CNStream Developer Guide

Release 2021-01-25 (Version 5.3.0)

Feb 19, 2021

Table of Contents

1 Copyright

2 Datatypes

CNStream data types support both on MLU270 and MLU220.

2.1 Data Source

2.1.1 DataSource

class cnstream::DataSource

`class DataSource` : `public` SourceModule, `public` cnstream::ModuleCreator<DataSource>

Class for handling input data.

2.1.2 DataSourceParam

struct DataSourceParam {

OutputType output_type_ = OUTPUT_CPU;

size_t interval_ = 1;

DecoderType decoder_type_ = DECODER_CPU;

bool reuse_cndec_buf = false;

int device_id_ = -1;

uint32_t input_buf_number_ = 2;

uint32_t output_buf_number_ = 3;

bool apply_stride_align_for_scaler_ = false;

};

`struct cnstream::DataSourceParam`

a structure for private usage

Public Members

OutputType `output_type_` = OUTPUT_CPU
: output data to cpu/mlu

size_t `interval_` = 1
: output image every "interval" frames

DecoderType `decoder_type_` = DECODER_CPU
: decoder type

bool `reuse_cndec_buf` = false
: valid when DECODER_MLU used

int `device_id_` = -1
: mlu device id, -1 :disable mlu

uint32_t `input_buf_number_` = 2
: valid when decoder_type = DECODER_MLU

uint32_t `output_buf_number_` = 3
: valid when decoder_type = DECODER_MLU

bool `apply_stride_align_for_scaler_` = false
: recommended for use on m200 platforms

2.1.3 DataType

enum DataType {

INVALID,

H264,

H265,

};

`enum cnstream::`*`ESMemHandler`*`::DataType`

The enum of data type.

Values:

`enumerator INVALID`
: Invalid data type.

`enumerator H264`
: The data type of H264.

`enumerator H265`
: The data type of H265.

2.1.4 DecoderType

enum DecoderType {

DECODER_CPU,

DECODER_MLU

};

`enum cnstream::DecoderType`

decoder type used in source module.

Values:

`enumerator DECODER_CPU`

`enumerator DECODER_MLU`

2.1.5 ESJpegMemHandler

class cnstream::ESJpegMemHandler

`class ESJpegMemHandler` : `public` SourceHandler

Source handler for Jpeg bitstreams in memory.

2.1.6 ESMemHandler

class cnstream::ESMemHandler

`class ESMemHandler` : `public` SourceHandler

Source handler for H264/H265 bitstreams in memory(with prefix-start-code).

2.1.7 ESPacket

typedef struct {

unsigned char *data = nullptr;

int size = 0;

uint64_t pts = 0;

uint32_t flags = 0;

enum {

FLAG_KEY_FRAME = 0x01,

FLAG_EOS = 0x02,

};

} ESPacket;

`struct cnstream::ESPacket`

The struct of ES data packet.

Public Types

`enum` **[anonymous]**

The flags of frame.

Values:

`enumerator FLAG_KEY_FRAME` = 0x01

flag of key frame.

`enumerator FLAG_EOS` = 0x02

flag of eos frame.

Public Members

unsigned char *`data` = nullptr

the data.

int `size` = 0

the size of the data.

uint64_t `pts` = 0

the pts of the data.

uint32_t `flags` = 0

the flags of the data.

2.1.8 OutputType

enum OutputType {

OUTPUT_CPU,

OUTPUT_MLU

};

`enum cnstream::OutputType`

storage type of output frame data for modules, storage on cpu or mlu.

Values:

`enumerator OUTPUT_CPU`

`enumerator OUTPUT_MLU`

2.1.9 RtspHandler

class cnstream::RtspHandler

`class RtspHandler` : `public` SourceHandler

Source handler for rtsp stream.

2.2 EventBus

2.2.1 Event

struct Event {

EventType type;

std::string stream_id;

std::string message;

std::string module_name;

std::thread::id thread_id;

};

`struct cnstream::Event`

A structure holding the event information.

Public Members

EventType `type`

The event type.

std::string `stream_id`

The stream that posts this event.

std::string `message`

Additional event messages.

std::string `module_name`

The module that posts this event.

std::thread::id `thread_id`

The thread id from which the event is posted.

2.2.2 EventBus

class cnstream::EventBus

`class EventBus : private` NonCopyable

The event bus that transmits events from modules to a pipeline.

2.2.3 EventHandleFlag

enum EventHandleFlag {

EVENT_HANDLE_NULL,

EVENT_HANDLE_INTERCEPTION,

EVENT_HANDLE_SYNCED,

EVENT_HANDLE_STOP

};

`enum cnstream::EventHandleFlag`

Flags to specify the way in which bus watchers handled one event.

Values:

`enumerator EVENT_HANDLE_NULL`

The event is not handled.

`enumerator EVENT_HANDLE_INTERCEPTION`

The bus watcher is informed, and the event is intercepted.

`enumerator EVENT_HANDLE_SYNCED`

The bus watcher is informed, and then other bus watchers are informed.

`enumerator EVENT_HANDLE_STOP`

A poll event is stopped.

2.3 Frame

2.3.1 CNDataFormat

enum CNDataFormat {

CN_INVALID = -1,

CN_PIXEL_FORMAT_YUV420_NV21 = 0,

CN_PIXEL_FORMAT_YUV420_NV12,

CN_PIXEL_FORMAT_BGR24,

CN_PIXEL_FORMAT_RGB24

};

`enum cnstream::CNDataFormat`

An enumerated type that is used to identify the pixel format of the data in CNDataFrame.

