Enumerator

Unknown	Unspecified layout.
Gray8	8-bit grayscale.
RGB888	Interleaved 8-bit RGB.
BGR888	Interleaved 8-bit BGR.
YUV422Packed	Packed YUV 4:2:2 (pattern indicated separately).
RGB565	16-bit packed RGB 5:6:5.
Gray16	16-bit grayscale.
Bayer16	16-bit Bayer (10/12/14/16 packed in 16).

Definition at line 48 of file [CImageDisplayer.h](#).

```
00048                                     : uint32_t {
00049     Unknown = 0,
00050     Gray8,
00051     RGB888,
00052     BGR888,
00053     YUV422Packed,
00054     RGB565,
00055     Gray16,
00056     Bayer16
00057 };
```

4.5.3 Function Documentation

4.5.3.1 __attribute__()

```
class cimage::__attribute__ (
    (visibility("default")) )
```

Construct with empty image and identity transforms.

Set image from an existing [csh_img::CSH_Image](#).

Parameters

in	<i>img</i>	Source image.
in	<i>mode</i>	Deep, Shallow, or MetaOnly copy semantics.

Note

Shallow retains caller's buffer; caller must keep it alive.

Set image from a foreign raw buffer with explicit metadata.

Parameters

in	<i>w,h</i>	Dimensions in pixels.
in	<i>fmt</i>	Image format.
in	<i>pat</i>	Pixel pattern (RGB/BGR/YUV422 order, etc.).
in	<i>align</i>	Memory alignment/packing.
in	<i>pixels</i>	Pointer to source bytes (nullable for MetaOnly).
in	<i>bytes</i>	Size of <i>pixels</i> in bytes.
in	<i>mode</i>	Deep/Shallow/MetaOnly semantics.

Note

Shallow uses a no-op deleter; caller retains ownership.

Allocate (or reallocate) an internal image buffer.

Parameters

<i>w,h</i>	Dimensions in pixels.
<i>fmt</i>	Desired format.
<i>count</i>	Image count (defaults to 1).

Postcondition

Internal buffer is owned by this object.

Immutable access to the current image.

Mutable access to the current image.

Set viewport size in pixels (non-negative).

Returns

Current viewport width in pixels.

Current viewport height in pixels.

Set fit strategy (None/Fit/Fill/Stretch).

Returns

Current fit strategy.

Switch between 2D and 3D modes.

Returns

Current dimensionality.

Set normalized anchor inside image rect $[0..1]^2$.

Set 2D translation in viewport pixels.

Set 2D scale factors (multiplicative).

Set rotation in degrees (counter-clockwise).

Reset 2D transform to identity (anchor center).

Set model translation.

Set model scale.

Set model rotation (quaternion).

Reset model transform to identity.

Set look-at target point.

Set camera eye position.

Set camera up vector (normalized internally).

Set orbit interaction style (Arcball/Turntable).

Set orthographic projection (OpenGL style).

Set perspective projection (fovy in degrees).

Returns

True if orthographic projection is active.

Current 2D model matrix (row-major 3×3).

Current 3D model matrix (column-major 4×4).

Current 3D view matrix (column-major 4×4).

Current projection matrix (column-major 4×4).

Convenience $P*V*M$ for 3D (column-major 4×4).

Transformed quad as 2D tri-strip with UVs.

Returns

TL,TR,BL,BR each as {x,y,u,v}.

Unit quad in object space for 3D pipelines.

Returns

TL,TR,BL,BR each as {x,y,u,v} with x/y in [-0.5,+0.5].

Produce an upload descriptor reflecting the current image and format.

Returns

UploadDescriptor with pointer/size/stride/layout info.

Begin a pointer interaction (mouse/touch).

Parameters

<i>x,y</i>	Cursor position in viewport pixels.
<i>btn</i>	Button mask.
<i>mods</i>	Key modifiers.

Update pointer position during active interaction.

End current pointer interaction.

Mouse wheel / trackpad zoom or dolly.

Parameters

<i>delta</i>	Positive for zoom in (2D) / dolly in (3D).
<i>cursorX,cursorY</i>	Cursor at event time (pixels).

Keyboard panning in 2D mode (pixels).

Keyboard dolly in 3D mode (world units along view).

< Owned/aliased image.

< Internal cursor anchor (implementation detail).

Definition at line 121 of file [CImageDisplay.h](#).

```
00133                                     {
00134     public:
00135         CImageDisplay();
00136
00137         // ---- Image ownership / allocation (delegates to CSH_Image) ----
00138
00139         void setImage(const csh_img::CSH_Image& img, csh_img::CopyMode mode =
00140             csh_img::CopyMode::Shallow);
00141
00142         void setImageRaw(uint32_t w, uint32_t h,
00143             csh_img::En_ImageFormat fmt,
00144             csh_img::En_ImagePattern pat,
00145             csh_img::En_ImageMemoryAlign align,
00146             const void* pixels, std::size_t bytes,
00147             csh_img::CopyMode mode);
00148
00149         void allocateImageBuffer(std::uint32_t w, std::uint32_t h, csh_img::En_ImageFormat fmt,
00150             std::uint32_t count = 1);
00151
00152         const csh_img::CSH_Image& image() const noexcept;
00153         csh_img::CSH_Image& image() noexcept;
00154
00155         // ---- Viewport & fit ----
00156
00157         void setViewport(int w, int h);
00158         int viewportWidth() const noexcept;
00159         int viewportHeight() const noexcept;
00160         void setFitMode(FitMode m);
00161         FitMode fitMode() const noexcept;
00162
00163         // ---- Dimensionality ----
00164
00165         void setDimensionality(Dimensionality d);
00166         Dimensionality dimensionality() const noexcept;
00167
00168         // ---- 2D transform state ----
00169
00170         void set2DAnchor(float ax, float ay);
00171         void set2DTranslation(float tx, float ty);
00172         void set2DScale(float sx, float sy);
00173         void set2DRotationDeg(float deg);
00174         void reset2D();
00175
00176         // ---- 3D transform state ----
00177
00178         void set3DModelTranslate(const Vec3& t);
00179         void set3DModelScale(const Vec3& s);
00180         void set3DModelRotation(const Quat& q);
00181         void reset3DModel();
00182
00183         void set3DTarget(const Vec3& t);
00184         void set3DEye(const Vec3& e);
00185         void set3DUp(const Vec3& u);
00186         void set3DOrbitStyle(OrbitStyle s);
00187
00188         // ---- projection ----
00189
00190         void setOrtho(float l, float r, float b, float t, float n, float f);
00191         void setPerspective(float fovyDeg, float aspect, float zNear, float zFar);
00192         bool isOrthographic() const noexcept;
00193
00194         // ---- matrices ----
00195
00196         Mat3 modelMatrix2D() const;
00197         Mat4 modelMatrix3D() const;
00198         Mat4 viewMatrix3D() const;
00199         Mat4 projectionMatrix() const;
00200         Mat4 mvp3D() const;
00201
00202         // ---- geometry helpers ----
00203
00204         std::array<std::array<float, 4>, 4> triStrip2D_XYUV() const;
00205
00206         static std::array<std::array<float, 4>, 4> triStrip3D_XYUV_ObjectSpace();
00207
00208         // ---- upload descriptor ----
00209
00210
```

```

00275         UploadDescriptor uploadDesc() const;
00276
00277         // ===== Input hooks (call from UI callbacks) =====
00278
00285         void beginPointer(float x, float y, MouseButton btn, KeyMod mods = KeyMod::None);
00286
00290         void updatePointer(float x, float y);
00291
00293         void endPointer();
00294
00300         void wheelScroll(float delta, float cursorX, float cursorY);
00301
00303         void keyPan2D(float dx, float dy);
00304
00306         void keyDolly3D(float amount);
00307
00308     private:
00309         // ----- state -----
00310         csh_img::CSH_Image image_{};
00311
00312         int viewportW_ = 0, viewportH_ = 0;
00313         FitMode fitMode_ = FitMode::None;
00314         Dimensionality mode_ = Dimensionality::Mode2D;
00315
00316         // 2D
00317         Vec2 anchor2D_{ 0.5f, 0.5f };
00318         Vec2 translate2D_{ 0, 0 };
00319         Vec2 scale2D_{ 1, 1 };
00320         Vec2 currDelta_{ 0, 0 };
00321         Vec2 cumDelta_{ 0, 0 };
00322         float rotation2D_ = 0.f;
00323         mutable Vec3 alpha_{ 0, 0, 1 };
00324         mutable Mat3 modelMatrix2D_ = kMat3Identity;
00325
00326         // 3D model/camera
00327         Vec3 modelT_{ 0, 0, 0 };
00328         Vec3 modelS_{ 1, 1, 1 };
00329         Quat modelR_{ Quat::identity() };
00330         Vec3 target_{ 0, 0, 0 };
00331         Vec3 eye_{ 0, 0, 1000 };
00332         Vec3 up_{ 0, 1, 0 };
00333         OrbitStyle orbitStyle_{ OrbitStyle::Arcball };
00334         Mat4 proj_{ Mat4::identity() };
00335         bool isOrtho_{ true };
00336
00337         // Pointer interaction (cached)
00338         bool pActive_ = false;
00339         MouseButton pBtn_ = MouseButton::None;
00340         KeyMod pMods_ = KeyMod::None;
00341         Vec2 pPrev_{}, pStart_{};
00342
00343         // 3D anchors
00344         Vec3 arcballStart_{};
00345         Vec3 eyeStart_{};
00346         Quat modelRStart_{};
00347         // 2D anchors
00348         Vec2 translate2DStart_{};
00349         Vec2 scale2DStart_{};
00350     };

```

4.6 CImageDisplayer.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 //
00003 // CImageDisplayer.h -- framework-agnostic image "displayer"
00004 // - DOES NOT render; only exposes math and upload descriptors
00005 // - 2D and 3D transforms (model/view/proj)
00006 // - Input hooks for UI callbacks (mouse/keys)
00007 // - Cross-platform export macro (Windows DLL / Linux .so)
00008 //
00009 // C++17
00010 //
00011
00012 #include <array>
00013 #include <cstdint>
00014 #include <cstddef>
00015
00016 #include "CSH_Image.h" // canonical image container (namespace csh_img)
00017
00018 // ===== export macro (Windows/Linux) =====

```

```

00019 #if defined(_WIN32) || defined(_WIN64)
00020 #   if defined(CIMAGE_DISPLAYER_EXPORT)
00021 #       define CIMAGE_API __declspec(dllexport)
00022 #   else
00023 #       define CIMAGE_API __declspec(dllimport)
00024 #   endif
00025 #else
00026 #   define CIMAGE_API __attribute__((visibility("default")))
00027 #endif
00028
00041 namespace cimage {
00042
00043     // ----- Upload / pixel descriptors -----
00044
00048     enum class PixelLayout : uint32_t {
00049         Unknown = 0,
00050         Gray8,
00051         RGB888,
00052         BGR888,
00053         YUV422Packed,
00054         RGB565,
00055         Gray16,
00056         Bayer16
00057     };
00058
00062     enum class Yuv422Pattern : uint32_t { YUYV = 0, UYVY, VYU, VYUY };
00063
00068     struct UploadDescriptor {
00069         const std::uint8_t* data = nullptr;
00070         std::size_t sizeBytes = 0;
00071         int width = 0, height = 0;
00072         int bytesPerPixel = 0;
00073         int strideBytes = 0;
00074         PixelLayout layout = PixelLayout::Unknown;
00075         Yuv422Pattern yuv422Pattern = Yuv422Pattern::YUYV;
00076         bool isPacked = true;
00077         bool isLittleEndian16 = true;
00078     };
00079
00080     // ----- simple math types -----
00081
00083     struct Vec2 { float x = 0.f, y = 0.f; };
00085     struct Vec3 { float x = 0.f, y = 0.f, z = 0.f; };
00087     struct Quat { float w = 1.f, x = 0.f, y = 0.f, z = 0.f; static Quat identity() { return {}; } };
00088
00092     struct Mat4 {
00093         float m[16]{ 1,0,0,0,
00094                     0,1,0,0,
00095                     0,0,1,0,
00096                     0,0,0,1 };
00098         static Mat4 identity() { return {}; }
00099     };
00100
00102     using Mat3 = std::array<float, 9>;
00104     inline constexpr Mat3 kMat3Identity{ 1,0,0, 0,1,0, 0,0,1 };
00105
00106     // ----- modes & input -----
00107
00109     enum class Dimensionality : uint32_t { Mode2D = 0, Mode3D };
00111     enum class FitMode : uint32_t { None = 0, Fit, Fill, Stretch };
00113     enum class OrbitStyle : uint32_t { Arcball = 0, Turntable };
00114
00116     enum class MouseButton : uint32_t { None = 0, Left = 1, Middle = 2, Right = 4 };
00118     enum class KeyMod : uint32_t { None = 0, Shift = 1, Ctrl = 2, Alt = 4 };
00119
00121     inline KeyMod operator|(KeyMod a, KeyMod b) { return KeyMod(uint32_t(a) | uint32_t(b)); }
00122
00123     // ===== CImageDisplay class =====
00124
00133     class CIMAGE_API CImageDisplay {
00134     public:
00136         CImageDisplay();
00137
00138         // ---- Image ownership / allocation (delegates to CSH_Image) ----
00139
00146         void setImage(const csh_img::CSH_Image& img, csh_img::CopyMode mode =
csh_img::CopyMode::Shallow);
00147
00159         void setImageRaw(uint32_t w, uint32_t h,
00160                         csh_img::En_ImageFormat fmt,
00161                         csh_img::En_ImagePattern pat,
00162                         csh_img::En_ImageMemoryAlign align,
00163                         const void* pixels, std::size_t bytes,
00164                         csh_img::CopyMode mode);
00165
00173         void allocateImageBuffer(std::uint32_t w, std::uint32_t h, csh_img::En_ImageFormat fmt,
std::uint32_t count = 1);

```

```

00174
00176     const csh_img::CSH_Image& image() const noexcept;
00178     csh_img::CSH_Image& image()         noexcept;
00179
00180     // ---- Viewport & fit ----
00181
00183     void setViewport(int w, int h);
00185     int  viewportWidth() const noexcept;
00187     int  viewportHeight() const noexcept;
00189     void setFitMode(FitMode m);
00191     FitMode fitMode() const noexcept;
00192
00193     // ---- Dimensionality ----
00194
00196     void setDimensionality(Dimensionality d);
00198     Dimensionality dimensionality() const noexcept;
00199
00200     // ---- 2D transform state ----
00201
00203     void set2DAnchor(float ax, float ay);
00205     void set2DTranslation(float tx, float ty);
00207     void set2DScale(float sx, float sy);
00209     void set2DRotationDeg(float deg);
00211     void reset2D();
00212
00213     // ---- 3D transform state ----
00214
00216     void set3DModelTranslate(const Vec3& t);
00218     void set3DModelScale(const Vec3& s);
00220     void set3DModelRotation(const Quat& q);
00222     void reset3DModel();
00223
00225     void set3DTarget(const Vec3& t);
00227     void set3DEye(const Vec3& e);
00229     void set3DUp(const Vec3& u);
00231     void set3DOrbitStyle(OrbitStyle s);
00232
00233     // ---- projection ----
00234
00236     void setOrtho(float l, float r, float b, float t, float n, float f);
00238     void setPerspective(float fovyDeg, float aspect, float zNear, float zFar);
00240     bool isOrthographic() const noexcept;
00241
00242     // ---- matrices ----
00243
00245     Mat3 modelMatrix2D() const;
00247     Mat4 modelMatrix3D() const;
00249     Mat4 viewMatrix3D() const;
00251     Mat4 projectionMatrix() const;
00253     Mat4 mvp3D() const;
00254
00255     // ---- geometry helpers ----
00256
00261     std::array<std::array<float, 4>, 4> triStrip2D_XYUV() const;
00262
00267     static std::array<std::array<float, 4>, 4> triStrip3D_XYUV_ObjectSpace();
00268
00269     // ---- upload descriptor ----
00270
00275     UploadDescriptor uploadDesc() const;
00276
00277     // ===== Input hooks (call from UI callbacks) =====
00278
00285     void beginPointer(float x, float y, MouseButton btn, KeyMod mods = KeyMod::None);
00286
00290     void updatePointer(float x, float y);
00291
00293     void endPointer();
00294
00300     void wheelScroll(float delta, float cursorX, float cursorY);
00301
00303     void keyPan2D(float dx, float dy);
00304
00306     void keyDolly3D(float amount);
00307
00308 private:
00309     // ----- state -----
00310     csh_img::CSH_Image image_{};
00311
00312     int viewportW_ = 0, viewportH_ = 0;
00313     FitMode fitMode_ = FitMode::None;
00314     Dimensionality mode_ = Dimensionality::Mode2D;
00315
00316     // 2D
00317     Vec2 anchor2D_{ 0.5f, 0.5f };
00318     Vec2 translate2D_{ 0, 0 };
00319     Vec2 scale2D_{ 1, 1 };

```

```

00320     Vec2 currDelta_{ 0,0 };
00321     Vec2 cumDelta_{ 0,0 };
00322     float rotation2D_ = 0.f;
00323     mutable Vec3 alpha_{ 0, 0, 1 };
00324     mutable Mat3 modelMatrix2D_ = kMat3Identity;
00325
00326     // 3D model/camera
00327     Vec3 modelT_{ 0,0,0 };
00328     Vec3 modelS_{ 1,1,1 };
00329     Quat modelR_{ Quat::identity() };
00330     Vec3 target_{ 0,0,0 };
00331     Vec3 eye_{ 0,0,1000 };
00332     Vec3 up_{ 0,1,0 };
00333     OrbitStyle orbitStyle_{ OrbitStyle::Arcball };
00334     Mat4 proj_{ Mat4::identity() };
00335     bool isOrtho_{ true };
00336
00337     // Pointer interaction (cached)
00338     bool pActive_ = false;
00339     MouseButton pBtn_ = MouseButton::None;
00340     KeyMod pMods_ = KeyMod::None;
00341     Vec2 pPrev_{}, pStart_{};
00342
00343     // 3D anchors
00344     Vec3 arcballStart_{};
00345     Vec3 eyeStart_{};
00346     Quat modelRStart_{};
00347     // 2D anchors
00348     Vec2 translate2DStart_{};
00349     Vec2 scale2DStart_{};
00350 };
00351
00352 } // namespace cimage

```

4.7 CImageDisplayer/CImageDisplayerC.h File Reference

C ABI wrapper for `cimage::CImageDisplayer`.