Values:

`enumerator CN_INVALID` = -1

This frame is invalid.

`enumerator CN_PIXEL_FORMAT_YUV420_NV21` = 0

This frame is in the YUV420SP(NV21) format.

`enumerator CN_PIXEL_FORMAT_YUV420_NV12`

This frame is in the YUV420sp(NV12) format.

`enumerator CN_PIXEL_FORMAT_BGR24`

This frame is in the BGR24 format.

`enumerator CN_PIXEL_FORMAT_RGB24`

This frame is in the RGB24 format.

`enumerator CN_PIXEL_FORMAT_ARGB32`

This frame is in the ARGB32 format.

`enumerator CN_PIXEL_FORMAT_ABGR32`

This frame is in the ABGR32 format.

`enumerator CN_PIXEL_FORMAT_RGBA32`

This frame is in the RGBA32 format.

`enumerator CN_PIXEL_FORMAT_BGRA32`

This frame is in the BGRA32 format.

2.3.2 CNDataFrame

`class CNDataFrame : public` NonCopyable

The structure holding a data frame and the frame description.

2.3.3 CNFrameFlag

enum CNFrameFlag {

CN_FRAME_FLAG_EOS = 1 << 0,

CN_FRAME_FLAG_INVALID = 1 << 1,

CN_FRAME_FLAG_REMOVED = 2 << 1

};

`enum cnstream::CNFrameFlag`

An enumerated type that specifies the mask of CNDataFrame.

Values:

`enumerator CN_FRAME_FLAG_EOS` = 1 << 0

Identifies the end of data stream.

`enumerator CN_FRAME_FLAG_INVALID` = 1 << 1
Identifies the invalid of frame.

`enumerator CN_FRAME_FLAG_REMOVED` = 2 << 1
Identifies the stream has been removed.

2.3.4 CNFrameInfo

`class CNFrameInfo` : `private` NonCopyable
A structure holding the information of a frame.

2.3.5 CNInferAttr

typedef struct {

int id = -1;

int value = -1;

float score = 0;

} CNInferAttr;

`struct cnstream::CNInferAttr`
A structure holding the classification properties of an object.

Public Members

int `id` = -1
The unique ID of the classification. The value -1 is invalid.

int `value` = -1
The label value of the classification.

float `score` = 0
The label score of the classification.

2.3.6 CNInferBoundingBox

typedef struct {

float x, y, w, h;

} CNInferBoundingBox;

`struct cnstream::CNInferBoundingBox`
A structure holding the bounding box for detection information of an object. Normalized coordinates.

Public Members

float `x`
: The x-axis coordinate in the upper left corner of the bounding box.

float `y`
: The y-axis coordinate in the upper left corner of the bounding box.

float `w`
: The width of the bounding box.

float `h`
: The height of the bounding box.

2.3.7 CNInferData

`struct CNInferData` : `public` NonCopyable

2.3.8 CNInferFeature

typedef std::vector<float> cnstream::CNInferFeature;

`using cnstream::CNInferFeature` = std::vector<float>
: The feature value for one object.

2.3.9 CNInferFeatures

typedef std::vector<std::pair<std::string, CNInferFeature>> cnstream::CNInferFeatures;

`using cnstream::CNInferFeatures` = std::vector<std::pair<std::string, CNInferFeature>>
: All kinds of features for one object.

2.3.10 CNInferObject

typedef struct {

public:

std::string id;

std::string track_id;

float score;

CNInferBoundingBox bbox;

void* user_data_ = nullptr;

private:

std::map<std::string, CNInferAttr> attributes_;

std::map<std::string, std::string> extra_attributes_;

std::vector<CNInferFeature> features_;

std::mutex attribute_mutex_;

std::mutex feature_mutex_;

} CNInferObject;

`struct CNInferObject`

A structure holding the information for an object.

2.3.11 CNInferObjs

`struct CNInferObjs` : `public` NonCopyable

2.3.12 DevContext

typedef struct {

DevType dev_type = INVALID;

int dev_id = 0;

int ddr_channel = 0

} DevContext;

`struct cnstream::DevContext`

Identifies if the CNDataFrame data is allocated by CPU or MLU.

Public Members

`enum` cnstream::DevContext::DevType `dev_type` = INVALID

Device type.

int `dev_id` = 0

Ordinal device ID.

int `ddr_channel` = 0

Ordinal channel ID for MLU. The value should be in the range [0, 4).

2.3.13 DevType

enum DevType {

INVALID = -1,

CPU = 0,

MLU = 1,

MLU_CPU = 2

};

`enum cnstream::`*`DevContext`*`::DevType`

Values:

`enumerator INVALID` = -1

Invalid device type.

`enumerator CPU` = 0

The data is allocated by CPU.

`enumerator MLU` = 1

The data is allocated by MLU.

`enumerator MLU_CPU` = 2

The data is allocated both by MLU and CPU. Used for M220_SOC.

2.3.14 ICNMediaImageMapper

class cnstream::ICNMediaImageMapper

`class ICNMediaImageMapper`

ICNMediaImageMapper is an abstract class, for M220_SOC only.

2.3.15 IDataDeallocator

class cnstream::IDataDeallocator

`class IDataDeallocator`

Dedicated deallocator for the CNDecoder buffer.

2.3.16 InferData

`struct InferData`

A structure holding the information for inference input & outputs(raw).

2.3.17 MemMapType

enum MemMapType {

MEMMAP_INVALID = 0,

MEMMAP_CPU = 1,

MEMMAP_MLU = 2

};

`enum cnstream::MemMapType`

Identifies memory shared type for multi-process.

Values:

`enumerator MEMMAP_INVALID` = 0

Invalid memory shared type.

`enumerator MEMMAP_CPU` = 1
  CPU memory is shared.

`enumerator MEMMAP_MLU` = 2
  MLU memory is shared.

2.3.18 StringPairs

typedef std::vector<std::pair<std::string, std::string>> cnstream::StringPairs;

`using cnstream::StringPairs` = std::vector<std::pair<std::string, std::string>>
  String pairs for extra attributes.

2.4 Inferencer

2.4.1 CNFrameInfoPtr

typedef std::shared_ptr<cnstream::CNFrameInfo> cnstream::CNFrameInfoPtr

`typedef` std::shared_ptr<CNFrameInfo> `cnstream::CNFrameInfoPtr`
  Constructs a pointer to CNFrameInfo.

  Pointer for frame info.

  Pointer for frame information.

2.4.2 Inferencer

class cnstream::Inferencer

`class Inferencer` : `public` cnstream::Module, `public` cnstream::ModuleCreator<Inferencer>
  Inferencer is a module for running offline model inference.

  The input could come from Decoder or other plugins, in MLU memory or CPU memory. Also, if the `preproc_name` parameter is set to `PreprocCpu` in the Open function or configuration file, CPU is used for image preprocessing. Otherwise, if the `preproc_name` parameter is not set, MLU is used for image preprocessing. The image preprocessing includes data shape resizing and color space convertion. Afterwards, you can infer with offline model loading from the model path.

  Attention The error log will be reported when the following two situations occur as mlu is used to do preprocessing. case 1: scale-up factor is greater than 100. case 2: the image width before resize is greater than 7680.

2.5 Module

2.5.1 IModuleObserver

class cnstream::IModuleObserver

`class IModuleObserver`

IModuleObserver virtual base class.

IModuleObserver is an interface class. User need to implement an observer based on this, and register it to one module.

2.5.2 Module

class cnstream::Module

`class Module` : `private` NonCopyable

Module virtual base class.

Module is the parent class of all modules. A module could have configurable number of upstream links and downstream links. Some modules are already constructed with a framework, such as source, inferencer, and so on. You can also design your own modules.

Subclassed by cnstream::Inferencer, cnstream::ModuleEx, cnstream::RtspSink, cnstream::Tracker

2.5.3 ModuleCreator

class cnstream::ModuleCreator

template<typename `T`>
`class ModuleCreator`

ModuleCreator A concrete ModuleClass needs to inherit ModuleCreator to enable reflection mechanism. ModuleCreator provides `CreateFunction`, and registers `ModuleClassName` and `CreateFunction` to ModuleFactory().

2.5.4 ModuleCreatorWorker

class cnstream::ModuleCreatorWorker

`class ModuleCreatorWorker`

ModuleCreatorWorker, a dynamic-creator helper.

2.5.5 ModuleEx

class cnstream::ModuleEx

`class ModuleEx : public` cnstream::Module

ModuleEx class.

Module has permission to transmit data by itself.

2.5.6 ModuleFactory

class cnstream::ModuleFactory

`class ModuleFactory`

ModuleCreator, ModuleFactory, and ModuleCreatorWorker: Implements reflection mechanism to create a module instance dynamically with the `ModuleClassName` and `moduleName` parameters. See ActorFactory&DynamicCreator in https://github.com/Bwar/Nebula (under Apache2.0 license)

ModuleFactory Provides functions to create instances with the `ModuleClassName`and `moduleName` parameters.

2.6 Pipeline

2.6.1 LinkStatus

typedef struct {

bool stopped;

std::vector<uint32_t> cache_size;

} LinkStatus;

`struct cnstream::LinkStatus`

The link status between modules.

Public Members

bool `stopped`

Whether the data transmissions between the modules are stopped.

std::vector<uint32_t> `cache_size`

The size of each queue that is used to cache data between modules.

2.6.2 Pipeline

class cnstream::Pipeline

`class Pipeline` : `private` NonCopyable

The manager of the modules. Manages data transmission between modules, and controls messages delivery.

2.6.3 StreamMsg

typedef struct {

StreamMsgType type;

std::string stream_id;

CNFrameINfo::stream_id;

std::string module_name;

int64_t pts = -1;

} StreamMsg;

`struct cnstream::StreamMsg`

Specifies a stream message.

See StreamMsgType.

Public Members

StreamMsgType `type`

The type of a message.

std::string `stream_id`

Stream id, set by user in CNFrameINfo::stream_id.

std::string `module_name`

The module that posts this event.

int64_t `pts` = -1

The pts of this frame.

2.6.4 StreamMsgObserver

class cnstream::StreamMsgObserver

`class StreamMsgObserver`

Stream message observer.