```

#include <stddef.h>
#include <stdint.h>

```

Classes

- struct [CImageUploadDesc](#)
Upload descriptor (C layout) matching the C++ UploadDescriptor.

Typedefs

- typedef struct CImageDisplayerHandle_t * **CImageDisplayerHandle**
Opaque instance handle for the displayer.

Enumerations

- enum [CImgFormat](#)
Image formats (numeric values match [csh_img::En_ImageFormat](#)).
 - enum [CImgPattern](#)
Pixel pattern / channel order. Mirrors [csh_img::En_ImagePattern](#).
 - enum [CImgAlign](#)
Memory alignment / packing. Mirrors [csh_img::En_ImageMemoryAlign](#).
 - enum [CImgCopyMode](#)
Copy semantics for image ingestion.
 - enum [CImgDimensionality](#)
Dimensionality / fit / orbit enums mirror the C++ ones.
-

Functions

- `__attribute__ ((visibility("default"))) CImageDisplayHandle cimgCreate(void)`
Create a new displayer instance.

4.7.1 Detailed Description

C ABI wrapper for `cimage::CImageDisplayer`.

The API exposes an opaque handle and simple functions to configure image data, transforms and to query matrices/geometry/upload descriptors from C.

Definition in file [CImageDisplayerC.h](#).

4.7.2 Function Documentation

4.7.2.1 `__attribute__()`

```
__attribute__ (  
    (visibility("default")) )
```

Create a new displayer instance.

Keyboard dolly in 3D mode (world units along view).

Keyboard panning in 2D mode (pixels).

Mouse wheel / trackpad zoom or dolly.

End current pointer interaction.

Update pointer position during active interaction.

Begin a pointer interaction.

Query the current upload descriptor (pointer/size/stride/layout).

Unit quad in object space for 3D pipelines.

Transformed quad as 2D tri-strip with UVs.

Fetch current MVP = $P \cdot V \cdot M$ (column-major 4×4).

Fetch current projection (column-major 4×4).

Fetch current 3D view (column-major 4×4).

Fetch current 3D model (column-major 4×4).

Fetch current 2D model (row-major 3×3).

Set perspective projection (fovy in degrees).

Set orthographic projection.

Set orbit interaction style.

Set camera up vector.

Set camera eye.

Set look-at target.

Reset model transform to identity.

Set model rotation as quaternion (w,x,y,z).

Set model scale.

Set model translation.

Reset 2D transform to identity.

Set rotation in degrees (CCW).

Set 2D scale factors.

Set 2D translation in viewport pixels.

Set normalized anchor inside image rect $[0..1]^2$.

Switch between 2D and 3D modes.

Set fit strategy (None/Fit/Fill/Stretch).

Set viewport size in pixels (non-negative).

Provide/replace image data from a raw buffer.

Destroy a previously created instance.

Returns

Opaque handle or NULL on allocation failure.

Parameters

<i>h</i>	Instance handle (nullable).
<i>h</i>	Instance handle.
<i>w,hgt</i>	Dimensions in pixels.
<i>fmt</i>	Image format.
<i>pat</i>	Pixel pattern / order.
<i>align</i>	Memory alignment / packing.
<i>pixels</i>	Pointer to buffer (nullable if <code>mode</code> is <code>MetaOnly</code>).
<i>bytes</i>	Size of <code>pixels</code> in bytes.
<i>mode</i>	Copy semantics (<code>MetaOnly</code> / <code>Shallow</code> / <code>Deep</code>).

Note

Shallow does not take ownership; caller must keep buffer alive.

Parameters

out	<i>out4x4</i>	Flattened [4][4] array in row-major {x,y,u,v} order per vertex (TL,TR,BL,BR).
out	<i>out4x4</i>	Flattened [4][4] array in row-major {x,y,u,v}.
	<i>h</i>	Instance handle.
	<i>outDesc</i>	Output struct (must not be NULL).
	<i>h</i>	Instance handle.
	<i>x,y</i>	Cursor in viewport pixels.
	<i>btn</i>	Mouse button bitmask.
	<i>keyMods</i>	Keyboard modifier bitmask.
	<i>delta</i>	Positive for zoom in (2D) / dolly in (3D).
	<i>cursorX,cursorY</i>	Cursor at event time (pixels).

4.8 CImageDisplayerC.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 //
00003 // cimage_displayer_c.h -- Pure C ABI for CImageDisplayer
00004 // Build this into your shared library alongside CImageDisplayer.cpp.
00005 // Opaque handle + procedural functions. No C++ types leak.
00006 //
00007 // Note: enum values mirror csh_img enums numerically for ease of mapping.
00008 //
00009
00010 #include <stddef.h>
00011 #include <stdint.h>
00012
00020
00021 #if defined(_WIN32) || defined(_WIN64)
00022 #   if defined(CIMAGE_DISPLAYER_EXPORT)
00023 #       define CIMAGE_C_API __declspec(dllexport)
00024 #   else
00025 #       define CIMAGE_C_API __declspec(dllimport)
00026 #   endif
00027 #else
00028 #   define CIMAGE_C_API __attribute__((visibility("default")))
00029 #endif
00030
00031 #ifdef __cplusplus
00032 extern "C" {
00033 #endif
00034
00036     typedef enum {
00037         CIMG_FMT_Bayer8 = 100,
00038         CIMG_FMT_Gray8,
00039         CIMG_FMT_Bayer10 = 200, CIMG_FMT_Bayer12, CIMG_FMT_Bayer14, CIMG_FMT_Bayer16,
00040         CIMG_FMT_Gray10, CIMG_FMT_Gray12, CIMG_FMT_Gray14, CIMG_FMT_Gray16,
00041         CIMG_FMT_YUV422, CIMG_FMT_RGB565,
00042         CIMG_FMT_YUV444 = 300, CIMG_FMT_RGB888, CIMG_FMT_BGR888
00043     } CImgFormat;
00044
00046     typedef enum {
00047         CIMG_PAT_RGGB = 0, CIMG_PAT_GRGB, CIMG_PAT_BGGR, CIMG_PAT_GBRG,
00048         CIMG_PAT_YUYV = 10, CIMG_PAT_UYVY, CIMG_PAT_YVYU, CIMG_PAT_VYUY,
00049         CIMG_PAT_RGB = 20, CIMG_PAT_BGR
00050     } CImgPattern;
00051
00053     typedef enum {
00054         CIMG_ALIGN_Packed = 0, CIMG_ALIGN_YYYYUUUVVVV = 10, CIMG_ALIGN_YYYYVVVVUUUU,
00055         CIMG_ALIGN_UUUUVVVVYYYY, CIMG_ALIGN_VVVUUUUYYYYY, CIMG_ALIGN_RRRRGGGGBBBB = 20,
00056         CIMG_ALIGN_BBBBGGGRRRRR, CIMG_ALIGN_YYYYUVUV = 30, CIMG_ALIGN_YYYYVUVU
00057     } CImgAlign;
00058
00060     typedef enum { CIMG_COPY_MetaOnly = 0, CIMG_COPY_Shallow, CIMG_COPY_Deep } CImgCopyMode;
00061
00063     typedef enum { CIMG_DIM_2D = 0, CIMG_DIM_3D } CImgDimensionality;
00064     typedef enum { CIMG_FIT_None = 0, CIMG_FIT_Fit, CIMG_FIT_Fill, CIMG_FIT_Stretch } CImgFitMode;
00065     typedef enum { CIMG_ORBIT_Arcball = 0, CIMG_ORBIT_Turntable } CImgOrbitStyle;
00066
00069     typedef uint32_t CImgMouseButton;
00070 #define CIMG_BTN_None    0u
00071 #define CIMG_BTN_Left    1u

```

```

00072 #define CIMG_BTN_Middle 2u
00073 #define CIMG_BTN_Right 4u
00075
00078 typedef uint32_t CImgKeyMod;
00079 #define CIMG_KMOD_NONE 0u
00080 #define CIMG_KMOD_SHIFT 1u
00081 #define CIMG_KMOD_CTRL 2u
00082 #define CIMG_KMOD_ALT 4u
00084
00086 typedef struct CImageDisplayHandle_t* CImageDisplayHandle;
00087
00092 typedef struct {
00093     const uint8_t* data;
00094     size_t sizeBytes;
00095     int32_t width, height;
00096     int32_t bytesPerPixel;
00097     int32_t strideBytes;
00098     int32_t layout;
00099     int32_t yuv422Pattern;
00100     int32_t isPacked;
00101     int32_t isLittleEndian16;
00102 } CImageUploadDesc;
00103
00104 // ----- lifecycle -----
00105
00110 CIMAGE_C_API CImageDisplayHandle cimgCreate(void);
00111
00116 CIMAGE_C_API void cimgDestroy(CImageDisplayHandle h);
00117
00118 // ----- image set (raw) -----
00119
00132 CIMAGE_C_API void cimgSetImageRaw(CImageDisplayHandle h,
00133     uint32_t w, uint32_t hgt, CImgFormat fmt, CImgPattern pat, CImgAlign align,
00134     const void* pixels /*nullable*/, size_t bytes, CImgCopyMode mode);
00135
00136 // ----- viewport / mode / fit -----
00137
00139 CIMAGE_C_API void cimgSetViewport(CImageDisplayHandle h, int32_t w, int32_t hgt);
00141 CIMAGE_C_API void cimgSetFitMode(CImageDisplayHandle h, CImgFitMode m);
00143 CIMAGE_C_API void cimgSetDimensionality(CImageDisplayHandle h, CImgDimensionality d);
00144
00145 // ----- 2D transform -----
00146
00148 CIMAGE_C_API void cimg2D_SetAnchor(CImageDisplayHandle h, float ax, float ay);
00150 CIMAGE_C_API void cimg2D_SetTranslation(CImageDisplayHandle h, float tx, float ty);
00152 CIMAGE_C_API void cimg2D_SetScale(CImageDisplayHandle h, float sx, float sy);
00154 CIMAGE_C_API void cimg2D_SetRotationDeg(CImageDisplayHandle h, float deg);
00156 CIMAGE_C_API void cimg2D_Reset(CImageDisplayHandle h);
00157
00158 // ----- 3D transform / camera / projection -----
00159
00161 CIMAGE_C_API void cimg3D_SetModelTranslate(CImageDisplayHandle h, float x, float y, float z);
00163 CIMAGE_C_API void cimg3D_SetModelScale(CImageDisplayHandle h, float x, float y, float z);
00165 CIMAGE_C_API void cimg3D_SetModelRotationQuat(CImageDisplayHandle h, float w, float x, float y,
float z);
00167 CIMAGE_C_API void cimg3D_ResetModel(CImageDisplayHandle h);
00168
00170 CIMAGE_C_API void cimg3D_SetTarget(CImageDisplayHandle h, float x, float y, float z);
00172 CIMAGE_C_API void cimg3D_SetEye(CImageDisplayHandle h, float x, float y, float z);
00174 CIMAGE_C_API void cimg3D_SetUp(CImageDisplayHandle h, float x, float y, float z);
00176 CIMAGE_C_API void cimg3D_SetOrbitStyle(CImageDisplayHandle h, CImgOrbitStyle s);
00177
00179 CIMAGE_C_API void cimgProj_SetOrtho(CImageDisplayHandle h, float l, float r, float b, float t,
float n, float f);
00181 CIMAGE_C_API void cimgProj_SetPerspective(CImageDisplayHandle h, float fovyDeg, float aspect,
float zNear, float zFar);
00182
00183 // ----- matrices (outputs) -----
00184
00186 CIMAGE_C_API void cimgGetModel2D_3x3(CImageDisplayHandle h, float* outRowMajor3x3);
00188 CIMAGE_C_API void cimgGetModel3D_4x4(CImageDisplayHandle h, float* outColMajor4x4);
00190 CIMAGE_C_API void cimgGetView3D_4x4(CImageDisplayHandle h, float* outColMajor4x4);
00192 CIMAGE_C_API void cimgGetProj_4x4(CImageDisplayHandle h, float* outColMajor4x4);
00194 CIMAGE_C_API void cimgGetMVP3D_4x4(CImageDisplayHandle h, float* outColMajor4x4);
00195
00196 // ----- geometry helpers -----
00197
00202 CIMAGE_C_API void cimgTriStrip2D_XYUV(CImageDisplayHandle h, float* out4x4 /*[4][4]*/);
00203
00208 CIMAGE_C_API void cimgTriStrip3D_XYUV_ObjectSpace(float* out4x4 /*[4][4]*/);
00209
00210 // ----- upload descriptor -----
00211
00217 CIMAGE_C_API void cimgGetUploadDesc(CImageDisplayHandle h, CImageUploadDesc* outDesc);
00218
00219 // ----- input hooks (for UI callbacks) -----
00220

```

```

00228     CIMAGE_C_API void cimgBeginPointer(CImageDisplayerHandle h, float x, float y, uint32_t btn,
uint32_t keyMods /**bitmask*/);
00229
00231     CIMAGE_C_API void cimgUpdatePointer(CImageDisplayerHandle h, float x, float y);
00233     CIMAGE_C_API void cimgEndPointer(CImageDisplayerHandle h);
00234
00240     CIMAGE_C_API void cimgWheelScroll(CImageDisplayerHandle h, float delta, float cursorX, float
cursorY);
00241
00242     // ----- convenience keyboard -----
00243
00245     CIMAGE_C_API void cimgKeyPan2D(CImageDisplayerHandle h, float dx, float dy);
00247     CIMAGE_C_API void cimgKeyDolly3D(CImageDisplayerHandle h, float amount);
00248
00249 #ifdef __cplusplus
00250 } // extern "C"
00251 #endif

```

4.9 CImageDisplayer/CImageDisplayerCPP.h File Reference

Thin C++ wrapper around the pure-C [CImageDisplayer](#) ABI.

```

#include "CImageDisplayerC.h"
#include "CSH_Image.h"
#include <stdexcept>
#include <cstring>

```

Classes

- class [cimage::CImageDisplayerCPP](#)
RAII C++ wrapper around the C ABI. No rendering; forwards to C API.

Enumerations

- enum class [cimage::Dimensionality](#) : int
2D/3D mode (scoped mirror of C enums).
- enum class [cimage::FitMode](#) : int
Fit/Fill strategy (scoped mirror).
- enum class [cimage::OrbitStyle](#) : int
Orbit interaction style (scoped mirror).
- enum class [cimage::MouseButton](#) : int
Mouse button bitmask (scoped mirror). Combinable via bitwise ops.
- enum class [cimage::KeyMod](#) : uint32_t
Keyboard modifier bitmask (scoped mirror). Combinable via bitwise ops.

Functions

- [MouseButton](#) [cimage::operator|](#) ([MouseButton](#) a, [MouseButton](#) b)
Bitwise OR for mouse buttons.
- [MouseButton](#) [cimage::operator&](#) ([MouseButton](#) a, [MouseButton](#) b)
Bitwise AND for mouse buttons.
- [MouseButton](#) & [cimage::operator|=](#) ([MouseButton](#) &a, [MouseButton](#) b)
In-place OR for mouse buttons.
- bool [cimage::any](#) ([MouseButton](#) m)

-
- `KeyMod cimage::operator| (KeyMod a, KeyMod b)`
Bitwise OR for key modifiers.
 - `KeyMod cimage::operator& (KeyMod a, KeyMod b)`
Bitwise AND for key modifiers.
 - `KeyMod & cimage::operator|= (KeyMod &a, KeyMod b)`
In-place OR for key modifiers.
 - `bool cimage::any (KeyMod m)`
True if any modifier bit is set.

Enum conversions to C ABI

Enum conversions from C ABI

4.9.1 Detailed Description

Thin C++ wrapper around the pure-C `CImageDisplayer` ABI.

Provides RAII lifetime management and strongly-typed enums while keeping ABI compatibility with the C layer. Zero rendering is performed here.

Definition in file `CImageDisplayerCPP.h`.

4.10 CImageDisplayerCPP.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 //
00003 // CImageDisplayerCPP.h -- C++ convenience wrapper over the pure C ABI.
00004 // Depends on: cimage_displayer_c.h and CSH_Image.h
00005 //
00006
00007 #include "CImageDisplayerC.h"
00008 #include "CSH_Image.h"
00009 #include <stdexcept>
00010 #include <cstring>
00011
00019
00020 namespace cimage {
00021
00022     // 1) Strongly-typed mirrors (scoped; safe)
00023
00025     enum class Dimensionality : int { _2D = CIMG_DIM_2D, _3D = CIMG_DIM_3D };
00026
00028     enum class FitMode : int {
00029         None = CIMG_FIT_None, Fit = CIMG_FIT_Fit,
00030         Fill = CIMG_FIT_Fill, Stretch = CIMG_FIT_Stretch
00031     };
00032
00034     enum class OrbitStyle : int { Arcball = CIMG_ORBIT_Arcball, Turntable = CIMG_ORBIT_Turntable };
00035
00039     enum class MouseButton : int {
00040         None = CIMG_BTN_None,
00041         Left = CIMG_BTN_Left,
00042         Middle = CIMG_BTN_Middle,
00043         Right = CIMG_BTN_Right
00044     };
00046     inline MouseButton operator|(MouseButton a, MouseButton b) { return
static_cast<MouseButton>(static_cast<int>(a) | static_cast<int>(b)); }
00048     inline MouseButton operator&(MouseButton a, MouseButton b) { return
static_cast<MouseButton>(static_cast<int>(a) & static_cast<int>(b)); }
00050     inline MouseButton& operator|=(MouseButton& a, MouseButton b) { a = a | b; return a; }
00052     inline bool any(MouseButton m) { return static_cast<int>(m) != 0; }
```

```

00053
00057     enum class KeyMod : uint32_t {
00058         None = CIMG_KMOD_NONE,
00059         Shift = CIMG_KMOD_SHIFT,
00060         Ctrl = CIMG_KMOD_CTRL,
00061         Alt = CIMG_KMOD_ALT
00062     };
00064     inline KeyMod operator|(KeyMod a, KeyMod b) { return static_cast<KeyMod>(static_cast<uint32_t>(a)
| static_cast<uint32_t>(b)); }
00066     inline KeyMod operator&(KeyMod a, KeyMod b) { return static_cast<KeyMod>(static_cast<uint32_t>(a)
& static_cast<uint32_t>(b)); }
00068     inline KeyMod& operator|=(KeyMod& a, KeyMod b) { a = a | b; return a; }
00070     inline bool any(KeyMod m) { return static_cast<uint32_t>(m) != 0; }
00071
00072     // Conversion helpers (constexpr, inline)
00075     constexpr uint32_t to_c(KeyMod m) { return static_cast<uint32_t>(m); }
00076     constexpr CImgDimensionality to_c(Dimensionality d) { return
static_cast<CImgDimensionality>(static_cast<int>(d)); }
00077     constexpr CImgFitMode to_c(FitMode f) { return
static_cast<CImgFitMode>(static_cast<int>(f)); }
00078     constexpr CImgOrbitStyle to_c(OrbitStyle o) { return
static_cast<CImgOrbitStyle>(static_cast<int>(o)); }
00079     constexpr uint32_t to_c(MouseButton m) { return static_cast<uint32_t>(m); }
00081
00084     constexpr KeyMod from_c(uint32_t m) { return static_cast<KeyMod>(m); }
00085     constexpr Dimensionality from_c(CImgDimensionality d) { return
static_cast<Dimensionality>(static_cast<int>(d)); }
00086     constexpr FitMode from_c(CImgFitMode f) { return static_cast<FitMode>(static_cast<int>(f)); }
00087 }
00087     constexpr OrbitStyle from_c(CImgOrbitStyle o) { return
static_cast<OrbitStyle>(static_cast<int>(o)); }
00088     constexpr MouseButton from_c_btns(int bits) { return static_cast<MouseButton>(bits); }
00090
00098     class CImageDisplayerCPP {
00099     public:
00101         CImageDisplayerCPP() {
00102             h_ = cimgCreate();
00103             if (!h_) throw std::bad_alloc();
00104         }
00105
00107         ~CImageDisplayerCPP() { cimgDestroy(h_); }
00108
00109         CImageDisplayerCPP(const CImageDisplayerCPP&) = delete;
00110         CImageDisplayerCPP& operator=(const CImageDisplayerCPP&) = delete;
00111
00113         CImageDisplayerCPP(CImageDisplayerCPP&& o) noexcept : h_(o.h_) { o.h_ = nullptr; }
00115         CImageDisplayerCPP& operator=(CImageDisplayerCPP&& o) noexcept {
00116             if (this != &o) { cimgDestroy(h_); h_ = o.h_; o.h_ = nullptr; }
00117             return *this;
00118         }
00119
00125         void setImage(const csh_img::CSH_Image& img, csh_img::CopyMode mode =
csh_img::CopyMode::Shallow) {
00126             const auto fmt = static_cast<CImgFormat>(static_cast<uint32_t>(img.getFormat()));
00127             const auto pat = static_cast<CImgPattern>(static_cast<uint32_t>(img.getPattern()));
00128             const auto alg = static_cast<CImgAlign>(static_cast<uint32_t>(img.getMemoryAlign()));
00129
00130             const void* data = nullptr;
00131             size_t bytes = 0;
00132             if (mode != csh_img::CopyMode::MetaOnly) {
00133                 data = img.data();
00134                 bytes = img.getBufferSize();
00135             }
00136             cimgSetImageRaw(h_, img.getWidth(), img.getHeight(), fmt, pat, alg, data, bytes,
static_cast<CImgCopyMode>(static_cast<uint32_t>(mode)));
00137         }
00138
00139         void setImageRaw(uint32_t w, uint32_t h,
00151             CImgFormat fmt, CImgPattern pat, CImgAlign align,
00152             const void* pixels, size_t bytes, CImgCopyMode mode) {
00153             cimgSetImageRaw(h_, w, h, fmt, pat, align, pixels, bytes, mode);
00154         }
00155
00156         // Viewport / fit / mode
00157
00159         void setViewport(int w, int h) { cimgSetViewport(h_, w, h); }
00161         void setFitMode(FitMode m) { cimgSetFitMode(h_, to_c(m)); }
00163         void setDimensionality(Dimensionality d) { cimgSetDimensionality(h_, to_c(d)); }
00164
00165         // 2D
00166
00168         void set2DAnchor(float ax, float ay) { cimg2D_SetAnchor(h_, ax, ay); }
00170         void set2DTranslation(float tx, float ty) { cimg2D_SetTranslation(h_, tx, ty); }
00172         void set2DScale(float sx, float sy) { cimg2D_SetScale(h_, sx, sy); }
00174         void set2DRotationDeg(float deg) { cimg2D_SetRotationDeg(h_, deg); }
00176         void reset2D() { cimg2D_Reset(h_); }
00177

```



```

00178         // 3D
00179
00181     void set3DModelTranslate(float x, float y, float z) { cimg3D_SetModelTranslate(h_, x, y, z); }
00183     void set3DModelScale(float x, float y, float z) { cimg3D_SetModelScale(h_, x, y, z); }
00185     void set3DModelRotationQuat(float w, float x, float y, float z) {
cimg3D_SetModelRotationQuat(h_, w, x, y, z); }
00187     void reset3DModel() { cimg3D_ResetModel(h_); }
00189     void set3DTarget(float x, float y, float z) { cimg3D_SetTarget(h_, x, y, z); }
00191     void set3DEye(float x, float y, float z) { cimg3D_SetEye(h_, x, y, z); }
00193     void set3DUp(float x, float y, float z) { cimg3D_SetUp(h_, x, y, z); }
00195     void set3DOrbitStyle(CImgOrbitStyle s) { cimg3D_SetOrbitStyle(h_, s); }
00196
00198     void setOrtho(float l, float r, float b, float t, float n, float f) { cimgProj_SetOrtho(h_, l,
r, b, t, n, f); }
00200     void setPerspective(float fovyDeg, float aspect, float zNear, float zFar) {
cimgProj_SetPerspective(h_, fovyDeg, aspect, zNear, zFar); }
00201
00202         // Matrices
00203
00205     void model2D_3x3(float outRowMajor3x3[9]) const { cimgGetModel2D_3x3(h_, outRowMajor3x3); }
00207     void model3D_4x4(float outColMajor4x4[16]) const { cimgGetModel3D_4x4(h_, outColMajor4x4); }
00209     void view3D_4x4(float outColMajor4x4[16]) const { cimgGetView3D_4x4(h_, outColMajor4x4); }
00211     void proj_4x4(float outColMajor4x4[16]) const { cimgGetProj_4x4(h_, outColMajor4x4); }
00213     void mvp3D_4x4(float outColMajor4x4[16]) const { cimgGetMVP3D_4x4(h_, outColMajor4x4); }
00214
00215         // Geometry
00216
00218     void triStrip2D_XYUV(float out4x4[16]) const { cimgTriStrip2D_XYUV(h_, out4x4); }
00220     static void triStrip3D_XYUV_ObjectSpace(float out4x4[16]) {
cimgTriStrip3D_XYUV_ObjectSpace(out4x4); }
00221
00222         // Upload
00223
00225     CImageUploadDesc uploadDesc() const { CImageUploadDesc d{}; cimgGetUploadDesc(h_, &d); return
d; }
00226
00227         // Input hooks
00228
00230     void beginPointer(float x, float y, MouseButton btn, KeyMod keyMods) { cimgBeginPointer(h_, x,
y, to_c(btn), to_c(keyMods)); }
00232     void updatePointer(float x, float y) { cimgUpdatePointer(h_, x, y); }
00234     void endPointer() { cimgEndPointer(h_); }
00240     void wheelScroll(float delta, float cx, float cy) { cimgWheelScroll(h_, delta, cx, cy); }
00241
00243     void keyPan2D(float dx, float dy) { cimgKeyPan2D(h_, dx, dy); }
00245     void keyDolly3D(float amount) { cimgKeyDolly3D(h_, amount); }
00246
00248     CImageDisplayHandle raw() const noexcept { return h_; }
00249
00250     private:
00251         CImageDisplayHandle h_{ nullptr };
00252     };
00253
00254 } // namespace cimage
00255 #pragma once

```

4.11 ImageProcessorManager/CImageProcessMng.h File Reference

High-level image processing pipeline manager (frame handoff + staged processing + display callback).

```

#include <vector>
#include <thread>
#include <atomic>
#include <mutex>
#include <condition_variable>
#include <functional>
#include "IpmTypes.h"
#include "CIpmFuncTable.h"
#include "CSH_Image.h"
#include "Converter/CConverter.h"

```

Typedefs

- using [DisplayCallback](#) = std::function<void(int, int, const [csh_img::CSH_Image&](#))>
UI/display callback signature.

Functions

- class [__attribute__](#) ((visibility("default"))) CImageProcessMng
Image processing manager: staging, chaining, and dispatch to UI.

4.11.1 Detailed Description

High-level image processing pipeline manager (frame handoff + staged processing + display callback).

Responsibilities:

- Accept frames from a grabber (producer) via onNewFrame (double-buffered, lock-light).
- Chain a list of processing stages (module/backend/algIndex) using ClpmFuncTable dispatch.
- Emit processed frames to a UI/display layer via a user-provided callback.

Concurrency Model:

- One internal worker thread consumes frames (created by run and joined in stop).
- Frame ingress uses a double buffer (#DoubleBuffer) with acquire/release memory ordering: producer writes inactive slot -> `active` index store (release) -> consumer reads (acquire).

Ownership:

- `addProcList()` stores raw pointers to input/output images supplied by the caller; it does not own them.
- The first stage's `in` is automatically anchored to the latest source frame (shallow copy).

See also

[ClpmFuncTable.h](#) Algorithm registry and dispatcher.

[lpmTypes.h](#) Types for modules/backends and status codes.

Definition in file [CImageProcessMng.h](#).

4.11.2 Typedef Documentation

4.11.2.1 DisplayCallback

using [DisplayCallback](#) = std::function<void(int, int, const [csh_img::CSH_Image&](#))>

UI/display callback signature.

Parameters

<i>cameraId</i>	The source camera identifier (propagated from the input frame).
<i>stageIndex</i>	Zero-based index of the processed stage in the proc list.
<i>image</i>	The processed image produced by that stage.

Note

The callback is invoked from the pipeline worker thread context.

Definition at line 44 of file [CImageProcessMng.h](#).

4.11.3 Function Documentation

4.11.3.1 `__attribute__()`

```
class __attribute__ (
    (visibility("default")) )
```

Image processing manager: staging, chaining, and dispatch to UI.

Typical flow:

```
CImageProcessMng ipm;
ipm.initialize();
ipm.addProcList(EnProcessBackend::CPU_Serial, EnIpModule::Converter, kYUYV_to_BGR,
               nullptr, & stage0Out, nullptr, nullptr);
*ipm.registerDisplayCallback([](int cam, int idx, const CSH_Image& img) { render });
// ... when a new camera frame arrives:
*ipm.onNewFrame(cameraFrame);
// ... shutdown:
*ipm.deinitialize();
```

```
@thread_safety
-Public methods are safe to call from appropriate contexts :
```

-onNewFrame may be called by the grabber thread at any time. -run / stop control the worker thread; avoid calling from the callback. Construct a manager (no implicit thread start).

Stop worker and release resources.

Initialize core components and start worker thread. Internally touches ClpmEnv/ClpmFuncTable/CConverter single-tons.

Returns

true on success (thread created or already running).

Stop the worker thread and clear pipeline stages.

Ingress point for a new camera frame from the grabber.

Copies the frame into an inactive double-buffer slot (deep copy), then swaps *active* with release semantics and signals the worker via *condition_variable*.

Parameters

<i>frame</i>	Incoming source frame.
--------------	------------------------

Append a processing stage to the pipeline (in order).

If *in* is nullptr for non-first stages, it will automatically chain to the previous stage's *out*. The first stage's *in* is anchored to the most recent source frame (shallow copy).

Parameters

<i>backend</i>	Execution backend.
<i>ipmModule</i>	Module (Converter/Scaler/...).
<i>algIndex</i>	Algorithm index within the (backend,module) catalog.
<i>in</i>	Optional input image for this stage (may be null; will be chained).
<i>out</i>	Output image for this stage (must be a valid pointer; caller-owned).
<i>p1</i>	Opaque parameter 1 (algorithm-specific).
<i>p2</i>	Opaque parameter 2 (algorithm-specific).

Returns

int *lpmStatus* cast to int.

Remove all processing stages.

Register a display callback to receive stage outputs.

Start the worker thread (no-op if already running).

Request worker stop and join the thread.

Temporary access to the converter facade.

Returns

The global converter instance pointer (non-owning).

Internal worker loop body (waits for new frames, processes pipeline).

Process a single available frame across the current stage list.

One pipeline stage description (module/backend + IO + opaque params).

< Input image (may be re-linked for chaining).

< Output image (must be valid).

< Opaque parameter 1.

< Opaque parameter 2.

Double buffer for source frames (producer: grabber / consumer: worker).

Memory ordering:

- Producer stores active=back with memory_order_release after writing slot [back].
- Consumer loads active with memory_order_acquire to see the fully published frame.

< Two deep-owning slots.

< Index of the readable slot.

< System CPU/GPU environment (singleton).

< Converter façade (singleton).

< Function table (singleton).

Definition at line 1 of file CImageProcessMng.h.

```

00067     {
00068     public:
00070         CImageProcessMng();
00072         ~CImageProcessMng();
00073
00079         bool initialize();
00080
00082         void deinitialize();
00083
00092         void onNewFrame(const csh_img::CSH_Image& frame);
00093
00109         int addProcList(ipmcommon::EnProcessBackend backend, ipmcommon::EnIpmModule ipmModule,
00110             int algIndex,
00111             csh_img::CSH_Image* in,
00112             csh_img::CSH_Image* out,
00113             void* p1,
00114             void* p2);
00115
00117         void clearProcList();
00118
00120         void registerDisplayerCallback(DisplayCallback cb);
00121
00123         bool run();
00124
00126         void stop();
00127
00132         CConverter* getConverter() const { return pConvert_; }
00133
00134     private:
00136         void threadEntry_();
00137
00139         void processOneFrame_();
00140
00141     private:
00143         struct ProcItem {
00144             ipmcommon::EnIpmModule    ipmModule;
00145             int                        algIndex;
00146             ipmcommon::EnProcessBackend backend;
00147             csh_img::CSH_Image* in;
00148             csh_img::CSH_Image* out;
00149             void* p1;
00150             void* p2;
00151         };
00152
00160         struct DoubleBuffer {
00161             csh_img::CSH_Image slot[2];
00162             std::atomic<int> active{ 0 };
00163             std::atomic<bool> ready{ false };
00164         };
00165
00166     private:
00167         // Core / Interfaces
00168         ipm::CIpmEnv* pImpEnv_{ nullptr };
00169         CConverter* pConvert_{ nullptr };
00170         ipmcommon::CIpmFuncTable* pFuncTable_{ nullptr };
00171
00172         // Processing / Synchronization
00173         std::vector<ProcItem> vecProcList_;
00174         std::thread thProc_;
00175         std::atomic<bool> bStop_{ false };
00176         std::mutex mtx_;
00177         std::condition_variable cv_;
00178         std::atomic<bool> bNewFrame_{ false };
00179
00180         // Frame ingress double-buffer
00181         DoubleBuffer dbuf_;
00182
00183         // Display callback
00184         DisplayCallback cbDisplay_;
00185     };

```

4.12 CImageProcessMng.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00023
00024 #include <vector>
00025 #include <thread>
00026 #include <atomic>
00027 #include <mutex>
00028 #include <condition_variable>
00029 #include <functional>
00030
00031 #include "IpmTypes.h"
00032 #include "CIpmFuncTable.h"
00033 #include "CSH_Image.h"
00034 #include "Converter/CConverter.h"
00035
00044 using DisplayCallback = std::function<void(int, int, const csh_img::CSH_Image&);>;
00045
00067 class IPM_API CImageProcessMng {
00068 public:
00070     CImageProcessMng();
00072     ~CImageProcessMng();
00073
00079     bool initialize();
00080
00082     void deinitialize();
00083
00092     void onNewFrame(const csh_img::CSH_Image& frame);
00093
00109     int addProcList(ipmcommon::EnProcessBackend backend, ipmcommon::EnIpmModule ipmModule,
00110                    int algIndex,
00111                    csh_img::CSH_Image* in,
00112                    csh_img::CSH_Image* out,
00113                    void* p1,
00114                    void* p2);
00115
00117     void clearProcList();
00118
00120     void registerDisplayerCallback(DisplayCallback cb);
00121
00123     bool run();
00124
00126     void stop();
00127
00132     CConverter* getConverter() const { return pConvert_; }
00133
00134 private:
00136     void threadEntry_();
00137
00139     void processOneFrame_();
00140
00141 private:
00143     struct ProcItem {
00144         ipmcommon::EnIpmModule      ipmModule;
00145         int                          algIndex;
00146         ipmcommon::EnProcessBackend backend;
00147         csh_img::CSH_Image* in;
00148         csh_img::CSH_Image* out;
00149         void* p1;
00150         void* p2;
00151     };
00152
00160     struct DoubleBuffer {
00161         csh_img::CSH_Image slot[2];
00162         std::atomic<int>   active{ 0 };
00163         std::atomic<bool>  ready{ false };
00164     };
00165
00166 private:
00167     // Core / Interfaces
00168     ipm::CIpmEnv* pImpEnv_{ nullptr };
00169     CConverter* pConvert_{ nullptr };
00170     ipmcommon::CIpmFuncTable* pFuncTable_{ nullptr };
00171
00172     // Processing / Synchronization
00173     std::vector<ProcItem> vecProcList_;
00174     std::thread           thProc_;
00175     std::atomic<bool>     bStop_{ false };
00176     std::mutex            mtx_;
00177     std::condition_variable cv_;
00178     std::atomic<bool>     bNewFrame_{ false };
00179
00180     // Frame ingress double-buffer
00181     DoubleBuffer dbuf_;
```

```

00182
00183         // Display callback
00184         DisplayCallback      cbDisplay_;
00185     };

```

4.13 ImageProcessorManager/ClpmCpuEnv.h File Reference

CPU feature probing (x86/x86_64 AVX2/AVX-512/AMX, ARM NEON/SVE/SVE2) and best-SIMD selection.

```
#include <cstdint>
```

Classes

- class [ipm::ClpmCpuEnv](#)
CPU capability probe (non-owning, POD-like; call [Detect](#) once).

Enumerations

- enum class [ipm::En_CpuType](#) : int
Coarse CPU family.
- enum class [ipm::En_SimdKind](#) : int { }
SIMD kinds recognized by the library (generic categorization).
- enum class [ipm::En_OpProfile](#) : int { [Integer8_16](#) , [Float32_64](#) , [Matrix2D](#) }
Coarse operation profiles used to pick a "best" SIMD.

4.13.1 Detailed Description

CPU feature probing (x86/x86_64 AVX2/AVX-512/AMX, ARM NEON/SVE/SVE2) and best-SIMD selection.

Implementation notes (see .cpp):

- x86_64 uses CUID and XGETBV to ensure OS state enables YMM/ZMM before reporting AVX/AVX-512.
- ARM64 on Linux reads HWCAP/HWCAP2 and PR_SVE_GET_VL to discover SVE vector length (bits).
- [Detect\(\)](#) is cheap and intended to run once; call via [ipm::ClpmEnv::Initialize](#).

Definition in file [ClpmCpuEnv.h](#).

4.13.2 Enumeration Type Documentation

4.13.2.1 En_OpProfile

```
enum class ipm::En_OpProfile : int [strong]
```

Coarse operation profiles used to pick a "best" SIMD.

Enumerator

Integer8_16	Pixel processing heavy in 8/16-bit integer math.
Float32_64	Floating-point oriented workloads.
Matrix2D	Convolution/correlation/GEMM; AMX if available.

Definition at line 44 of file [ClpmCpuEnv.h](#).

```
00044                                     : int {
00045     Integer8_16,
00046     Float32_64,
00047     Matrix2D
00048 };
```

4.13.2.2 En_SimdKind

```
enum class ipm::En_SimdKind : int [strong]
```

SIMD kinds recognized by the library (generic categorization).

Enumerator

AVX2	256-bit vectors
AVX512F	512-bit (foundation)
AVX512BW	512-bit byte/word extensions
AMX_Tile	Matrix tile ISA (not a generic vector ISA).
NEON	128-bit SIMD
SVE	Scalable Vector Extension (128–2048 bits).
SVE2	SVE2 (integer/bit ops enhanced).