Receives stream messages from a pipeline. To receive stream messages from the pipeline, you can define a class to inherit the StreamMsgObserver class and call the `Update` function. The observer instance is bounded to the pipeline using the Pipeline::SetStreamMsgObserver function .

See Pipeline::SetStreamMsgObserver StreamMsg StreamMsgType.

2.6.5 StreamMsgType

enum StreamMsgType {

EOS_MSG = 0,

ERROR_MSG,

STREAM_ERR_MSG,

FRAME_ERR_MSG,

USER_MSG0 = 32,

USER_MSG1,

USER_MSG2,

USER_MSG3,

USER_MSG4,

USER_MSG5,

USER_MSG6,

USER_MSG7,

USER_MSG8,

USER_MSG9

};

`enum cnstream::StreamMsgType`

Data stream message type.

Values:

`enumerator EOS_MSG` = 0

The end of a stream message. The stream has received EOS message in all modules.

`enumerator ERROR_MSG`

An error message. The stream process has failed in one of the modules.

`enumerator STREAM_ERR_MSG`

Stream error message, stream process failed at source.

`enumerator FRAME_ERR_MSG`

Frame error message, frame decode failed at source.

`enumerator USER_MSG0` = 32

Reserved message. You can define your own messages.

`enumerator USER_MSG1`

Reserved message. You can define your own messages.

`enumerator USER_MSG2`

Reserved message. You can define your own messages.

`enumerator USER_MSG3`
: Reserved message. You can define your own messages.

`enumerator USER_MSG4`
: Reserved message. You can define your own messages.

`enumerator USER_MSG5`
: Reserved message. You can define your own messages.

`enumerator USER_MSG6`
: Reserved message. You can define your own messages.

`enumerator USER_MSG7`
: Reserved message. You can define your own messages.

`enumerator USER_MSG8`
: Reserved message. You can define your own messages.

`enumerator USER_MSG9`
: Reserved message. You can define your own messages.

2.7 Profiler

2.7.1 ModuleProfiler

class ModuleProfiler : private NonCopyable

`class ModuleProfiler : private` NonCopyable

ModuleProfiler is responsible for the performance statistics of a module. ModuleProfiler contains multiple ProcessProfilers for multiple process proiling. The trace event of the processes will be recorded when `ProfilerConfig::enable_tracing` is true. Profiling and tracing of custom process is supported, see `RegisterProcessName` for detail. This class is thread-safe.

2.7.2 PipelineProfiler

class PipelineProfiler : private NonCopyable

`class PipelineProfiler : private` NonCopyable

PipelineProfiler is responsible for the performance statistics of a pipeline. PipelineProfiler contains multiple ModuleProfilers for multiple modules profiling.

By default, it will perform two processes of profiling for all modules. The two processes are named `kPROCESS_PROFILER_NAME` and `kINPUT_PROFILER_NAME`. The process named `kPROCESS_PROFILER_NAME` is started before Module::Process called and ended before Module::Transmit called. The process named `kINPUT_PROFILER_NAME` is started when datas go into the data queue of module and ended when datas start to be processed by module.

It also does profiling of the data processing process from entering to exiting the pipeline.

The start and end trace events of each process are recorded when the `config.enable_tracing` is true.

This class is thread-safe.

2.7.3 PipelineTracer

class PipelineTracer : private NonCopyable

`class PipelineTracer` : `private` NonCopyable

PipelineTracer can be used to record trace events for pipeline.

2.7.4 ProcessProfiler

class ProcessProfiler : private NonCopyable

`class ProcessProfiler` : `private` NonCopyable

A profiler for a process. A process can be a function call or a piece of code. This class is thread-safe.

2.7.5 StreamProfiler

class StreamProfiler

`class StreamProfiler`

StreamProfiler is responsible for the performance statistics of a certain processing process of a stream. It is used by ProcessProfiler.

See ProcessProfiler.

2.7.6 TraceSerializeHelper

class TraceSerializeHelper

`class TraceSerializeHelper`

Serialize trace data into json format. You can load json file by chrome-tracing to show the trace data.

2.7.7 StreamProfile

`struct cnstream::StreamProfile`

Public Functions

`StreamProfile(const` StreamProfile &it) = default

StreamProfile copy constructor.

Parameters

- `it`: which instance copy from.

StreamProfile `&operator=(const` StreamProfile &it) = default

StreamProfile operator =.

Parameters

- `it`: Which instance copy from.

Return Returns a lvalue reference to the current instance.

`StreamProfile(StreamProfile &&it)`

StreamProfile move constructor.

Parameters

- `it`: which instance move from.

`StreamProfile &operator=(StreamProfile &&it)`

StreamProfile operator =.

Parameters

- `it`: Which instance move from.

Return Returns a lvalue reference to the current instance.

Public Members

std::string `stream_name`

stream name.

uint64_t `counter` = 0

frame counter, it is equal to `completed` plus `dropped`.

uint64_t `completed` = 0

completed frame counter.

int64_t `dropped` = 0

dropped frame counter.

double `latency` = 0.0

average latency. (ms)

double `maximum_latency` = 0.0

maximum latency. (ms)

double `minimum_latency` = 0.0

minimum latency. (ms)

double `fps` = 0.0

fps.

2.7.8 ProcessProfile

`struct cnstream::ProcessProfile`

Public Functions

`ProcessProfile(const ProcessProfile &it) = default`
ProcessProfile copy constructor.

Parameters
- `it`: which instance copy from.