Definition at line 30 of file [ClpmCpuEnv.h](#).

```
00030                                     : int {
00031     None = 0,
00032     // x86
00033     AVX2,
00034     AVX512F,
00035     AVX512BW,
00036     AMX_Tile,
00037     // ARM
00038     NEON,
00039     SVE,
00040     SVE2
00041 };
```

4.14 ClpmCpuEnv.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00011
00012 #include <stdint>
00013
00014 #if defined(_WIN32) || defined(_WIN64)
00015 #ifdef CSH_IPM_EXPORT
00016 #define IPM_API __declspec(dllexport)
00017 #else
00018 #define IPM_API __declspec(dllimport)
00019 #endif
00020 #else
00021 #define IPM_API __attribute__((visibility("default")))
00022 #endif
```



```

00023
00024 namespace ipm {
00025
00027     enum class En_CpuType : int { x86 = 0, x86_64, ARM8, ARM9, Count };
00028
00030     enum class En_SimdKind : int {
00031         None = 0,
00032         // x86
00033         AVX2,
00034         AVX512F,
00035         AVX512BW,
00036         AMX_Tile,
00037         // ARM
00038         NEON,
00039         SVE,
00040         SVE2
00041     };
00042
00044     enum class En_OpProfile : int {
00045         Integer8_16,
00046         Float32_64,
00047         Matrix2D
00048     };
00049
00056     class IPM_API CIpmCpuEnv {
00057     public:
00058         CIpmCpuEnv() = default;
00059
00061         void Detect();
00062
00063         // --- Basic identity/state ---
00064         En_CpuType cpu() const { return cpu_; }
00065
00066         // --- x86 feature flags ---
00067         bool hasAVX2() const { return has_avx2_; }
00068         bool hasAVX512F() const { return has_avx512f_; }
00069         bool hasAVX512BW() const { return has_avx512bw_; }
00070         bool hasAMX() const { return has_amx_tile_; }
00071
00072         // --- ARM feature flags ---
00073         bool hasNEON() const { return has_neon_; }
00074         bool hasSVE() const { return has_sve_; }
00075         bool hasSVE2() const { return has_sve2_; }
00076
00078         int simdMaxBits() const { return simd_max_bits_; }
00079
00081         int sveVectorBits() const { return sve_vl_bits_; }
00082
00084         En_SimdKind bestSimdGeneric() const { return best_simd_generic_; }
00085
00091         En_SimdKind bestSimdFor(En_OpProfile prof) const;
00092
00093     private:
00094         void DetectCpuType_();
00095         void DetectSimd_x86_();
00096         void DetectSimd_arm_();
00097
00098         // x86 helpers
00099         static void cpuid_(int leaf, int subleaf, int regs[4]) noexcept;
00100         static uint64_t xgetbv_(uint32_t xcr) noexcept;
00101
00102     private:
00103         // State
00104         En_CpuType cpu_{ En_CpuType::x86 };
00105         int simd_max_bits_{ 0 };
00106         En_SimdKind best_simd_generic_{ En_SimdKind::None };
00107
00108         // x86 feature flags
00109         bool has_avx2_{ false };
00110         bool has_avx512f_{ false };
00111         bool has_avx512bw_{ false };
00112         bool has_amx_tile_{ false };
00113
00114         // ARM feature flags
00115         bool has_neon_{ false };
00116         bool has_sve_{ false };
00117         bool has_sve2_{ false };
00118         int sve_vl_bits_{ 0 };
00119     };
00120
00121 } // namespace ipm

```

4.15 ImageProcessorManager/ClpmEnv.h File Reference

Orchestrator singleton wrapping CPU and GPU environment probes and selection.

```
#include <string>
#include <vector>
#include <mutex>
#include <atomic>
#include "CIpmCpuEnv.h"
#include "CIpmGpuEnv.h"
```

Classes

- class [ipm::ClpmEnv](#)
Process-wide IPM environment (CPU + GPU).

4.15.1 Detailed Description

Orchestrator singleton wrapping CPU and GPU environment probes and selection.

Responsibilities:

- One-time initialization of ClpmCpuEnv and ClpmGpuEnv.
- GPU device enumeration and selection helpers (by name/CUDA/OpenCL).
- Logging of the detected environment (see implementation for formatted summary).

Threading:

- Instance returns a process-wide singleton and ensures Initialize is called once.
- All getters are thread-safe after initialization completes.

Definition in file [ClpmEnv.h](#).

4.16 ClpmEnv.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00015
00016 #include <string>
00017 #include <vector>
00018 #include <mutex>
00019 #include <atomic>
00020
00021 #if defined(_WIN32) || defined(_WIN64)
00022 #ifdef CSH_IPM_EXPORT
00023 #define IPM_API __declspec(dllexport)
00024 #else
00025 #define IPM_API __declspec(dllimport)
00026 #endif
00027 #else
00028 #define IPM_API __attribute__((visibility("default")))
```

```

00029 #endif
00030
00031 #include "CIpmCpuEnv.h"
00032 #include "CIpmGpuEnv.h"
00033
00034 namespace ipm {
00035
00036     class IPM_API CIpmEnv {
00037     public:
00038         static CIpmEnv& Instance();
00039
00040         void Initialize();
00041
00042         void Refresh();
00043
00044         // --- CPU facade ---
00045         EnCpuType cpuType() const noexcept { return cpu_.cpu(); }
00046
00047         // --- GPU facade (public members for convenience) ---
00048         CIpmCpuEnv cpu_;
00049         CIpmGpuEnv gpu_;
00050
00051         size_t getGpuCount() const { return gpu_.getGpuCount(); }
00052         GpuInfo getGpu(size_t idx) const { return gpu_.getGpu(idx); }
00053         int getSelectedIndex() const { return gpu_.getSelectedIndex(); }
00054         GpuInfo getSelected() const { return gpu_.getSelected(); }
00055
00056         bool selectByNameSubstring(const std::string& s, bool preferCUDA = true) { return
00057         gpu_.selectByNameSubstring(s, preferCUDA); }
00058         bool selectByCudaIndex(int i) { return gpu_.selectByCudaIndex(i); }
00059         bool selectByOpenCL(int p, int d) { return gpu_.selectByOpenCL(p, d); }
00060         void clearSelection() { gpu_.clearSelection(); }
00061
00062         std::string getSelectedOpenGLVersion() const { return gpu_.getSelectedOpenGLVersion(); }
00063
00064         SupportState selectedCudaState() const { return gpu_.selectedCudaState(); }
00065         SupportState selectedOpenCLState() const { return gpu_.selectedOpenCLState(); }
00066         SupportState selectedOpenGLState() const { return gpu_.selectedOpenGLState(); }
00067
00068     private:
00069         CIpmEnv() = default;
00070
00071         void setSelectedOpenGLVersion(const std::string& v) { gpu_.setSelectedOpenGLVersion(v); }
00072     }
00073
00074     void writeCpuGpuStatus();
00075
00076     private:
00077         std::atomic<bool> initialized_{ false };
00078     };
00079
00080 } // namespace ipm

```

4.17 ImageProcessorManager/CIpmFuncTable.h File Reference

Lazy-initialized registry mapping (backend,module,algIndex) -> callable [IpmFn](#) + UI name.

```

#include <vector>
#include <unordered_map>
#include <string>
#include <functional>
#include <mutex>
#include <cstdint>
#include <atomic>
#include "IpmTypes.h"

```

Classes

- class [ipmcommon::CIpmFuncTable](#)
Function Table Registry (singleton).

Functions

- void [ClpmFuncTable_RegisterDummyForDev](#) ()

Optional developer hook to add temporary registrations during development.

4.17.1 Detailed Description

Lazy-initialized registry mapping (backend,module,algIndex) -> callable [lpmFn](#) + UI name.

Design:

- Singleton accessed via [ClpmFuncTable::Instance\(\)](#) (thread-safe lazy init using `std::once_flag`).
- 3-level container: `funcTable_[backend][module]` is an `unordered_map<algIndex, FuncInfo>`.
- Built-ins:
 - Converter algorithms from [CConverter](#) are registered for `CPU_Serial` in `InitConverterFuncTable()`.
 - User plug-ins discovered/loaded via [ipm_internal::UserCustomLoader](#) into `User_Custom`.

See also

[ClpmUserCustomLoader.h](#) for the plug-in ABI and search logic.

[lpmTypes.h](#) for `#EnProcessBackend`, `#EnIpmModule`, [lpmStatus](#), [FuncInfo](#).

Definition in file [ClpmFuncTable.h](#).

4.17.2 Function Documentation

4.17.2.1 [ClpmFuncTable_RegisterDummyForDev\(\)](#)

```
void ClpmFuncTable_RegisterDummyForDev ()
```

Optional developer hook to add temporary registrations during development.

Note

No-op by default.

4.18 ClpmFuncTable.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00016
00017 #include <vector>
00018 #include <unordered_map>
00019 #include <string>
00020 #include <functional>
00021 #include <mutex>
00022 #include <cstdint>
00023 #include <atomic>
00024
00025 #include "IpmTypes.h"
00026
00027 namespace ipmcommon {
00028
00042     class CIpmFuncTable final {
00043     public:
00045         static CIpmFuncTable& Instance();
00046
00059         IpmStatus process(ipmcommon::EnProcessBackend backend,
00060             ipmcommon::EnIpmModule module,
00061             int algIndex,
00062             const csh_img::CSH_Image* in,
00063             csh_img::CSH_Image* out,
00064             void* param1,
00065             void* param2) const;
00066
00071         std::vector<std::pair<int, std::wstring>>
00072             getAlgorithmList(ipmcommon::EnProcessBackend backend,
00073                 ipmcommon::EnIpmModule module) const;
00074
00076         static const std::vector<std::wstring>& getBackendNames();
00078         static const std::vector<std::wstring>& getModuleNames();
00079
00081         static bool TryParseBackend(const std::wstring& name, ipmcommon::EnProcessBackend& out);
00083         static bool TryParseModule(const std::wstring& name, ipmcommon::EnIpmModule& out);
00084
00085     private:
00086         CIpmFuncTable(); // internal construction only
00087
00088         // Helpers
00089         static bool isValidBackendIndex_(int b);
00090         static bool isValidModuleIndex_(int m);
00091
00093         void ensureInitialized_() const;
00094
00096         IpmStatus registerFunc(ipmcommon::EnProcessBackend backend,
00097             ipmcommon::EnIpmModule module,
00098             int algIndex,
00099             IpmFn fn,
00100             std::wstring uiName);
00101
00102         // Initialization steps (called once)
00103         void InitFuncTable();
00104         void InitConverterFuncTable();
00105         void InitScalerFuncTable();
00106         void InitUserCustomFuncTable();
00107
00108     private:
00110         std::vector<std::vector<std::unordered_map<int, FuncInfo>>> funcTable_;
00111
00112         mutable std::mutex m_regMtx_;
00113         mutable std::once_flag init_once_;
00114         mutable std::atomic<bool> initialized_{ false };
00115     };
00116 } // namespace ipmcommon
00117
00122 void CIpmFuncTable_RegisterDummyForDev();
```

4.19 ImageProcessorManager/CIpmGpuEnv.h File Reference

GPU runtime/environment probe (OS adapters, CUDA/OpenCL presence, optional OpenGL probing) and selection.

```
#include <string>
#include <vector>
```

```
#include <mutex>
#include <atomic>
```

Classes

- struct `ipm::GpuInfo`
One GPU record with runtime capability flags.
- class `ipm::ClpmGpuEnv`
GPU environment probe and selection (non-singleton).

Enumerations

- enum class `ipm::En_GpuType` : int
Coarse GPU type bucket.
- enum class `ipm::SupportState` : int
Feature/runtime support state.

4.19.1 Detailed Description

GPU runtime/environment probe (OS adapters, CUDA/OpenCL presence, optional OpenGL probing) and selection.

Platforms:

- Windows: DXGI enumeration, CUDA via `nvcuda.dll`, OpenCL via `OpenCL.dll`, optional WGL probe.
- Linux: DRM (`/sys/class/drm`) enumeration, CUDA via `libcuda.so`, OpenCL via `libOpenCL.so`, optional X11/GLX or EGL/GLES probe (controlled by macros).

All public strings are UTF-8.

Definition in file `ClpmGpuEnv.h`.

4.20 ClpmGpuEnv.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00013
00014 #include <string>
00015 #include <vector>
00016 #include <mutex>
00017 #include <atomic>
00018
00019 #if defined(_WIN32) || defined(_WIN64)
00020 #ifdef CSH_IPM_EXPORT
00021 #define IPM_API __declspec(dllexport)
00022 #else
00023 #define IPM_API __declspec(dllimport)
00024 #endif
00025 #else
00026 #define IPM_API __attribute__((visibility("default")))
00027 #endif
00028
00029 namespace ipm {
00030
00032     enum class En_GpuType : int { None = 0, Internal, nVidia };
```

```

00033
00035 enum class SupportState : int { Unknown = 0, Available, NotAvailable };
00036
00045 struct GpuInfo {
00046     int id = -1;
00047     std::string name;
00048     std::string vendor;
00049     En_GpuType type = En_GpuType::None;
00050
00051     SupportState cudaState = SupportState::Unknown;
00052     SupportState openclState = SupportState::Unknown;
00053     SupportState openglState = SupportState::Unknown;
00054
00055     std::string cudaVersion;
00056     std::string openclVersion;
00057     std::string openglVersion;
00058
00059     int cudaDeviceIndex = -1;
00060     int openclPlatformIndex = -1;
00061     int openclDeviceIndex = -1;
00062 };
00063
00070 class IPM_API CIpmGpuEnv {
00071 public:
00072     CIpmGpuEnv() = default;
00073
00075     void Refresh();
00076
00077     // List / selection
00078     size_t getGpuCount() const;
00079     GpuInfo getGpu(size_t idx) const;
00080     int getSelectedIndex() const;
00081     GpuInfo getSelected() const;
00082
00089     bool selectByNameSubstring(const std::string& substr, bool preferCUDA = true);
00090
00092     bool selectByCudaIndex(int cudaIndex);
00093
00095     bool selectByOpenCL(int platformIndex, int deviceIndex);
00096
00098     void clearSelection();
00099
00101     void setSelectedOpenGLVersion(const std::string& glVersion);
00103     std::string getSelectedOpenGLVersion() const;
00104
00106     SupportState selectedCudaState() const;
00107     SupportState selectedOpenCLState() const;
00108     SupportState selectedOpenGLState() const;
00109
00110 private:
00111     // Internal steps
00112     void enumerateGpusOS();
00113     void selectOsActiveDisplayGpu(); // best-effort on Windows (DXGI + QueryDisplayConfig)
00114     void probeOpenCLRuntime(); // optional by macro (CSH_IPM_ENABLE_GL_PROBE)
00115     void probeCudaRuntime(); // CUDA Driver API (no CUDA SDK dependency)
00116     void probeOpenCLRuntime(); // OpenCL ICD
00117
00118     // List helpers
00119     void clearGpuListUnsafe();
00120     int addOrMergeByKey(const std::string& nameU8,
00121         const std::string& vendorU8,
00122         En_GpuType type);
00123
00124 private:
00125     mutable std::mutex mtx_;
00126     std::vector<GpuInfo> gpus_;
00127     int selected_{ -1 };
00128 };
00129
00130 } // namespace ipm

```

4.21 ImageProcessorManager/ClpmUserCustomLoader.h File Reference

Loader for User_Custom plug-ins and exposure of their registered algorithms.

```

#include <vector>
#include <string>
#include <mutex>
#include "IpmTypes.h"

```

Classes

- class [ipm_internal::UserCustomLoader](#)
Singleton user plug-in loader for the User_Custom module.

4.21.1 Detailed Description

Loader for User_Custom plug-ins and exposure of their registered algorithms.

ABI (expected exported symbols in the plug-in shared library):

- `int ipm_user_custom_register(const AlgEntry** out, int* cnt);`
 - On success, returns 0 and sets (*out, *cnt) to a contiguous array of [AlgEntry](#).
- `void ipm_user_custom_unregister();`

Search locations:

- Windows: executable directory and /plugins subdir (see .cpp).
- Linux: <exe>/../lib, <exe>, <exe>/plugins, and dynamic loader fallback.

Threading:

- `loadOnce()` is guarded by `std::once_flag` and is idempotent per process.
- `unload()` is safe to call during shutdown; it frees the library and clears entries.

Definition in file [ClpmUserCustomLoader.h](#).

4.22 ClpmUserCustomLoader.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00019
00020 #include <vector>
00021 #include <string>
00022 #include <mutex>
00023 #include "IpmTypes.h"
00024
00025 // Internal implementation namespace
00026 namespace ipm_internal {
00027
00036     class UserCustomLoader {
00037     public:
00039         static UserCustomLoader& instance();
00040
00047         int loadOnce();
00048
00050         const std::vector<AlgEntry>& entries() const { return entries_; }
00051
00053         void unload();
00054
00055     private:
00056         UserCustomLoader() = default;
00057         ~UserCustomLoader();
00058
00059         // Platform library handle
00060         void* hmod_ = nullptr;
```



```

00061
00062     // Plug-in ABI types
00063     using RegisterFn = int (*) (const AlgEntry** out, int* cnt);
00064     using UnregisterFn = void (*) ();
00065
00066     RegisterFn reg_ = nullptr;
00067     UnregisterFn unreg_ = nullptr;
00068
00069     std::vector<AlgEntry> entries_;
00070     std::once_flag once_;
00071     std::mutex mtx_;
00072
00073     bool tryOpen(const std::vector<std::string>& names);
00074 };
00075
00076
00077 } // namespace ipm_internal

```

4.23 CConverter.h

```

00001 #pragma once
00002 #include <vector>
00003 #include <memory>
00004 #include <mutex>
00005 #include <string>
00006 #include "../IpmTypes.h" // IpmFn, IpmStatus, FuncInfo, AlgEntry declaration
00007 #include "../CIpmEnv.h" // Uses hasGLCompute(), hasCUDA()
00008
00009 // Forward declarations of worker classes (included in .cpp)
00010 class CCpuSerialConverter;
00011 class CCpuParaConverter;
00012 class CGpuGLComputeConverter;
00013 class CGpuCLConverter;
00014 class CGpuCudaConverter;
00015
00022 class CConverter final {
00023 public:
00024     // Enum for the third argument of registerFunc in the function table
00025     enum class Ipm_Converter_Func : int {
00026         YUV422_8bit_To_RGB888 = 0,
00027         YUV422_8bit_To_BGR888,
00028         RGB888_To_Gray8, // (Only shown as Gray8, the actual literals could be freely chosen such as
00029         L"Gray8")
00030         Count
00031     };
00032
00033     // Singleton Instance
00034     static CConverter& Instance();
00035
00036     // Catalog for each backend (used for registering function table)
00037     const std::vector<AlgEntry>& CpuSerialList() const { return listCpuSerial_; }
00038     const std::vector<AlgEntry>& CpuParallelList() const { return listCpuParallel_; }
00039     const std::vector<AlgEntry>& GLComputeList() const { return listGLCompute_; }
00040     const std::vector<AlgEntry>& OpenCLList() const { return listOpenCL_; }
00041     const std::vector<AlgEntry>& CudaList() const { return listCuda_; }
00042 private:
00043     CConverter(); // Singleton: must not be instantiated outside
00044     void AddFunctions(); // Uploads all algorithms to each backend
00045
00046     // Lazy worker creation per backend (thread-safe)
00047     std::shared_ptr<CCpuSerialConverter> getCpuSerial_();
00048     std::shared_ptr<CCpuParaConverter> getCpuParallel_();
00049     std::shared_ptr<CGpuGLComputeConverter> getGLCompute_();
00050     std::shared_ptr<CGpuCudaConverter> getCuda_();
00051
00052     // enum -> calls for actual worker methods (lambdas)
00053     IpmFn makeCpuSerial_(Ipm_Converter_Func f);
00054     IpmFn makeCpuParallel_(Ipm_Converter_Func f);
00055     IpmFn makeGLCompute_(Ipm_Converter_Func f);
00056     IpmFn makeOpenCL_NotAvailable_(); // OpenCL /A
00057     IpmFn makeCuda_(Ipm_Converter_Func f);
00058
00059 private:
00060     // Catalog(to be read by the function table for registration)
00061     std::vector<AlgEntry> listCpuSerial_;
00062     std::vector<AlgEntry> listCpuParallel_;
00063     std::vector<AlgEntry> listGLCompute_;
00064     std::vector<AlgEntry> listOpenCL_;
00065     std::vector<AlgEntry> listCuda_;
00066
00067     // Worker instances (lazily created)
00068     std::shared_ptr<CCpuSerialConverter> cpuSerial_;
00069     std::shared_ptr<CCpuParaConverter> cpuParallel_;

```

```

00070 //std::shared_ptr<CGpuGlcComputeConverter> glCompute_;
00071 //std::shared_ptr<CGpuCudaConverter> cuda_;
00072
00073 // Concurrent access guard
00074 mutable std::mutex mtX_;
00075 };

```

4.24 CCpuParaConverter.h

```

00001 #pragma once
00002 #include <stdint>
00003 #include "../IpmTypes.h"
00004
00010 class CCpuParaConverter final {
00011 public:
00012     CCpuParaConverter();
00013
00014     // YUV422 8bit -> RGB888 (out.format must be RGB888)
00015     int ConvertYUV422_8_To_RGB888(const csh_img::CSH_Image* in,
00016         csh_img::CSH_Image* out,
00017         void* /*param1*/, void* /*param2*/);
00018
00019     // YUV422 8bit -> BGR888 (out.format must be BGR888)
00020     int ConvertYUV422_8_To_BGR888(const csh_img::CSH_Image* in,
00021         csh_img::CSH_Image* out,
00022         void* /*param1*/, void* /*param2*/);
00023
00024     // RGB888/BGR888 -> Gray8 (automatic input format detection)
00025     int ConvertRGB888_To_Gray8(const csh_img::CSH_Image* in,
00026         csh_img::CSH_Image* out,
00027         void* /*param1*/, void* /*param2*/);
00028
00029 private:
00030     // Shared validation (in/out formats, size, buffer)
00031     static int validateYUV422ToRGB_(const csh_img::CSH_Image* in,
00032         csh_img::CSH_Image* out); // Removes bgrOrder
00033
00034     static int validateRGB888ToGray8_(const csh_img::CSH_Image* in,
00035         csh_img::CSH_Image* out);
00036
00037     // Internal conversion cores
00038     // Automatic output according to out.getFormat(): RGB888->RGB, BGR888->BGR
00039     static void yuv422_to_rgb888_core_(const csh_img::CSH_Image& in,
00040         csh_img::CSH_Image& out);
00041
00042     static void rgb888_to_gray8_core_(const csh_img::CSH_Image& in,
00043         csh_img::CSH_Image& out);
00044 };

```

4.25 CCpuSerialConverter.h

```

00001 #pragma once
00002 #include <stdint>
00003 #include "../IpmTypes.h"
00004
00010 class CCpuSerialConverter final {
00011 public:
00012     CCpuSerialConverter();
00013
00014     // YUV422 8bit -> RGB888 (out.format must be RGB888)
00015     int ConvertYUV422_8_To_RGB888(const csh_img::CSH_Image* in,
00016         csh_img::CSH_Image* out,
00017         void* /*param1*/, void* /*param2*/);
00018
00019     // YUV422 8bit -> BGR888 (out.format must be BGR888)
00020     int ConvertYUV422_8_To_BGR888(const csh_img::CSH_Image* in,
00021         csh_img::CSH_Image* out,
00022         void* /*param1*/, void* /*param2*/);
00023
00024     // RGB888/BGR888 -> Gray8 (automatic input format detection)
00025     int ConvertRGB888_To_Gray8(const csh_img::CSH_Image* in,
00026         csh_img::CSH_Image* out,
00027         void* /*param1*/, void* /*param2*/);
00028
00029 private:
00030     // Shared validation (in/out formats, size, buffer)
00031     static int validateYUV422ToRGB_(const csh_img::CSH_Image* in,
00032         csh_img::CSH_Image* out); // Removes bgrOrder

```

```

00033
00034     static int validateRGB888ToGray8_(const csh_img::CSH_Image* in,
00035         csh_img::CSH_Image* out);
00036
00037     // Internal conversion cores
00038     // Automatic output according to out.getFormat(): RGB888->RGB, BGR888->BGR
00039     static void yuv422_to_rgb888_core_(const csh_img::CSH_Image& in,
00040         csh_img::CSH_Image& out);
00041
00042     static void rgb888_to_gray8_core_(const csh_img::CSH_Image& in,
00043         csh_img::CSH_Image& out);
00044 };

```

4.26 CGpuClConverter.h

```
00001 #pragma once
```

4.27 CGpuCudaConverter.h

```
00001 #pragma once
```

4.28 CGpuGLComputeConverter.h

```
00001 #pragma once
```

4.29 ImageProcessorManager/lpmClamp.h File Reference

Tiny header-only clamping and saturating-cast utilities optimized for image pipelines.

```

#include <stdint>
#include <type_traits>
#include <limits>

```

Functions

- `template<typename T>`
`ipm::util::__attribute__ ((always_inline)) const expr T clamp(const T &v`
Generic clamp to [lo, hi].

4.29.1 Detailed Description

Tiny header-only clamping and saturating-cast utilities optimized for image pipelines.

Key points:

- `constexpr + always-inline` to encourage full inlining in hot loops.
- Branch-minimized fixed-range helpers for 8/10/12/16b common pixel depths.
- Type-safe `saturated_cast<D, S>()` that gracefully clamps across signed/unsigned and float/int.

Definition in file [lpmClamp.h](#).

4.29.2 Function Documentation

4.29.2.1 `__attribute__()`

```
template<typename T>
ipm::util::__attribute__ (
    (always_inline) ) const & [inline]
```

Generic clamp to [lo, hi].

Template Parameters

<i>T</i>	Arithmetic type (integral or floating).
----------	-----------------------------------------

Parameters

<i>v</i>	Value to clamp.
<i>lo</i>	Lower bound (inclusive).
<i>hi</i>	Upper bound (inclusive).

Returns

Clamped value.

Warning

Assumes `lo <= hi`.

4.30 `ipmClamp.h`

[Go to the documentation of this file.](#)

```
00001 #pragma once
00011
00012 #include <cstdint>
00013 #include <type_traits>
00014 #include <limits>
00015
00016 #if defined(_MSC_VER)
00017 #define IPM_FORCE_INLINE __forceinline
00018 #else
00019 #define IPM_FORCE_INLINE inline __attribute__((always_inline))
00020 #endif
00021
00022 namespace ipm {
00023     namespace util {
00024
00025         template <typename T>
00035         IPM_FORCE_INLINE constexpr T clamp(const T& v, const T& lo, const T& hi) noexcept {
00036             return (v < lo) ? lo : ((v > hi) ? hi : v);
00037         }
00038
00041
00043         IPM_FORCE_INLINE constexpr std::uint8_t clamp_u8(std::int32_t v) noexcept {
00044             return static_cast<std::uint8_t>(v < 0 ? 0 : (v > 255 ? 255 : v));
00045         }
00046
00048         IPM_FORCE_INLINE constexpr std::uint16_t clamp_u16(std::int32_t v) noexcept {
```

```

00049         return static_cast<std::uint16_t>(v < 0 ? 0 : (v > 65535 ? 65535 : v));
00050     }
00051
00053     IPM_FORCE_INLINE constexpr std::uint16_t clamp_u10(std::int32_t v) noexcept {
00054         return static_cast<std::uint16_t>(v < 0 ? 0 : (v > 1023 ? 1023 : v));
00055     }
00056
00058     IPM_FORCE_INLINE constexpr std::uint16_t clamp_u12(std::int32_t v) noexcept {
00059         return static_cast<std::uint16_t>(v < 0 ? 0 : (v > 4095 ? 4095 : v));
00060     }
00062
00069     template <typename Dst, typename Src>
00070     IPM_FORCE_INLINE constexpr Dst saturated_cast(Src v) noexcept {
00071         static_assert(std::is_arithmetic<Src>::value, "Src must be arithmetic");
00072         static_assert(std::is_arithmetic<Dst>::value, "Dst must be arithmetic");
00073
00074         constexpr long double Dmin = static_cast<long double>(std::numeric_limits<Dst>::lowest());
00075         constexpr long double Dmax = static_cast<long double>(std::numeric_limits<Dst>::max());
00076         const long double lv = static_cast<long double>(v);
00077
00078         if (lv < Dmin) return static_cast<Dst>(Dmin);
00079         if (lv > Dmax) return static_cast<Dst>(Dmax);
00080         return static_cast<Dst>(lv);
00081     }
00082
00085     IPM_FORCE_INLINE constexpr std::uint8_t sat_u8(int v) noexcept { return
saturated_cast<std::uint8_t>(v); }
00086     IPM_FORCE_INLINE constexpr std::uint16_t sat_u16(int v) noexcept { return
saturated_cast<std::uint16_t>(v); }
00087     IPM_FORCE_INLINE constexpr std::int16_t sat_i16(int v) noexcept { return
saturated_cast<std::int16_t>(v); }
00089
00090     } // namespace util
00091 } // namespace ipm

```

4.31 ImageProcessorManager/lpmStringUtils.h File Reference

Lightweight, dependency-minimal UTF-8/UTF-16 helpers and enum stringifiers for IPM.

```

#include <string>
#include <cwchar>
#include <codecvt>
#include <locale>
#include "CIpmEnv.h"
#include "CIpmGpuEnv.h"

```

Functions

- std::wstring [ipm::str::u8_to_w](#) (const std::string &s)
Convert UTF-8 string to wide (UTF-16 on Windows, UTF-32 on Linux).
- std::string [ipm::str::w_to_u8](#) (const std::wstring &w)
Convert wide string to UTF-8.
- const wchar_t * [ipm::str::cpu_to_w](#) (En_CpuType c)
Convert CPU type to wide literal.
- const wchar_t * [ipm::str::gpuType_to_w](#) (En_GpuType t)
Convert GPU kind to wide literal.
- const wchar_t * [ipm::str::state_to_w](#) (SupportState s)
Convert feature/support state to wide literal.

4.31.1 Detailed Description

Lightweight, dependency-minimal UTF-8/UTF-16 helpers and enum stringifiers for IPM.

- Windows uses Win32 WideChar APIs (no extra deps).
- Linux/macOS fallback uses `std::wstring_convert` (codecvt) for simple tools-layer conversions.

Also provides `wchar_t*` stringifiers for public IPM enums declared in [ClpmEnv.h](#) and [ClpmGpuEnv.h](#) (e.g., [ipm::En_CpuType](#), [ipm::En_GpuType](#), [ipm::SupportState](#)).

Definition in file [lpmStringUtils.h](#).

4.31.2 Function Documentation

4.31.2.1 `u8_to_w()`

```
std::wstring ipm::str::u8_to_w (  
    const std::string & s) [inline]
```

Convert UTF-8 string to wide (UTF-16 on Windows, UTF-32 on Linux).

Parameters

s	UTF-8 input.
---	--------------

Returns

Wide string; empty on failure.

Definition at line 34 of file [lpmStringUtils.h](#).

```
00034                                     {  
00035 #if defined(_WIN32) || defined(_WIN64)  
00036     if (s.empty()) return L"";  
00037     int len = ::MultiByteToWideChar(CP_UTF8, 0, s.c_str(), -1, nullptr, 0);  
00038     if (len <= 0) return L"";  
00039     std::wstring w(static_cast<size_t>(len) - 1, L'\0');  
00040     ::MultiByteToWideChar(CP_UTF8, 0, s.c_str(), -1, w.data(), len);  
00041     return w;  
00042 #else  
00043     try {  
00044         std::wstring_convert<std::codecvt_utf8_utf16<wchar_t>> conv;  
00045         return conv.from_bytes(s);  
00046     }  
00047     catch (...) { return L""; }  
00048 #endif  
00049 }
```

4.31.2.2 `w_to_u8()`

```
std::string ipm::str::w_to_u8 (  
    const std::wstring & w) [inline]
```

Convert wide string to UTF-8.

Parameters

w	Wide input.
---	-------------

Returns

UTF-8 string; empty on failure.

Definition at line 56 of file [lpmStringUtils.h](#).

```
00056                                     {
00057 #if defined(_WIN32) || defined(_WIN64)
00058     if (w.empty()) return {};
00059     int len = ::WideCharToMultiByte(CP_UTF8, 0, w.c_str(), -1, nullptr, 0, nullptr, nullptr);
00060     if (len <= 0) return {};
00061     std::string s(static_cast<size_t>(len) - 1, '\0');
00062     ::WideCharToMultiByte(CP_UTF8, 0, w.c_str(), -1, s.data(), len, nullptr, nullptr);
00063     return s;
00064 #else
00065     try {
00066         std::wstring_convert<std::codecvt_utf8_utf16<wchar_t>> conv;
00067         return conv.to_bytes(w);
00068     }
00069     catch (...) { return {}; }
00070 #endif
00071 }
```

4.32 lpmStringUtils.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00012
00013 #include <string>
00014 #include <cwchar>
00015
00016 #if defined(_WIN32) || defined(_WIN64)
00017 #ifndef NOMINMAX
00018 #define NOMINMAX
00019 #endif
00020 #include <windows.h>
00021 #else
00022 // codecvt is deprecated but sufficient for a small tool/helper (no heavy ICU dependency).
00023 #include <codecvt>
00024 #include <locale>
00025 #endif
00026
00027 namespace ipm::str {
00028
00034     inline std::wstring u8_to_w(const std::string& s) {
00035 #if defined(_WIN32) || defined(_WIN64)
00036     if (s.empty()) return L"";
00037     int len = ::MultiByteToWideChar(CP_UTF8, 0, s.c_str(), -1, nullptr, 0);
00038     if (len <= 0) return L"";
00039     std::wstring w(static_cast<size_t>(len) - 1, L'\0');
00040     ::MultiByteToWideChar(CP_UTF8, 0, s.c_str(), -1, w.data(), len);
00041     return w;
00042 #else
00043     try {
00044         std::wstring_convert<std::codecvt_utf8_utf16<wchar_t>> conv;
00045         return conv.from_bytes(s);
00046     }
00047     catch (...) { return L""; }
00048 #endif
00049     }
00050
00056     inline std::string w_to_u8(const std::wstring& w) {
00057 #if defined(_WIN32) || defined(_WIN64)
00058     if (w.empty()) return {};
00059     int len = ::WideCharToMultiByte(CP_UTF8, 0, w.c_str(), -1, nullptr, 0, nullptr, nullptr);
00060     if (len <= 0) return {};
00061     std::string s(static_cast<size_t>(len) - 1, '\0');
00062     ::WideCharToMultiByte(CP_UTF8, 0, w.c_str(), -1, s.data(), len, nullptr, nullptr);
00063     return s;
00064 #else
00065     try {
00066         std::wstring_convert<std::codecvt_utf8_utf16<wchar_t>> conv;
```

```

00067         return conv.to_bytes(w);
00068     }
00069     catch (...) { return {}; }
00070 #endif
00071 }
00072
00073 } // namespace ipm::str
00074
00075 // ===== Enum stringifiers depend on IPM public enums =====
00076 #include "CipmEnv.h" // for En_CpuType
00077 #include "CipmGpuEnv.h" // for En_GpuType / SupportState
00078
00079 namespace ipm::str {
00080
00081     inline const wchar_t* cpu_to_w(En_CpuType c) {
00082         switch (c) {
00083             case En_CpuType::x86: return L"x86";
00084             case En_CpuType::x86_64: return L"x86_64";
00085             case En_CpuType::ARM8: return L"ARMv8";
00086             case En_CpuType::ARM9: return L"ARMv7/9";
00087             default: return L"Unknown";
00088         }
00089     }
00090
00091     inline const wchar_t* gpuType_to_w(En_GpuType t) {
00092         switch (t) {
00093             case En_GpuType::Internal: return L"Internal";
00094             case En_GpuType::nVidia: return L"NVIDIA";
00095             case En_GpuType::None: return L"None";
00096             default: return L"Unknown";
00097         }
00098     }
00099
00100     inline const wchar_t* state_to_w(SupportState s) {
00101         switch (s) {
00102             case SupportState::Available: return L"Available";
00103             case SupportState::NotAvailable: return L"NotAvailable";
00104             default: return L"Unknown";
00105         }
00106     }
00107
00108 } // namespace ipm::str
00109
00110 } // namespace ipm::str
00111

```

4.33 ImageProcessorManager/lpmTypes.h File Reference

Core IPM type aliases, enums, and small PODs used across the image processing modules.

```

#include <functional>
#include <string>
#include <CSH_Image.h>

```

Classes

- struct [FuncInfo](#)
Metadata for a registered algorithm.
- struct [AlgEntry](#)
Entry stored in a module catalog.

Typedefs

- using [lpmFn](#) = std::function<int(const [csh_img::CSH_Image*](#), [csh_img::CSH_Image*](#), void*, void*)>
Canonical function signature for all processing algorithms.

Enumerations

- enum class [ipmcommon::EnProcessBackend](#) : int {
 [CPU_Serial](#) = 0 , [CPU_Parallel](#) , [GPU_GL_Compute](#) , [GPU_OpenCL](#) ,
 [GPU_CUDA](#) , [Count](#) }
 Compute backend options (UI first list).
- enum class [ipmcommon::EnIpmModule](#) : int { [Converter](#) = 0 , [Scaler](#) , [Splitter](#) , [User_Custom](#) , [Count](#) }
 High-level module groups (UI second list).
- enum class [IpmStatus](#) : int {
 [NotAvailable](#) = 0 , [OK](#) , [Err_InvalidBackend](#) , [Err_InvalidModule](#) ,
 [Err_AlgorithmNotFound](#) , [Err_InvalidSize](#) , [Err_InvalidFormat](#) , [Err_NullFunction](#) ,
 [Err_NullImage](#) , [Err_Internal](#) , [IsDevelopping](#) }
 IPM status / error codes.

4.33.1 Detailed Description

Core IPM type aliases, enums, and small PODs used across the image processing modules.

This header centralizes:

- The canonical function signature for all processing algorithms ([IpmFn](#)).
- Frontend/UI enumerations for backends and modules ([ipmcommon::EnProcessBackend](#), [ipmcommon::EnIpmModule](#)).
- Canonical status codes ([IpmStatus](#)) returned by algorithms and dispatchers.
- Lightweight structures that describe registered functions and catalog entries ([FuncInfo](#), [AlgEntry](#)).

See also

[ClpmFuncTable.h](#) For the registry that uses these types.

[ClimageProcessMng.h](#) For the pipeline manager that consumes registered functions.

Definition in file [IpmTypes.h](#).

4.33.2 Typedef Documentation

4.33.2.1 IpmFn

```
using IpmFn = std::function<int(const csh_img::CSH_Image*, csh_img::CSH_Image*, void*, void*)>
```

Canonical function signature for all processing algorithms.

The function must return an [IpmStatus](#) cast to int (see [IpmStatus](#)). Ownership:

- `in` is a borrowed pointer (may be null for source-less stages; most algorithms require it).
- `out` must be a valid, writable image (caller typically allocates; algorithms may reallocate if needed).
- `p1`, `p2` are opaque user parameters (algorithm-specific).