`ProcessProfile &operator=(const ProcessProfile &it) = default`
ProcessProfile operator =.

Parameters
- `it`: Which instance copy from.

Return Returns a lvalue reference to the current instance.

`ProcessProfile(ProcessProfile &&it)`
ProcessProfile move constructor.

Parameters
- `it`: which instance move from.

`ProcessProfile &operator=(ProcessProfile &&it)`
ProcessProfile operator =.

Parameters
- `it`: Which instance move from.

Return Returns a lvalue reference to the current instance.

Public Members

std::string `process_name`
process name.

uint64_t `counter` = 0
frame counter, it is equal to `completed` plus `dropped`.

uint64_t `completed` = 0
completed frame counter.

int64_t `dropped` = 0
dropped frame counter.

int64_t `ongoing` = 0
number of frame being processed.

double `latency` = 0.0
average latency. (ms)

double `maximum_latency` = 0.0
maximum latency. (ms)

double `minimum_latency` = 0.0
minimum latency. (ms)

double `fps` = 0.0
fps.

std::vector<StreamProfile> `stream_profiles`
: stream profiles.

2.7.9 ModuleProfile

`struct cnstream::ModuleProfile`

Public Functions

`ModuleProfile(const ModuleProfile &it) = default`
: ModuleProfile copy constructor.

  Parameters
  - `it`: which instance copy from.

`ModuleProfile &operator=(const ModuleProfile &it) = default`
: ModuleProfile operator =.

  Parameters
  - `it`: Which instance copy from.

  Return Returns a lvalue reference to the current instance.

`ModuleProfile(ModuleProfile &&it)`
: ModuleProfile move constructor.

  Parameters
  - `it`: which instance move from.

`ModuleProfile &operator=(ModuleProfile &&it)`
: ModuleProfile operator =.

  Parameters
  - `it`: Which instance move from.

  Return Returns a lvalue reference to the current instance.

Public Members

std::string `module_name`
: module name.

std::vector<ProcessProfile> `process_profiles`
: process profiles.

2.7.10 PipelineProfile

`struct cnstream::PipelineProfile`

Public Functions

`PipelineProfile(const` PipelineProfile &it`)` = default

PipelineProfile copy constructor.

Parameters

- `it`: which instance copy from.

PipelineProfile &`operator=(const` PipelineProfile &it`)` = default

PipelineProfile operator =.

Parameters

- `it`: Which instance copy from.

Return Returns a lvalue reference to the current instance.

`PipelineProfile(`PipelineProfile &&it`)`

PipelineProfile move constructor.

Parameters

- `it`: which instance move from.

PipelineProfile &`operator=(`PipelineProfile &&it`)`

PipelineProfile operator =.

Parameters

- `it`: Which instance move from.

Return Returns a lvalue reference to the current instance.

Public Members

std::string `pipeline_name`

pipeline name.

std::vector<ModuleProfile> `module_profiles`

module profiles.

ProcessProfile `overall_profile`

profile of the whole pipeline.

2.7.11 TraceEvent

`struct cnstream::TraceEvent`

Class TraceEvent represents an trace event.

Public Functions

`TraceEvent(const RecordKey &key)`

TraceEvent constructor.

Parameters
- `key`: Unique identification of a frame.

`TraceEvent(RecordKey &&key)`

TraceEvent constructor.

Parameters
- `key`: Unique identification of a frame.

`TraceEvent(const TraceEvent &other) = default`

TraceEvent copy constructor.

Parameters
- `other`: which instance copy from.

`TraceEvent &operator=(const TraceEvent &other) = default`

TraceEvent operator =.

Parameters
- `other`: Which instance copy from.

Return Returns a lvalue reference to the current instance.

`TraceEvent(TraceEvent &&other)`

TraceEvent move constructor.

Parameters
- `other`: which instance move from.

`TraceEvent &operator=(TraceEvent &&other)`

TraceEvent operator =.

Parameters
- `other`: Which instance move from.

Return Returns a lvalue reference to the current instance.

`TraceEvent &SetKey(const RecordKey &key)`

Set unique identification of a frame.

Parameters
- `key`: Unique identification of a frame.

Return Returns a lvalue reference to the current instance.

`TraceEvent &SetKey(RecordKey &&key)`

Set unique identification of a frame.

Parameters

- `key`: Unique identification of a frame.

Return Returns a lvalue reference to the current instance.

TraceEvent &`SetModuleName`(`const` std::string &module_name)

Set module name.

Parameters

- `module_name`: Module name.

Return Returns a lvalue reference to the current instance.

TraceEvent &`SetModuleName`(std::string &&module_name)

Set module name.

Parameters

- `module_name`: Module name.

Return Returns a lvalue reference to the current instance.

TraceEvent &`SetProcessName`(`const` std::string &process_name)

Set process name.

Parameters

- `process_name`: Process name.

Return Returns a lvalue reference to the current instance.

TraceEvent &`SetProcessName`(std::string &&process_name)

Set process name.

Parameters

- `process_name`: Process name.

Return Returns a lvalue reference to the current instance.

TraceEvent &`SetTime`(`const` Time &time)

Set time.

Parameters

- `time`: Time.

Return Returns a lvalue reference to the current instance.

TraceEvent &`SetTime`(Time &&time)

Set time.

Parameters

- `time`: Time.