Parameters

<i>in</i>	Input image (may be nullptr only for specific source operators).
<i>out</i>	Output image (must not be nullptr).
<i>p1</i>	Optional parameter block (algorithm-specific).
<i>p2</i>	Optional parameter block (algorithm-specific).

Returns

int Status code compatible with [lpmStatus](#).

Definition at line 37 of file [lpmTypes.h](#).

4.33.3 Enumeration Type Documentation

4.33.3.1 EnIpmModule

```
enum class ipmcommon::EnIpmModule : int [strong]
```

High-level module groups (UI second list).

Modules group related algorithms. Registration and lookup use (backend,module,algIndex).

Note

User_Custom must remain the last regular module so plug-ins append cleanly.

Enumerator

Converter	Color space / pixel format converters.
Scaler	Resamplers / scalers.
Splitter	Stream/image split utilities.
User_Custom	User plug-in module bucket (always last before Count).

Definition at line 64 of file [lpmTypes.h](#).

```
00064                                     : int {
00065     Converter = 0,
00066     Scaler,
00067     Splitter,
00068
00069     User_Custom,
00070     Count
00071 };
```

4.33.3.2 EnProcessBackend

```
enum class ipmcommon::EnProcessBackend : int [strong]
```

Compute backend options (UI first list).

These enumerators segment the same algorithm catalog by execution target. The function table stores separate maps per backend.

See also

ClpmFuncTable

Enumerator

CPU_Serial	Single-threaded CPU path.
CPU_Parallel	Multi-threaded CPU path (TBB/OpenMP/tasking; TBD).
GPU_GL_Compute	GPU via OpenGL Compute or GLES compute (if enabled).
GPU_OpenCL	GPU via OpenCL (if available).
GPU_CUDA	GPU via CUDA (if driver/runtime available).

Definition at line 49 of file [lpmTypes.h](#).

```

00049                                     : int {
00050     CPU_Serial = 0,
00051     CPU_Parallel,
00052     GPU_GL_Compute,
00053     GPU_OpenCL,
00054     GPU_CUDA,
00055     Count
00056 };

```

4.33.3.3 IpmStatus

```
enum class IpmStatus : int [strong]
```

IPM status / error codes.

Returned by algorithm functions and by #CipmFuncTable::process.

Enumerator

NotAvailable	Feature/backend not available on this system.
OK	Success.
Err_InvalidBackend	Backend out of range / not registered.
Err_InvalidModule	Module out of range / not registered.
Err_AlgNotFound	Algorithm index not found for (backend,module).
Err_InvalidSize	Unsupported or mismatched size.
Err_InvalidFormat	Unsupported or mismatched pixel format.
Err_NullFunction	Function pointer not set.
Err_NullImage	Null in or out where required.
Err_Internal	Exception or internal fault.
IsDevelopping	Placeholder status for WIP paths.

Definition at line 80 of file [lpmTypes.h](#).

```

00080                                     : int {
00081     NotAvailable = 0,
00082     OK,
00083     Err_InvalidBackend,
00084     Err_InvalidModule,
00085     Err_AlgNotFound,
00086     Err_InvalidSize,
00087     Err_InvalidFormat,
00088     Err_NullFunction,
00089     Err_NullImage,
00090     Err_Internal,
00091     IsDevelopping
00092 };

```

4.34 IpmTypes.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00015
00016 #include <functional>
00017 #include <string>
00018 #include <CSH_Image.h>
00019
00020 using namespace csh_img;
00021
00037 using IpmFn = std::function<int(const csh_img::CSH_Image*, csh_img::CSH_Image*, void*, void*)>;
00038
00039 namespace ipmcommon {
00040
00049     enum class EnProcessBackend : int {
00050         CPU_Serial = 0,
00051         CPU_Parallel,
00052         GPU_GL_Compute,
00053         GPU_OpenCL,
00054         GPU_CUDA,
00055         Count
00056     };
00057
00064     enum class EnIpmModule : int {
00065         Converter = 0,
00066         Scaler,
00067         Splitter,
00068
00069         User_Custom,
00070         Count
00071     };
00072
00073 } // namespace ipmcommon
00074
00080 enum class IpmStatus : int {
00081     NotAvailable = 0,
00082     OK,
00083     Err_InvalidBackend,
00084     Err_InvalidModule,
00085     Err_AlgorithmNotFound,
00086     Err_InvalidSize,
00087     Err_InvalidFormat,
00088     Err_NullFunction,
00089     Err_NullImage,
00090     Err_Internal,
00091     IsDeveloping
00092 };
00093
00099 struct FuncInfo {
00100     IpmFn      fn;
00101     std::wstring uiName;
00102 };
00103
00109 struct AlgEntry {
00110     int      alg;
00111     FuncInfo func;
00112 };
```

4.35 ClpmUserCustom.h

```
00001 #pragma once
00002 #include <IpmTypes.h>    // AlgEntry / FuncInfo / IpmFn
00003
00004 // export
00005 #if defined(_WIN32)
00006 #if defined(IPM_USER_CUSTOM_EXPORTS)
00007 #define IPM_USER_CUSTOM_API __declspec(dllexport)
00008 #else
00009 #define IPM_USER_CUSTOM_API __declspec(dllimport)
00010 #endif
00011 #else
00012 #define IPM_USER_CUSTOM_API __attribute__((visibility("default")))
00013 #endif
00014
00015 enum class Ipm_UserCustom_Func : int {
00016     BGR8882Gray8 = 0,
00017     Scaler,
00018
00019     Count
```

```

00020 };
00021
00022 // C-ABI: Ȳ ă
00023 extern "C" {
00024
00025     // : AlgEntry /
00026     IPM_USER_CUSTOM_API int ipm_user_custom_register(const AlgEntry** out_entries, int* out_count);
00027
00028     // : ( )
00029     IPM_USER_CUSTOM_API void ipm_user_custom_unregister();
00030
00031 } // extern "C"

```

4.36 utility/CSH_Image/CSH_Image.h File Reference

Cross-platform C++17 image container class for camera/vision pipelines.

```

#include <cstdint>
#include <cstddef>
#include <memory>
#include <string>
#include <vector>
#include <stdexcept>
#include <fstream>
#include <type_traits>
#include <filesystem>

```

Classes

- class [csh_img::CSH_Image](#)
Image container with explicit format metadata and flexible buffer ownership.

Enumerations

- enum class [csh_img::En_ImageFormat](#) : uint32_t {
[Bayer8](#) = 100 , [Gray8](#) , [Bayer10](#) = 200 , [Bayer12](#) ,
[Bayer14](#) , [Bayer16](#) , [Gray10](#) , [Gray12](#) ,
[Gray14](#) , [Gray16](#) , [YUV422](#) , [RGB565](#) ,
[YUYV444](#) = 300 , [RGB888](#) , [BGR888](#) }
Logical pixel/container formats (not colorspace conversions).
- enum class [csh_img::En_ImagePattern](#) : uint32_t
Pixel order / component layout associated with a format.
- enum class [csh_img::En_ImageMemoryAlign](#) : uint32_t { [Packed](#) = 0 , [YYYYUUUUVVVV](#) = 10 ,
[YYYYVVVVUUUU](#) , [UUUUVVVVYYYY](#) , [VVVVUUUUYYYY](#) , [RRRRGGGGBBBB](#) = 20 , [BBBBGGGGRRRR](#)
, [YYYYUVUV](#) = 30 , [YYYYVUVU](#) }
Memory layout / plane arrangement.
- enum class [csh_img::CopyMode](#) : uint32_t
Copy semantics for CSH_Image::copy and related APIs.

4.36.1 Detailed Description

Cross-platform C++17 image container class for camera/vision pipelines.

- Dynamic library ready (dllexport/dllimport on Windows, default visibility on Linux).
- Smart-pointer owned buffer with view pointer managed internally (no raw address moves).
- Shallow / deep copy semantics.
- Robust, forward/backward-compatible binary persistence via tagged fields (TLV).
- Optional zero-copy OpenCV cv::Mat view (CSH_IMAGE_WITH_OPENCV).

Definition in file [CSH_Image.h](#).

4.36.2 Enumeration Type Documentation

4.36.2.1 CopyMode

```
enum class csh_img::CopyMode : uint32_t [strong]
```

Copy semantics for CSH_Image::copy and related APIs.

- CopyMode::MetaOnly : metadata only (no buffer).
- CopyMode::Shallow : share the same buffer (shared_ptr) and view.
- CopyMode::Deep : copy bytes into an already allocated destination buffer.

Definition at line 112 of file [CSH_Image.h](#).

```
00112 : uint32_t { MetaOnly = 0, Shallow, Deep };
```

4.36.2.2 En_ImageFormat

```
enum class csh_img::En_ImageFormat : uint32_t [strong]
```

Logical pixel/container formats (not colorspace conversions).

Values are grouped by typical container bit depth:

- 100s: 8-bit family (e.g., Bayer8, Gray8)
- 200s: 16-bit/packed 10/12/14-bit family, YUV422, RGB565
- 300s: 24-bit family (RGB888/BGR888/YUYV444)

Enumerator

Bayer8	8-bit Bayer mosaic (pattern in En_ImagePattern).
Gray8	8-bit grayscale.
Bayer10	10-bit Bayer stored in 16-bit container or packed as policy.
Bayer12	12-bit Bayer stored in 16-bit container or packed as policy.
Bayer14	14-bit Bayer stored in 16-bit container or packed as policy.
Bayer16	16-bit Bayer, full 16-bit container.
Gray10	10-bit gray stored in 16-bit container or packed.
Gray12	12-bit gray stored in 16-bit container or packed.
Gray14	14-bit gray stored in 16-bit container or packed.
Gray16	16-bit gray, full 16-bit container.
YUV422	Packed 4:2:2 (2 bytes per pixel pair).
RGB565	16-bit RGB (5-6-5).
YUYV444	38-bit container variant (API compatibility).
RGB888	8-bit per channel RGB.
BGR888	8-bit per channel BGR.

Definition at line 48 of file [CSH_Image.h](#).

```

00048                                     : uint32_t {
00049     // 8-bit container group (100+)
00050     Bayer8 = 100,
00051     Gray8,
00052
00053     // 16-bit container group (200+)
00054     Bayer10 = 200,
00055     Bayer12,
00056     Bayer14,
00057     Bayer16,
00058     Gray10,
00059     Gray12,
00060     Gray14,
00061     Gray16,
00062     YUV422,
00063     RGB565,
00064
00065     // 24-bit container group (300+)
00066     YUYV444 = 300,
00067     RGB888,
00068     BGR888,
00069 };

```

4.36.2.3 En_ImageMemoryAlign

```
enum class csh_img::En_ImageMemoryAlign : uint32_t [strong]
```

Memory layout / plane arrangement.

Note

Current implementation mainly operates on Packed. Other values exist for forward compatibility and future planar/semi-planar support.

Enumerator

Packed	Interleaved/packed bytes in a single plane.
--------	---------------------------------------------

Definition at line 94 of file [CSH_Image.h](#).

```
00094                                     : uint32_t {
00095     Packed = 0,
00096     // planar examples
00097     YYYYUUUVVVV = 10, YYYYVVVVUUUU, UUUUVVVVYYYY, VVVUUUUYYYYY,
00098     // planar RGB
00099     RRRRGGGBBBB = 20, BBBBGGGRRRR,
00100     // semi-planar examples
00101     YYYYUVUV = 30, YYYYVUVU,
00102     };
```

4.36.2.4 En_ImagePattern

```
enum class csh_img::En_ImagePattern : uint32_t [strong]
```

Pixel order / component layout associated with a format.

For Bayer formats, set the CFA order; for YUV422, the packed ordering; for 24-bit RGB/BGR, choose channel order.

Definition at line 78 of file [CSH_Image.h](#).

```
00078                                     : uint32_t {
00079     // Bayer
00080     RRGB = 0, GRBG, BGGR, GBRG,
00081     // YUV422 (packed)
00082     YUYV = 10, UYVY, YVYU, VYUY,
00083     // RGB/BGR (24-bit)
00084     RGB = 20, BGR,
00085     };
```

4.37 CSH_Image.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00012
00013 #include <cstdint>
00014 #include <cstdint>
00015 #include <memory>
00016 #include <string>
00017 #include <vector>
00018 #include <stdexcept>
00019 #include <fstream>
00020 #include <type_traits>
00021 #include <filesystem>
00022
00023 #if defined(_WIN32) || defined(_WIN64)
00024 #   if defined(CSH_IMAGE_EXPORT)
00025 #       define CSH_IMAGE_API __declspec(dllexport)
00026 #   else
00027 #       define CSH_IMAGE_API __declspec(dllimport)
00028 #   endif
00029 #else
00030 #   define CSH_IMAGE_API __attribute__((visibility("default")))
00031 #endif
00032
00033 #ifdef CSH_IMAGE_WITH_OPENCV
00034 #include <opencv2/opencv.hpp>
00035 #endif
00036
00037 namespace csh_img {
00038
00048     enum class En_ImageFormat : uint32_t {
00049         // 8-bit container group (100+)
00050         Bayer8 = 100,
00051         Gray8,
00052
00053         // 16-bit container group (200+)
00054         Bayer10 = 200,
00055         Bayer12,
00056         Bayer14,
00057         Bayer16,
00058         Gray10,
00059         Gray12,
```



```

00060         Gray14,
00061         Gray16,
00062         YUV422,
00063         RGB565,
00064
00065         // 24-bit container group (300+)
00066         YUYV444 = 300,
00067         RGB888,
00068         BGR888,
00069     };
00070
00071     enum class En_ImagePattern : uint32_t {
00072         // Bayer
00073         RGGB = 0, GRBG, BGGR, GBRG,
00074         // YUV422 (packed)
00075         YUYV = 10, UYVY, YVYU, VYUY,
00076         // RGB/BGR (24-bit)
00077         RGB = 20, BGR,
00078     };
00079
00080     enum class En_ImageMemoryAlign : uint32_t {
00081         Packed = 0,
00082         // planar examples
00083         YYYYUUUUUVVVV = 10, YYYYVVVVUUUU, UUUUVVVVYYYY, VVVUUUUYYYY,
00084         // planar RGB
00085         RRRRGGGGBBBB = 20, BBBBGGGGRRRR,
00086         // semi-planar examples
00087         YYYYUVUV = 30, YYYYVUVU,
00088     };
00089
00090     enum class CopyMode : uint32_t { MetaOnly = 0, Shallow, Deep };
00091
00092     class CSH_IMAGE_API CSH_Image {
00093     public:
00094         using byte = uint8_t;
00095
00096         CSH_Image();
00097
00098         CSH_Image(uint32_t width, uint32_t height, En_ImageFormat format,
00099             bool alloc_mem = true, uint32_t image_count = 1);
00100
00101         CSH_Image(const CSH_Image&) = default;
00102         CSH_Image(CSH_Image&&) noexcept = default;
00103         CSH_Image& operator=(const CSH_Image&) = default;
00104         CSH_Image& operator=(CSH_Image&&) noexcept = default;
00105         ~CSH_Image() = default;
00106
00107         // -----
00108         // Copy operations
00109         // -----
00110
00111         void copy(const CSH_Image& src, CopyMode mode);
00112
00113         void copyBufferPointer(const CSH_Image& src);
00114
00115         void copyBufferPointer(byte* pFrame);
00116
00117         // -----
00118         // Persistence (TLV)
00119         // -----
00120
00121         void saveImage(const std::filesystem::path& filepath) const;
00122
00123         void loadImage(const std::filesystem::path& filepath);
00124
00125         inline void saveImage(const std::string& filepath) const {
00126             saveImage(std::filesystem::path(filepath)); }
00127         inline void loadImage(const std::string& filepath) {
00128             loadImage(std::filesystem::path(filepath)); }
00129         inline void saveImage(const char* filepath) const {
00130             saveImage(std::filesystem::path(filepath)); }
00131         inline void loadImage(const char* filepath) { loadImage(std::filesystem::path(filepath)); }
00132 #if defined(_WIN32) || defined(_WIN64)
00133         inline void saveImage(const std::wstring& filepath) const {
00134             saveImage(std::filesystem::path(filepath)); }
00135         inline void loadImage(const std::wstring& filepath) {
00136             loadImage(std::filesystem::path(filepath)); }
00137         inline void saveImage(const wchar_t* filepath) const {
00138             saveImage(std::filesystem::path(filepath)); }
00139         inline void loadImage(const wchar_t* filepath) { loadImage(std::filesystem::path(filepath)); }
00140 #endif
00141
00142         // -----
00143         // Accessors / view
00144         // -----
00145
00146         inline uint32_t getWidth() const { return width; }

```

```

00234     inline uint32_t getHeight() const { return height; }
00236     inline bool isEnabled() const { return bEnable; }
00238     inline uint32_t getCameraId() const { return camera_id; }
00240     inline En_ImageFormat getFormat() const { return format; }
00242     inline uint32_t getMemoryBit() const { return memory_bit; }
00244     inline uint32_t getOriginalBit() const { return original_bit; }
00246     inline En_ImagePattern getPattern() const { return pattern; }
00248     inline En_ImageMemoryAlign getMemoryAlign() const { return memory_align; }
00250     inline std::size_t getBufferSize() const { return buffer_size; }
00252     inline uint32_t getImageCount() const { return image_count; }
00254     inline uint32_t getSelectedImage() const { return sel_image; }
00255
00260     inline byte* data() { return buffer ? (buffer.get() + buffer_offset) : nullptr; }
00261
00266     inline const byte* data() const { return buffer ? (buffer.get() + buffer_offset) : nullptr; }
00267
00274     byte* getImagePtr(uint32_t n);
00275
00279     const byte* getImagePtr(uint32_t n) const;
00280
00287     void setSelectedImage(uint32_t idx);
00288
00293     inline std::size_t totalBytes() const { return buffer_size *
static_cast<std::size_t>(image_count); }
00294
00301     void recomputeBufferSize();
00302
00310     void allocateBuffer();
00311
00312 #ifdef CSH_IMAGE_WITH_OPENCV
00313     // ==== OpenCV zero-copy view (CSH -> cv::Mat) ====
00314
00322     cv::Mat toCvMat(bool deep_copy = false) const;
00323
00328     bool isShareableToCv() const;
00329
00330     // ==== OpenCV -> CSH (instance method) ====
00331
00348     int fromCvMat(const cv::Mat& mat,
00349                  CopyMode mode = CopyMode::Deep,
00350                  En_ImageFormat fmt = static_cast<En_ImageFormat>(0), // auto if 0
00351                  En_ImagePattern pat = static_cast<En_ImagePattern>(0), // auto if 0 or fmt-provided
00352                  En_ImageMemoryAlign align = En_ImageMemoryAlign::Packed);
00353 #endif
00354
00355     // -----
00356     // Public metadata (intentionally plain for POD-like access)
00357     // -----
00358
00359     uint32_t width = 0;
00360     uint32_t height = 0;
00361     bool bEnable = false;
00362     uint32_t camera_id = 0;
00363
00364     En_ImageFormat format = En_ImageFormat::Gray8;
00365     uint32_t memory_bit = 8;
00366     uint32_t original_bit = 8;
00367     En_ImagePattern pattern = En_ImagePattern::RGGB;
00368     En_ImageMemoryAlign memory_align = En_ImageMemoryAlign::Packed;
00369     std::size_t buffer_size = 0;
00370     uint32_t image_count = 1;
00371     uint32_t sel_image = 0;
00372
00373     std::shared_ptr<byte[]> buffer;
00374
00375 private:
00381     std::size_t buffer_offset = 0;
00382
00388     std::size_t buffer_capacity_bytes = 0;
00389
00390     // ---- Static helpers (format defaults / math) ----
00391
00395     static uint32_t defaultMemoryBitForFormat(En_ImageFormat fmt);
00396
00400     static En_ImagePattern defaultPatternForFormat(En_ImageFormat fmt);
00401
00405     static En_ImageMemoryAlign defaultAlignForFormat(En_ImageFormat fmt);
00406
00410     static std::size_t bytesPerPixelForFormat(En_ImageFormat fmt);
00411
00419     static void checkedAddOffset(std::size_t total, std::ptrdiff_t delta, std::size_t& outOffset);
00420
00425     std::size_t writableBytesFromView() const {
00426         const std::size_t cap = buffer_capacity_bytes ? buffer_capacity_bytes : totalBytes();
00427         return (cap > buffer_offset) ? (cap - buffer_offset) : 0;
00428     }
00429

```

```

00430         // ---- TLV constants & helpers ----
00431         static constexpr uint32_t kMagic = 0x43485349;
00432         static constexpr uint32_t kVersion = 1;
00433
00434         enum : uint32_t {
00435             F_WIDTH = 1, F_HEIGHT = 2, F_BENABLE = 3, F_CAMERA_ID = 4, F_FORMAT = 5, F_MEMORY_BIT = 6,
00436             F_ORIGINAL_BIT = 7, F_PATTERN = 8, F_MEM_ALIGN = 9, F_BUFFER_SIZE = 10,
00437             F_IMAGE_COUNT = 11, F_SEL_IMAGE = 12, F_BUFFER_OFF = 13, F_BUFFER_BYTES = 100
00438         };
00439
00440         static void write_u32(std::ostream& os, uint32_t v);
00441         static void write_u64(std::ostream& os, uint64_t v);
00442         static uint32_t read_u32(std::istream& is);
00443         static uint64_t read_u64(std::istream& is);
00444         static void write_bytes(std::ostream& os, const void* data, std::size_t sz);
00445         static void read_bytes(std::istream& is, void* data, std::size_t sz);
00446
00447         #ifdef CSH_IMAGE_WITH_OPENCV
00448         static bool inferFormatFromMat(const cv::Mat& mat, En_ImageFormat& outFmt, En_ImagePattern&
00449             outPat);
00450         #endif
00451     };
00452
00453 } // namespace csh_img

```

4.38 utility/CWatchTime/CWatchTime.h File Reference

Lightweight wall-clock stopwatch and timestamp formatting utilities.

```

#include <chrono>
#include <cwchar>

```

Classes

- class [CWatchTime](#)

RAII-less stopwatch using high_resolution_clock with simple elapsed queries.