Return Returns a lvalue reference to the current instance.

TraceEvent &`SetLevel`(`const` Level &level)

Set event level.

Parameters

- `level`: event level.

Return Returns a lvalue reference to the current instance.

TraceEvent &`SetType`(`const` Type &type)

Set event type.

Parameters

- `type`: event type.

Return Returns a lvalue reference to the current instance.

Public Members

RecordKey **`key`**
Unique identification of a frame.

std::string **`module_name`**
Module name.

std::string **`process_name`**
Process name. A process can be a function call or a piece of code.

Time **`time`**
Event time.

`enum` cnstream::TraceEvent::Level **`level`** = PIPELINE
Event level.

`enum` cnstream::TraceEvent::Type **`type`** = START
Event type.

2.7.12 TraceElem

`struct cnstream::TraceElem`

Public Functions

`TraceElem`(`const` TraceElem &other) = default
TraceElem copy constructor.

Parameters

- `other`: which instance copy from.

TraceElem &**`operator=`**(`const` TraceElem &other) = default
TraceElem operator =.

Parameters

- `other`: Which instance copy from.

Return Returns a lvalue reference to the current instance.

`TraceElem`(TraceElem &&other)
TraceElem move constructor.

Parameters

- `other`: which instance move from.

TraceElem &**`operator=`**(TraceElem &&other)
TraceElem operator =.

Parameters

- `other`: Which instance move from.

Return Returns a lvalue reference to the current instance.

`TraceElem(const` TraceEvent &event)

TraceElem constructor.

Parameters

- `event`: Trace event.

`TraceElem(`TraceEvent &&event)

TraceElem constructor.

Parameters

- `event`: Trace event.

Public Members

RecordKey **`key`**

Unique identification of a frame.

Time **`time`**

Event time.

TraceEvent::Type **`type`**

Event type. Process start or process end.

2.7.13 PipelineTrace

`struct cnstream::PipelineTrace`

Trace data for a pipeline.

Public Functions

`PipelineTrace()` = default

`PipelineTrace(const` PipelineTrace &other) = default

PipelineTrace copy constructor.

Parameters

- `other`: which instance copy from.

PipelineTrace `&operator=(const` PipelineTrace &other) = default

PipelineTrace operator =.

Parameters

- `other`: Which instance copy from.

Return Returns a lvalue reference to the current instance.

`PipelineTrace(`PipelineTrace &&other)

PipelineTrace move constructor.

Parameters

- `other`: which instance move from.

PipelineTrace &**operator=**(PipelineTrace &&other)
PipelineTrace operator =.

Parameters
- `other`: Which instance move from.

Return Returns a lvalue reference to the current instance.

Public Members

std::unordered_map<std::string, ModuleTrace> **module_traces**

2.7.14 RecordKey

`using cnstream::RecordKey` = std::pair<std::string, int64_t>
Unique identification of a frame in tracing and profiling. Usually, first: stream_name(CNFrameInfo::stream_id), second: pts(CNFrameInfo::timestamp).

2.7.15 ProcessTrace

`using cnstream::ProcessTrace` = std::vector<TraceElem>
Type of trace data for a process.

2.7.16 ModuleTrace

`using cnstream::ModuleTrace` = std::unordered_map<std::string, ProcessTrace>
Type of trace data for a module.

2.8 RTSP Sink

2.8.1 ColorFormat

enum ColorFormat {

YUV420 = 0,

RGB24,

BGR24,

NV21,

NV12,

};

`enum cnstream::ColorFormat`
The enum of color format.

Values:

`enumerator YUV420` = 0
  Planar Y4-U1-V1.

`enumerator RGB24`
  Packed R8G8B8.

`enumerator BGR24`
  Packed B8G8R8.

`enumerator NV21`
  Semi-Planar Y4-V1U1.

`enumerator NV12`
  Semi-Planar Y4-U1V1.

2.8.2 EncoderType

enum EncoderType {

FFMPEG = 0,

MLU,

};

`enum cnstream::EncoderType`
  The enum of encoder type.

  Values:

  `enumerator FFMPEG` = 0
    Encoder with ffmpeg.

  `enumerator MLU`
    Encoder with MLU.

2.8.3 RtspParam

typedef struct {

int frame_rate = 25;

int udp_port = 9554;

int http_port = 8080;

int src_width = 1920;

int src_height = 1080;

int dst_width = 1920;

int dst_height = 1080;

int gop = 20;

int kbps = 2 * 1024;

ColorFormat color_format = NV21;

```
    VideoCodecType codec_type = H264;
    EncoderType enc_type = FFMPEG;
    int device_id;
    int view_rows;
    int view_cols;
    std::string view_mode;
    std::string color_mode;
    std::string preproc_type;
    std::string encoder_type;
} RtspParam;
```

`struct cnstream::RtspParam`

The struct of rtsp parameters.

Public Members

int `frame_rate` = 25
Target fps.

int `udp_port` = 9554
UDP port.

int `http_port` = 8080
RTSP-over-HTTP channel port.

int `src_width` = 1920
Source width.

int `src_height` = 1080
Source height.

int `dst_width` = 1920
Target width, preferred size is the same with input.

int `dst_height` = 1080
Target height, prefered size is the same with input.

int `gop` = 20
Target gop, the default is 10.

int `kbps` = 2 * 1024
Target Kbps, the default is 2*1024(2M).