4.38.1 Detailed Description

Lightweight wall-clock stopwatch and timestamp formatting utilities.

Provides simple start/stop timing with queries in seconds, milliseconds, and microseconds, plus convenience functions to get the current local time as a formatted wide/narrow C string.

Note

This class is **not thread-safe**. The narrow/wide time-string functions return pointers to internal static/stack buffers; treat the returned pointers as transient and copy the string if you need to retain it.

Definition in file [CWatchTime.h](#).

4.39 CWatchTime.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <chrono>
00003 #include <cwchar>
00004
00017
00018 #if defined(_WIN32)
00019 #if defined(WATCHTIME_EXPORTS)
00020 #define WATCHTIME_API __declspec(dllexport)
00021 #else
00022 #define WATCHTIME_API __declspec(dllimport)
00023 #endif
00024 #else
00025 #define WATCHTIME_API __attribute__((visibility("default")))
00026 #endif
00027
00046 class WATCHTIME_API CWatchTime {
00047 public:
00049     using Clock = std::chrono::high_resolution_clock;
00051     using TimePoint = Clock::time_point;
00052
00058     CWatchTime();
00059
00066     void start();
00067
00074     void stop();
00075
00083     double GetSecond();
00084
00092     double GetMilliSecond();
00093
00101     double GetMicroSecond();
00102
00110     const wchar_t* getString() const;
00111
00120     const wchar_t* GetCurrentTimeString();
00121
00130     const char* GetCurrentTimeStringA();
00131
00132 private:
00134     TimePoint startTime_{};
00136     TimePoint endTime_{};
00138     bool running_{ false };
00139
00141     mutable wchar_t buffer_[32];
00142 };
```

4.40 utility/SH_Log/CSH_Log.h File Reference

Simple thread-safe logging class with printf-style and complete message logging.

```
#include <string>
#include <mutex>
#include <atomic>
#include <cstdarg>
#include <filesystem>
```

Macros

- #define [CSH_FUNC_SIG](#) __PRETTY_FUNCTION__
Compiler-specific macro expanding to the current function signature.
- #define [LOG_WRITE](#)(level, fmt, ...)
Convenience macro for formatted logging with source context.
- #define [LOG_WRITE_MSG](#)(level, wmsg)
Convenience macro for logging a preformatted wide string with source context.

Enumerations

- enum class [cshlog::LogLevel](#) : int {
 [Fatal](#) = 0 , [Error](#) , [Warn](#) , [Info](#) ,
 [Debug](#) , [Trace](#) , [Count](#) }

Log severity levels from most severe to most verbose.

Functions

- class [cshlog::__attribute__](#) ((visibility("default"))) [CSH_Log](#) final

4.40.1 Detailed Description

Simple thread-safe logging class with printf-style and complete message logging.

This header defines the [CSH_Log](#) class, which provides a singleton logger with configurable log levels, file output, and thread safety.

Definition in file [CSH_Log.h](#).

4.40.2 Macro Definition Documentation

4.40.2.1 CSH_FUNC_SIG

```
#define CSH_FUNC_SIG __PRETTY_FUNCTION__
```

Compiler-specific macro expanding to the current function signature.

- MSVC: `__FUNCSIG__`
- GCC/Clang: `__PRETTY_FUNCTION__`

Definition at line [312](#) of file [CSH_Log.h](#).

4.40.2.2 LOG_WRITE

```
#define LOG_WRITE(  
    level,  
    fmt,  
    ...)
```

Value:

```
::cshlog::CSH_Log::Instance().writeLog((level), __FILE__, __LINE__, CSH\_FUNC\_SIG, (fmt), ##__VA_ARGS__)
```

Convenience macro for formatted logging with source context.

Expands to a call to `CSH_Log::writeLog(LogLevel,const char*,int,const char*,const wchar_t*,...)` with `__FILE__`, `__LINE__`, and [CSH_FUNC_SIG](#) automatically supplied.

Parameters

<i>level</i>	A LogLevel value (e.g., LogLevel::Info).
<i>fmt</i>	Wide printf-style format string (UTF-16/UTF-32 depending on platform).
...	Variadic arguments for <i>fmt</i> .

```
LOG_WRITE(cshlog::LogLevel::Info, L"started: pid=%d", pid);
```

Definition at line 330 of file [CSH_Log.h](#).

```
00330 #define LOG_WRITE(level, fmt, ...) \
00331     ::cshlog::CSH_Log::Instance().writeLog((level), __FILE__, __LINE__, CSH_FUNC_SIG, (fmt), \
    ##__VA_ARGS__)
```

4.40.2.3 LOG_WRITE_MSG

```
#define LOG_WRITE_MSG(
    level,
    wmsg)
```

Value:

```
::cshlog::CSH_Log::Instance().writeLog((level), __FILE__, __LINE__, CSH_FUNC_SIG, (wmsg))
```

Convenience macro for logging a preformatted wide string with source context.

Parameters

<i>level</i>	A LogLevel value.
<i>wmsg</i>	Wide string message (no formatting).

```
LOG_WRITE_MSG(cshlog::LogLevel::Error, L"failed to open configuration file");
```

Definition at line 344 of file [CSH_Log.h](#).

```
00344 #define LOG_WRITE_MSG(level, wmsg) \
00345     ::cshlog::CSH_Log::Instance().writeLog((level), __FILE__, __LINE__, CSH_FUNC_SIG, (wmsg))
```

4.40.3 Enumeration Type Documentation

4.40.3.1 LogLevel

```
enum class cshlog::LogLevel : int [strong]
```

Log severity levels from most severe to most verbose.

Lower numeric values indicate higher severity. The value LogLevel::Count is not a real severity; it is a sentinel that indicates the number of levels.

Enumerator

Fatal	Fatal error that prevents the program from continuing.
Error	Recoverable error: operation failed but process continues.
Warn	Suspicious or unexpected condition that may require attention.

Info	High-level informational message about normal operation.
Debug	Detailed messages useful when debugging.
Trace	Very fine-grained messages for deep tracing.
Count	Sentinel; number of log levels (not to be used as a level).

Definition at line 38 of file [CSH_Log.h](#).

```
00038                                     : int {
00039     Fatal = 0,
00040     Error,
00041     Warn,
00042     Info,
00043     Debug,
00044     Trace,
00045     Count
00046 };
```

4.40.4 Function Documentation

4.40.4.1 `__attribute__()`

```
class cshlog::__attribute__ (
    (visibility("default")) ) [final]
```

Returns the singleton instance of the logger.

Returns

Reference to the global logger instance.

Note

The instance is initialized on first use in a thread-safe way.

Initializes the global logger configuration.

Call this once at program start (optional; the logger has defaults). This sets the output directory, whether to persist logs, the minimum severity level, and the alignment width for the function/line field.

Parameters

<i>directory</i>	Target directory to store the log file. If empty, " ." is used.
<i>saveLog</i>	If <code>true</code> , log lines are appended to a file. If <code>false</code> , logging is disabled.
<i>level</i>	Minimum severity to write. Messages below this level are ignored.
<i>funcFieldWidth</i>	Width used to right-pad the func : line field in each log line.

Note

This function is safe to call multiple times; the latest call overwrites previous settings.

Enables or disables on-disk logging.

Parameters

<i>v</i>	Set <code>true</code> to write to the log file; <code>false</code> to disable writes.
----------	---------------------------------------------------------------------------------------

Note

This flag is read atomically by writers.

Returns whether on-disk logging is enabled.

Returns

`true` if logging to file is enabled; otherwise `false`.

Sets the minimum log level.

Parameters

<i>lv</i>	Messages with a severity numerically greater than <code>lv</code> are ignored.
-----------	--------------------------------------------------------------------------------

Note

Stored atomically and read by writers.

Returns the current minimum log level.

Returns

The current `LogLevel`.

Changes the directory where the log file will be written.

If the directory does not exist, it is created (best effort). Future writes will continue using the same timestamp-based file name.

Parameters

<i>dir</i>	Directory path. If empty, <code>" . "</code> is used.
------------	-------------------------------------------------------

Note

Thread-safe: the directory update is protected by a mutex.

Returns the current log directory.

Returns

The directory path currently configured for the log file.

Returns the base log file name (without extension).

Returns

Timestamp string in the form `YYYYMMDD_HHMMSS`.

Writes a formatted log line using a wide printf-style format.

The final message is formatted with `std::vswprintf` and then written as a UTF-8 line. If `fmt` is `nullptr`, an empty message is written. Messages are dropped when logging is disabled or the level is below the configured threshold.

Parameters

<i>level</i>	Severity level for this message.
<i>file</i>	Source file (typically <code>__FILE__</code>). Used for context only; not persisted by default.
<i>line</i>	Source line number (typically <code>__LINE__</code>).
<i>funcsig</i>	Function signature (e.g., <code>__FUNCSIG__</code> or <code>__PRETTY_FUNCTION__</code>), used to derive a concise function name.
<i>fmt</i>	Wide-character printf-style format string (UTF-16/UTF-32 depending on platform).
<i>...</i>	Variadic arguments for the format string.

Note

Thread-safe. The function/line field is padded to funcFieldWidth.

Warning

Avoid passing user-controlled strings directly as format strings.

See also

[LOG_WRITE](#) convenience macro.

Writes a complete message (no formatting) to the log.

Parameters

<i>level</i>	Severity level for this message.
<i>file</i>	Source file (typically <code>__FILE__</code>). Used for context only; not persisted by default.
<i>line</i>	Source line number (typically <code>__LINE__</code>).
<i>funcsig</i>	Function signature (e.g., <code>__FUNCSIG__</code> or <code>__PRETTY_FUNCTION__</code>), used to derive a concise function name.
<i>msg</i>	Fully formatted wide-string message (UTF-16/UTF-32 depending on platform).

Note

Thread-safe. The message is encoded as UTF-8 and appended with a newline.

See also

[LOG_WRITE_MSG](#) convenience macro.

< Non-copyable (singleton).

< Non-copyable (singleton).

< Non-movable (singleton).

< Non-movable (singleton).

Constructs the logger with default configuration.

Use Init to supply custom settings. Construction is internal-only.

Destructor (flushes/closing are handled by ofstream RAI in write paths).

Internal initializer invoked by Init under lock.

Parameters

<i>directory</i>	Log directory.
<i>saveLog</i>	Enable file logging.
<i>level</i>	Minimum log level.
<i>funcFieldWidth</i>	Field width for the func : line block.

Builds one complete log line: timestamp + function/line + level + message.

Parameters

<i>lv</i>	Log level.
<i>funcDisplay</i>	A concise function identifier derived from <i>funcsig</i> .
<i>line</i>	Source line number.
<i>message</i>	Final message text.

Returns

Wide string representing a single log line.

Converts a log level to its wide-string name.

Parameters

<i>lv</i>	Log level.
-----------	------------

Returns

Wide-string representation (e.g., L"Info").

Extracts a concise function name from a compiler-specific signature.

Parameters

<i>funcsig</i>	Function signature (e.g., <code>__FUNCSIG__ / __PRETTY_FUNCTION__</code>).
----------------	-----------------------------------------------------------------------------

Returns

Best-effort shortened function name as wide string.

Converts UTF-8 to wide string (UTF-16 on Windows, UTF-32 on Unix-like platforms).

Parameters

<i>narrow</i>	UTF-8 encoded C-string (may be <code>nullptr</code>).
---------------	--------------------------------------------------------

Returns

Wide string (empty on failure or `nullptr`).

Converts wide string to UTF-8.

Parameters

<i>wide</i>	Wide string to convert.
-------------	-------------------------

Returns

UTF-8 encoded `std::string` (empty on failure).

Composes the full path of the current log file including extension.

Returns

`<dir>/<YYYYMMDD_HHMMSS>.log`

Ensures a directory exists, creating it if necessary (best effort).

Parameters

<i>dir</i>	Directory path to create.
------------	---------------------------

Converts [CWatchTime](#) ASCII timestamp to a file-stamp (seconds precision).

Parameters

<i>aTime</i>	ASCII time "YYYY-MM-DD HH:MM:SS.mmm". If <code>nullptr</code> , returns "00000000_000000".
--------------	--------------------------------------------------------------------------------------------

Returns

Wide string "YYYYMMDD_HHMMSS".

Converts [CWatchTime](#) ASCII timestamp to a printable stamp with milliseconds.

Parameters

<i>aTime</i>	ASCII time "YYYY-MM-DD HH:MM:SS.mmm". If <code>nullptr</code> , uses "00000000_000000.000".
--------------	---------------------------------------------------------------------------------------------

Returns

Wide string "YYYYMMDD_HHMMSS.mmm".

< If false, all writes are no-ops.

< Minimum severity to write.

< Output directory for the log file.

< Base name (without extension), e.g., YYYYMMDD_HHMMSS.

< Padding width for the func : line field.

< Serializes initialization and file writes.

Definition at line 1 of file CSH_Log.h.

```
00063         {
00064     public:
00072         static CSH_Log& Instance();
00073
00089         static void Init(const std::wstring& directory,
00090             bool saveLog = true,
00091             LogLevel level = LogLevel::Info,
00092             std::size_t funcFieldWidth = 60);
00093
00094         // -----
00095         // Configuration accessors
00096         // -----
00097
00103         void setSaveLog(bool v) noexcept { bSaveLog.store(v, std::memory_order_relaxed); }
00104
00109         bool getSaveLog() const noexcept { return bSaveLog.load(std::memory_order_relaxed); }
00110
00116         void setLogLevel(LogLevel lv) noexcept { logLevel.store(static_cast<int>(lv),
std::memory_order_relaxed); }
00117
00122         LogLevel getLogLevel() const noexcept { return
static_cast<LogLevel>(logLevel.load(std::memory_order_relaxed)); }
00123
00133         void setLogDirectory(const std::wstring& dir);
00134
00139         std::wstring getLogDirectory() const { return strFilePath; }
00140
00145         std::wstring getFileName() const { return strFileName; }
00146
00147         // -----
00148         // Logging APIs
00149         // -----
00150
00170         void writeLog(LogLevel level,
00171             const char* file,
00172             int line,
00173             const char* funcsig,
00174             const wchar_t* fmt, ...) noexcept;
00175
00188         void writeLog(LogLevel level,
00189             const char* file,
00190             int line,
00191             const char* funcsig,
00192             const std::wstring& msg) noexcept;
00193
00194         CSH_Log(const CSH_Log&) = delete;
00195         CSH_Log& operator=(const CSH_Log&) = delete;
00196         CSH_Log(CSH_Log&&) = delete;
00197         CSH_Log& operator=(CSH_Log&&) = delete;
00198
00199     private:
00205         CSH_Log();
00206
00210         ~CSH_Log();
00211
00219         void initialize(const std::wstring& directory,
00220             bool saveLog,
00221             LogLevel level,
00222             std::size_t funcFieldWidth);
00223
00232         std::wstring buildLogLine(LogLevel lv,
00233             const std::wstring& funcDisplay,
00234             int line,
00235             const std::wstring& message) const;
00236
00242         std::wstring levelToWString(LogLevel lv) const;
00243
00249         std::wstring sanitizeFunctionFromSignature(const char* funcsig) const;
00250
00256         static std::wstring toWString(const char* narrow);
00257
00263         static std::string toUTF8(const std::wstring& wide);
00264
00269         std::filesystem::path composeFullPathWithExt() const;
00270
00275         static void ensureDirectory(const std::filesystem::path& dir);
00276
00282         static std::wstring ToFileNameStampFromCWatchA(const char* aTime);
00283
00289         static std::wstring ToMilliStampFromCWatchA(const char* aTime);
00290
00291     private:
00292         // --- Configuration/state ---
```

```

00293         std::atomic<bool> bSaveLog{ true };
00294         std::atomic<int> logLevel{ static_cast<int>(LogLevel::Info) };
00295         std::wstring strFilePath;
00296         std::wstring strFileName;
00297         std::size_t funcFieldWidth{ 60 };
00298
00299         mutable std::mutex mtx;
00300     };

```

4.41 CSH_Log.h

[Go to the documentation of this file.](#)