ColorFormat `color_format` = NV21
Color format.

VideoCodecType `codec_type` = H264
Video codec type.

EncoderType `enc_type` = FFMPEG
Encoder type.

int `device_id`
Device id.

int `view_rows`
Row of the display grid. Only used in mosaic mode.

int `view_cols`
Column of the display grid. Only used in mosaic mode.

std::string `view_mode`
Display mode.

std::string `color_mode`
Color mode.

std::string `preproc_type`
Preproc type.

std::string `encoder_type`
Encoder type.

2.8.4 RtspSink

class cnstream::RtspSink

`class RtspSink` : `public` cnstream::Module, `public` cnstream::ModuleCreator<RtspSink>
RtspSink is a module to deliver stream by RTSP protocol.

2.8.5 VideoCodecType

enum VideoCodecType {

H264 = 0,

HEVC,

MPEG4,

};

`enum cnstream::VideoCodecType`
The enum of video codec type.

Values:

`enumerator H264` = 0
The video in H264 type.

`enumerator HEVC`
The video in HEVC type.

`enumerator MPEG4`
The video in MPEG4 type.

2.9 SyncMem

2.9.1 CNSyncedMemory

class cnstream::CNSyncedMemory

`class CNSyncedMemory : private` NonCopyable

Synchronizes memory between CPU and MLU.

If the data on MLU is the latest, the data on CPU should be synchronized before processing the data on CPU. Vice versa, if the data on CPU is the latest, the data on MLU should be synchronized before processing the data on MLU.

Note CNSyncedMemory::Head() always returns CNSyncedMemory::UNINITIALIZED when memory size is 0.

2.9.2 SyncedHead

enum SyncedHead {

UNINITIALIZED,

HEAD_AT_CPU,

HEAD_AT_MLU,

SYNCED

};

`enum cnstream::CNSyncedMemory::SyncedHead`

Head synchronization.

Values:

`enumerator UNINITIALIZED`

The memory is not allocated.

`enumerator HEAD_AT_CPU`

The data is updated to CPU but is not synchronized to MLU yet.

`enumerator HEAD_AT_MLU`

The data is updated to MLU but is not synchronized to CPU yet.

`enumerator SYNCED`

The data is synchronized to both CPU and MLU.

2.10 Tracker

2.10.1 Tracker

class cnstream::Tracker

`class Tracker` : `public` cnstream::Module, `public` cnstream::ModuleCreator<Tracker>

Tracker is a module for realtime tracking. Extracts feature on MLU if the model path is provided. Otherwise, it would be done on CPU.

3 API Reference

CNStream APIs support both on MLU270 and MLU220.

3.1 Data Source

3.1.1 CheckParamSet

bool `cnstream::`*`DataSource`*`::CheckParamSet(const` ModuleParamSet &paramSet) `const override`

Check ParamSet for a module.

Parameters

- `paramSet`: Parameters for this module.

Return Returns true if this API run successfully. Otherwise, returns false.

3.1.2 DataSource::Close

void `cnstream::`*`DataSource`*`::Close() override`

Called by pipeline when pipeline stop.

3.1.3 RtspHandler::Close

void `cnstream::`*`RtspHandler`*`::Close() override`

Closes source handler.

3.1.4 ESMemHandler::Close

void `cnstream::`*`ESMemHandler`*`::Close() override`

Closes source handler.

3.1.5 ESJpegMemHandler::Close

void `cnstream::ESJpegMemHandler::Close() override`

Closes source handler.

3.1.6 RawImgMemHandler::Close

void `cnstream::RawImgMemHandler::Close() override`

Closes source handler.

3.1.7 RtspHandler::Create

std::shared_ptr<SourceHandler> `cnstream::RtspHandler::Create(`DataSource *module, `const` std::string &stream_id, `const` std::string &url_name, bool use_ffmpeg = false, int reconnect = 10)

Creates source handler.

Parameters

- `module`: The data source module.
- `stream_id`: The stream id of the stream.
- `url_name`: The url of the stream.
- `use_ffmpeg`: Uses ffmpeg demuxer if it is true, otherwise uses live555 demuxer.
- `reconnect`: It is valid when "use_ffmpeg" set false.

Return Returns source handler if it is created successfully, otherwise returns nullptr.

3.1.8 ESMemHandler::Create

std::shared_ptr<SourceHandler> `cnstream::ESMemHandler::Create(`DataSource *module, `const` std::string &stream_id)

Creates source handler.

Parameters

- `module`: The data source module.
- `stream_id`: The stream id of the stream.

Return Returns source handler if it is created successfully, otherwise returns nullptr.

3.1.9 ESJpegMemHandler::Create

std::shared_ptr<SourceHandler> `cnstream::ESJpegMemHandler::Create(DataSource *module, const std::string &stream_id, int max_width = 7680, int max_height = 4320)`

Creates source handler.

Parameters

- `module`: The data source module.
- `stream_id`: The stream id of the stream.
- `max_width`: The maximum width of the image.
- `max_height`: The maximum height of the image.

Return Returns source handler if it is created successfully, otherwise returns nullptr.

3.1.10 RawImgMemHandler::Create

std::shared_ptr<SourceHandler> `cnstream::RawImgMemHandler::Create(DataSource *module, const std::string &stream_id)`

Creates source handler.