```

00001
00009
00010 #pragma once
00011 // C++17, Unicode (wchar_t) interface, DLL export macro
00012
00013 #include <string>
00014 #include <mutex>
00015 #include <atomic>
00016 #include <cstdint>
00017 #include <filesystem>
00018
00019 #if defined(_WIN32)
00020 #if defined(CSH_LOG_EXPORT)
00021 #define CSHLOG_API __declspec(dllexport)
00022 #else
00023 #define CSHLOG_API __declspec(dllimport)
00024 #endif
00025 #else
00026 #define CSHLOG_API __attribute__((visibility("default")))
00027 #endif
00028
00029 namespace cshlog {
00030
00038     enum class LogLevel : int {
00039         Fatal = 0,
00040         Error,
00041         Warn,
00042         Info,
00043         Debug,
00044         Trace,
00045         Count
00046     };
00047
00063     class CSHLOG_API CSH_Log final {
00064     public:
00072         static CSH_Log& Instance();
00073
00089         static void Init(const std::wstring& directory,
00090             bool saveLog = true,
00091             LogLevel level = LogLevel::Info,
00092             std::size_t funcFieldWidth = 60);
00093
00094         // -----
00095         // Configuration accessors
00096         // -----
00097
00103         void setSaveLog(bool v) noexcept { bSaveLog.store(v, std::memory_order_relaxed); }
00104
00109         bool getSaveLog() const noexcept { return bSaveLog.load(std::memory_order_relaxed); }
00110
00116         void setLogLevel(LogLevel lv) noexcept { logLevel.store(static_cast<int>(lv),
std::memory_order_relaxed); }
00117
00122         LogLevel getLogLevel() const noexcept { return
static_cast<LogLevel>(logLevel.load(std::memory_order_relaxed)); }
00123
00133         void setLogDirectory(const std::wstring& dir);
00134
00139         std::wstring getLogDirectory() const { return strFilePath; }
00140
00145         std::wstring getFileName() const { return strFileName; }
00146
00147         // -----
00148         // Logging APIs
00149         // -----
00150
00170         void writeLog(LogLevel level,
00171             const char* file,
00172             int line,

```

```

00173         const char* funcsig,
00174         const wchar_t* fmt, ...) noexcept;
00175
00176
00177 void writeLog(LogLevel level,
00178             const char* file,
00179             int line,
00180             const char* funcsig,
00181             const std::wstring& msg) noexcept;
00182
00183
00184 CSH_Log(const CSH_Log&) = delete;
00185 CSH_Log& operator=(const CSH_Log&) = delete;
00186 CSH_Log(CSH_Log&&) = delete;
00187 CSH_Log& operator=(CSH_Log&&) = delete;
00188
00189 private:
00190     CSH_Log();
00191
00192     ~CSH_Log();
00193
00194 void initialize(const std::wstring& directory,
00195             bool saveLog,
00196             LogLevel level,
00197             std::size_t funcFieldWidth);
00198
00199 std::wstring buildLogLine(LogLevel lv,
00200             const std::wstring& funcDisplay,
00201             int line,
00202             const std::wstring& message) const;
00203
00204 std::wstring levelToWString(LogLevel lv) const;
00205
00206 std::wstring sanitizeFunctionFromSignature(const char* funcsig) const;
00207
00208 static std::wstring toWString(const char* narrow);
00209
00210 static std::string toUTF8(const std::wstring& wide);
00211
00212 std::filesystem::path composeFullPathWithExt() const;
00213
00214 static void ensureDirectory(const std::filesystem::path& dir);
00215
00216 static std::wstring ToFileNameStampFromCWatchA(const char* aTime);
00217
00218 static std::wstring ToMilliStampFromCWatchA(const char* aTime);
00219
00220 private:
00221     // --- Configuration/state ---
00222     std::atomic<bool> bSaveLog{ true };
00223     std::atomic<int> logLevel{ static_cast<int>(LogLevel::Info) };
00224     std::wstring strFilePath;
00225     std::wstring strFileName;
00226     std::size_t funcFieldWidth{ 60 };
00227
00228     mutable std::mutex mtx;
00229 };
00230
00231 #if defined(_MSC_VER)
00232 #define CSH_FUNC_SIG __FUNCSIG__
00233 #else
00234 #define CSH_FUNC_SIG __PRETTY_FUNCTION__
00235 #endif
00236
00237 #define LOG_WRITE(level, fmt, ...) \
00238     ::cshlog::CSH_Log::Instance().writeLog((level), __FILE__, __LINE__, CSH_FUNC_SIG, (fmt), \
00239     ##__VA_ARGS__)
00240
00241 #define LOG_WRITE_MSG(level, wmsg) \
00242     ::cshlog::CSH_Log::Instance().writeLog((level), __FILE__, __LINE__, CSH_FUNC_SIG, (wmsg))
00243
00244 } // namespace cshlog

```

Index

- `__attribute__`
 - `ClImageDisplay.h`, [42](#)
 - `ClImageDisplayC.h`, [50](#)
 - `ClImageProcessMng.h`, [59](#)
 - `CSH_Log.h`, [95](#)
 - `lpmClamp.h`, [76](#)
- `AlgEntry`, [5](#)
- `allocateBuffer`
 - `csh_img::CSH_Image`, [19](#)
- `AMX_Tile`
 - `ClpmCpuEnv.h`, [64](#)
- `AVX2`
 - `ClpmCpuEnv.h`, [64](#)
- `AVX512BW`
 - `ClpmCpuEnv.h`, [64](#)
- `AVX512F`
 - `ClpmCpuEnv.h`, [64](#)
- `Bayer10`
 - `CSH_Image.h`, [87](#)
- `Bayer12`
 - `CSH_Image.h`, [87](#)
- `Bayer14`
 - `CSH_Image.h`, [87](#)
- `Bayer16`
 - `ClImageDisplay.h`, [41](#)
 - `CSH_Image.h`, [87](#)
- `Bayer8`
 - `CSH_Image.h`, [87](#)
- `bestSimdFor`
 - `ipm::ClpmCpuEnv`, [12](#)
- `BGR888`
 - `ClImageDisplay.h`, [41](#)
 - `CSH_Image.h`, [87](#)
- `CConverter`, [5](#)
- `CCpuParaConverter`, [6](#)
- `CCpuSerialConverter`, [6](#)
- `CFrameGrabber/CFrameGrabber.h`, [37](#)
- `CFrameGrabber/CGrabberConfig.h`, [38](#)
- `CFrameGrabber/IFrameGrabImpl.h`, [39](#)
- `cimage::ClImageDisplayerCPP`, [7](#)
 - `setImage`, [9](#)
 - `setImageRaw`, [10](#)
 - `wheelScroll`, [10](#)
- `cimage::Mat4`, [32](#)
- `cimage::Quat`, [33](#)
- `cimage::UploadDescriptor`, [33](#)
- `cimage::Vec2`, [35](#)
- `cimage::Vec3`, [35](#)
- `ClImageDisplay`, [7](#)
- `ClImageDisplay.h`
 - `__attribute__`, [42](#)
 - `Bayer16`, [41](#)
 - `BGR888`, [41](#)
 - `Gray16`, [41](#)
 - `Gray8`, [41](#)
 - `PixelLayout`, [41](#)
 - `RGB565`, [41](#)
 - `RGB888`, [41](#)
 - `Unknown`, [41](#)
 - `YUV422Packed`, [41](#)
- `ClImageDisplay/ClImageDisplay.h`, [40](#), [46](#)
- `ClImageDisplay/ClImageDisplayC.h`, [49](#), [52](#)
- `ClImageDisplay/ClImageDisplayerCPP.h`, [54](#), [55](#)
- `ClImageDisplayC.h`
 - `__attribute__`, [50](#)
- `ClImageProcessMng.h`
 - `__attribute__`, [59](#)
 - `DisplayCallback`, [58](#)
- `ClImageUploadDesc`, [11](#)
- `ClpmCpuEnv.h`
 - `AMX_Tile`, [64](#)
 - `AVX2`, [64](#)
 - `AVX512BW`, [64](#)
 - `AVX512F`, [64](#)
 - `En_OpProfile`, [63](#)
 - `En_SimdKind`, [64](#)
 - `Float32_64`, [64](#)
 - `Integer8_16`, [64](#)
 - `Matrix2D`, [64](#)
 - `NEON`, [64](#)
 - `SVE`, [64](#)
 - `SVE2`, [64](#)
- `ClpmFuncTable.h`
 - `ClpmFuncTable_RegisterDummyForDev`, [68](#)
- `ClpmFuncTable_RegisterDummyForDev`
 - `ClpmFuncTable.h`, [68](#)
- `Converter`
 - `lpmTypes.h`, [82](#)
- `copy`
 - `csh_img::CSH_Image`, [20](#)
- `copyBufferPointer`
 - `csh_img::CSH_Image`, [20](#), [21](#)
- `CopyMode`
 - `CSH_Image.h`, [86](#)
- `Count`
 - `CSH_Log.h`, [95](#)

- CPU_Parallel
 - lpmTypes.h, [83](#)
- CPU_Serial
 - lpmTypes.h, [83](#)
- CSH_FUNC_SIG
 - CSH_Log.h, [93](#)
- CSH_Image
 - csh_img::CSH_Image, [19](#)
- CSH_Image.h
 - Bayer10, [87](#)
 - Bayer12, [87](#)
 - Bayer14, [87](#)
 - Bayer16, [87](#)
 - Bayer8, [87](#)
 - BGR888, [87](#)
 - CopyMode, [86](#)
 - En_ImageFormat, [86](#)
 - En_ImageMemoryAlign, [87](#)
 - En_ImagePattern, [88](#)
 - Gray10, [87](#)
 - Gray12, [87](#)
 - Gray14, [87](#)
 - Gray16, [87](#)
 - Gray8, [87](#)
 - Packed, [87](#)
 - RGB565, [87](#)
 - RGB888, [87](#)
 - YUV422, [87](#)
 - YUYV444, [87](#)
- csh_img::CSH_Image, [16](#)
 - allocateBuffer, [19](#)
 - copy, [20](#)
 - copyBufferPointer, [20](#), [21](#)
 - CSH_Image, [19](#)
 - data, [21](#)
 - getBufferSize, [21](#)
 - getCameraId, [22](#)
 - getFormat, [22](#)
 - getHeight, [22](#)
 - getImageCount, [22](#)
 - getImagePtr, [23](#)
 - getMemoryAlign, [24](#)
 - getMemoryBit, [24](#)
 - getOriginalBit, [24](#)
 - getPattern, [24](#)
 - getSelectedImage, [25](#)
 - getWidth, [25](#)
 - isEnabled, [25](#)
 - loadImage, [25](#)
 - recomputeBufferSize, [26](#)
 - saveImage, [26](#)
 - setSelectedImage, [26](#)
 - totalBytes, [27](#)
- CSH_Log, [27](#)
- CSH_Log.h
 - __attribute__, [95](#)
 - Count, [95](#)
 - CSH_FUNC_SIG, [93](#)
 - Debug, [95](#)
 - Error, [94](#)
 - Fatal, [94](#)
 - Info, [95](#)
 - LOG_WRITE, [93](#)
 - LOG_WRITE_MSG, [94](#)
 - LogLevel, [94](#)
 - Trace, [95](#)
 - Warn, [94](#)
- CWatchTime, [28](#)
 - CWatchTime, [29](#)
 - GetCurrentTimeString, [29](#)
 - GetCurrentTimeStringA, [29](#)
 - GetMicroSecond, [29](#)
 - GetMilliSecond, [29](#)
 - GetSecond, [30](#)
 - getString, [30](#)
 - start, [30](#)
 - stop, [30](#)
- data
 - csh_img::CSH_Image, [21](#)
- Debug
 - CSH_Log.h, [95](#)
- DisplayCallback
 - ClmageProcessMng.h, [58](#)
- En_ImageFormat
 - CSH_Image.h, [86](#)
- En_ImageMemoryAlign
 - CSH_Image.h, [87](#)
- En_ImagePattern
 - CSH_Image.h, [88](#)
- En_OpProfile
 - ClpmCpuEnv.h, [63](#)
- En_SimdKind
 - ClpmCpuEnv.h, [64](#)
- EnlpmModule
 - lpmTypes.h, [82](#)
- EnProcessBackend
 - lpmTypes.h, [82](#)
- Err_AlgNotFound
 - lpmTypes.h, [83](#)
- Err_Internal
 - lpmTypes.h, [83](#)
- Err_InvalidBackend
 - lpmTypes.h, [83](#)
- Err_InvalidFormat
 - lpmTypes.h, [83](#)
- Err_InvalidModule
 - lpmTypes.h, [83](#)
- Err_InvalidSize
 - lpmTypes.h, [83](#)
- Err_NullFunction
 - lpmTypes.h, [83](#)
- Err_NullImage
 - lpmTypes.h, [83](#)
- Error
 - CSH_Log.h, [94](#)

Fatal
 CSH_Log.h, 94
Float32_64
 ClpmCpuEnv.h, 64
FuncInfo, 31

getAlgorithmList
 ipmcommon::ClpmFuncTable, 14
getBufferSize
 csh_img::CSH_Image, 21
getCamerald
 csh_img::CSH_Image, 22
GetCurrentTimeString
 CWatchTime, 29
GetCurrentTimeStringA
 CWatchTime, 29
getFormat
 csh_img::CSH_Image, 22
getHeight
 csh_img::CSH_Image, 22
getImageCount
 csh_img::CSH_Image, 22
getImagePtr
 csh_img::CSH_Image, 23
getMemoryAlign
 csh_img::CSH_Image, 24
getMemoryBit
 csh_img::CSH_Image, 24
GetMicroSecond
 CWatchTime, 29
GetMilliSecond
 CWatchTime, 29
getOriginalBit
 csh_img::CSH_Image, 24
getPattern
 csh_img::CSH_Image, 24
GetSecond
 CWatchTime, 30
getSelectedImage
 csh_img::CSH_Image, 25
getString
 CWatchTime, 30
getWidth
 csh_img::CSH_Image, 25
GPU_CUDA
 lpmTypes.h, 83
GPU_GL_Compute
 lpmTypes.h, 83
GPU_OpenCL
 lpmTypes.h, 83
Gray10
 CSH_Image.h, 87
Gray12
 CSH_Image.h, 87
Gray14
 CSH_Image.h, 87
Gray16
 CImageDisplay.h, 41
 CSH_Image.h, 87

Gray8
 CImageDisplay.h, 41
 CSH_Image.h, 87

ImageProcessorManager/CImageProcessMng.h, 57, 62
ImageProcessorManager/ClpmCpuEnv.h, 63, 64
ImageProcessorManager/ClpmEnv.h, 66
ImageProcessorManager/ClpmFuncTable.h, 67, 69
ImageProcessorManager/ClpmGpuEnv.h, 69, 70
ImageProcessorManager/ClpmUserCustomLoader.h,
 71, 72
ImageProcessorManager/Converter/CConverter.h, 73
ImageProcessorManager/Converter/CCpuParaConverter.h,
 74
ImageProcessorManager/Converter/CCpuSerialConverter.h,
 74
ImageProcessorManager/Converter/CGpuCIConverter.h,
 75
ImageProcessorManager/Converter/CGpuCudaConverter.h,
 75
ImageProcessorManager/Converter/CGpuGILComputeConverter.h,
 75
ImageProcessorManager/lpmClamp.h, 75, 76
ImageProcessorManager/lpmStringUtils.h, 77, 79
ImageProcessorManager/lpmTypes.h, 80, 84
Info
 CSH_Log.h, 95
Integer8_16
 ClpmCpuEnv.h, 64
ipm::ClpmCpuEnv, 12
 bestSimdFor, 12
ipm::ClpmEnv, 13
ipm::ClpmGpuEnv, 15
 selectByNameSubstring, 16
ipm::GpuInfo, 31
ipm_internal::UserCustomLoader, 34
 loadOnce, 35
lpmClamp.h
 __attribute__, 76
ipmcommon::ClpmFuncTable, 13
 getAlgorithmList, 14
 process, 14
lpmFn
 lpmTypes.h, 81
lpmStatus
 lpmTypes.h, 83
lpmStringUtils.h
 u8_to_w, 78
 w_to_u8, 78
lpmTypes.h
 Converter, 82
 CPU_Parallel, 83
 CPU_Serial, 83
 EnlpmModule, 82
 EnProcessBackend, 82
 Err_AlgorithmNotFound, 83
 Err_Internal, 83
 Err_InvalidBackend, 83
 Err_InvalidFormat, 83

- Err_InvalidModule, [83](#)
- Err_InvalidSize, [83](#)
- Err_NullFunction, [83](#)
- Err_NullImage, [83](#)
- GPU_CUDA, [83](#)
- GPU_GL_Compute, [83](#)
- GPU_OpenCL, [83](#)
- IpmFn, [81](#)
- IpmStatus, [83](#)
- IsDevellopping, [83](#)
- NotAvailable, [83](#)
- OK, [83](#)
- Scaler, [82](#)
- Splitter, [82](#)
- User_Custom, [82](#)
- IsDevellopping
 - IpmTypes.h, [83](#)
- isEnabled
 - csh_img::CSH_Image, [25](#)
- loadImage
 - csh_img::CSH_Image, [25](#)
- loadOnce
 - ipm_internal::UserCustomLoader, [35](#)
- LOG_WRITE
 - CSH_Log.h, [93](#)
- LOG_WRITE_MSG
 - CSH_Log.h, [94](#)
- LogLevel
 - CSH_Log.h, [94](#)
- Matrix2D
 - ClpmCpuEnv.h, [64](#)
- NEON
 - ClpmCpuEnv.h, [64](#)
- NotAvailable
 - IpmTypes.h, [83](#)
- OK
 - IpmTypes.h, [83](#)
- Packed
 - CSH_Image.h, [87](#)
- PixelLayout
 - ClmageDisplay.h, [41](#)
- plugins/ClpmUserCustom/ClpmUserCustom.h, [84](#)
- process
 - ipmcommon::ClpmFuncTable, [14](#)
- recomputeBufferSize
 - csh_img::CSH_Image, [26](#)
- RGB565
 - ClmageDisplay.h, [41](#)
 - CSH_Image.h, [87](#)
- RGB888
 - ClmageDisplay.h, [41](#)
 - CSH_Image.h, [87](#)
- saveImage
 - csh_img::CSH_Image, [26](#)
- Scaler
 - IpmTypes.h, [82](#)
- selectByNameSubstring
 - ipm::ClpmGpuEnv, [16](#)
- setImage
 - cimage::ClmageDisplayerCPP, [9](#)
- setImageRaw
 - cimage::ClmageDisplayerCPP, [10](#)
- setSelectedImage
 - csh_img::CSH_Image, [26](#)
- Splitter
 - IpmTypes.h, [82](#)
- start
 - CWatchTime, [30](#)
- stop
 - CWatchTime, [30](#)
- SVE
 - ClpmCpuEnv.h, [64](#)
- SVE2
 - ClpmCpuEnv.h, [64](#)
- totalBytes
 - csh_img::CSH_Image, [27](#)
- Trace
 - CSH_Log.h, [95](#)
- u8_to_w
 - IpmStringUtils.h, [78](#)
- Unknown
 - ClmageDisplay.h, [41](#)
- User_Custom
 - IpmTypes.h, [82](#)
- utility/CSH_Image/CSH_Image.h, [85](#), [88](#)
- utility/CWatchTime/CWatchTime.h, [91](#), [92](#)
- utility/SH_Log/CSH_Log.h, [92](#), [102](#)
- w_to_u8
 - IpmStringUtils.h, [78](#)
- Warn
 - CSH_Log.h, [94](#)
- wheelScroll
 - cimage::ClmageDisplayerCPP, [10](#)
- YUV422
 - CSH_Image.h, [87](#)
- YUV422Packed
 - ClmageDisplay.h, [41](#)
- YUYV444
 - CSH_Image.h, [87](#)