Parameters

- `module`: The data source module.
- `stream_id`: The stream id of the stream.

Return Returns source handler if it is created successfully, otherwise returns nullptr.

3.1.11 GetSourceParam

DataSourceParam `cnstream::DataSource::GetSourceParam() const`

Get module parameters.

Return Returns data source parameters.

Note This function should be called after `Open` function.

3.1.12 DataSource::Open

bool `cnstream::DataSource::Open(ModuleParamSet paramSet) override`

Called by pipeline when the pipeline is started.

Parameters

- `paramSet`:
  - output_type: Optional. The output type. The default output_type is cpu. Supported values are `mlu` and `cpu`.

- interval: Optional. Process one frame for every `interval` frames. Process every frame by default.
- decoder_type : Optional. The decoder type. The default decoder_type is cpu. Supported values are `mlu` and `cpu`.
- reuse_cndec_buf: Optional. Whether the codec buffer will be reused. The default value is false. This parameter is used when decoder type is `mlu`. Supported values are `true` and `false`.
- device_id: Required when MLU is used. Device id. Set the value to -1 for CPU. Set the value for MLU in the range 0 - N.
- interlaced: Interlaced mode.
- input_buf_number: Optional. The input buffer number. The default value is 2.
- output_buf_number: Optional. The output buffer number. The default value is 3.
- apply_stride_align_for_scaler: Optional. Apply stride align for scaler on m220(m.2/edge).

Return true if paramSet are supported and valid, othersize false

3.1.13 RtspHandler::Open

bool `cnstream::`*`RtspHandler`*`::Open() override`

Opens source handler.

Return Returns true if the source handler is opened successfully, otherwise returns false.

3.1.14 ESMemHandler::Open

bool `cnstream::`*`ESMemHandler`*`::Open() override`

Opens source handler.

Return Returns true if the source handler is opened successfully, otherwise returns false.

3.1.15 ESJpegMemHandler::Open

bool `cnstream::`*`ESJpegMemHandler`*`::Open() override`

Opens source handler.

Return Returns true if the source handler is opened successfully, otherwise returns false.

3.1.16 RawImgMemHandler::Open

bool `cnstream::RawImgMemHandler::Open() override`

Opens source handler.

Return Returns true if the source handler is opened successfully, otherwise returns false.

3.1.17 RawImgMemHandler::SetDataType

int `cnstream::`*`ESMemHandler`*`::SetDataType(`DataType type`)`

Sets data type.

Parameters

- `type`: The data type.

Return Returns 0 if data type is set successfully, otherwise returns -1.

Note This function must be called before `Write` function

3.1.18 ESMemHandler::Write

int `cnstream::`*`ESMemHandler`*`::Write(`ESPacket *pkt`)`

Sends data in frame mode.

Parameters

- `pkt`: The data packet

Return Value

- `0`: The data is write successfully,
- `-1`: Write failed, maybe the handler is closed.
- `-2`: Invalid data. Can not parse video infomations from `pkt`.

3.1.19 ESMemHandler::Write

int `cnstream::`*`ESMemHandler`*`::Write(`unsigned char *buf, int len`)`

Sends data in chunk mode.

Parameters

- `buf`: The data buffer
- `len`: The len of the data

Return Value

- `0`: The data is write successfully,
- `-1`: Write failed, maybe the handler is closed.
- `-2`: Invalid data. Can not parse video infomations from `buf`.

3.1.20 ESJpegMemHandler::Write

int `cnstream::`*`ESJpegMemHandler`*`::Write(`ESPacket *pkt`)`

Sends data in frame mode.

Parameters

- `pkt`: The data packet.

Return Value

- `0`: The data is write successfully,
- `-1`: Write failed, maybe the handler is closed.
- `-2`: Invalid data. Can not parse image infomations from `pkt`.

3.1.21 RawImgMemHandler::Write

int `cnstream::RawImgMemHandler::Write(const` uint8_t *data, `const` int size, `const` uint64_t pts, `const` int width = 0, `const` int height = 0, `const` CNDataFormat pixel_fmt = CN_INVALID)

Sends raw image with image data and image infomation, support formats: bgr24, rgb24, nv21 and nv12.

Parameters

- `data`: The data of the image, which is a continuous buffer.
- `size`: The size of the data.
- `pts`: The pts for raw image, should be different for each image.
- `width`: The width of the image.
- `height`: The height of the image.
- `pixel_fmt`: The pixel format of the image. These formats are supported, bgr24, rgb24, nv21 and nv12.

Return Value

- `0`: The data is write successfully,
- `-1`: Write failed, maybe eos got or handler is closed.
- `-2`: Invalid data.

Note Sends nullptr as data and passes 0 as size after all data are sent.

3.2 Eventbus

3.2.1 AddBusWatch

uint32_t `cnstream::`*`EventBus`*`::AddBusWatch(`BusWatcher func`)`

Adds the watcher to the event bus.

Parameters

- `func`: The bus watcher to be added.

Return The number of bus watchers that has been added to this event bus.

3.2.2 BusWatcher

`typedef` std::function<EventHandleFlag(`const` Event&, Module*)> `cnstream::BusWatcher`

The bus watcher function.

Parameters

- `event`: The event polled from the event bus.
- *`Pipeline`*: The module that is watching.

Return Returns the flag that specifies how the event is handled.

3.2.3 ClearAllWatchers

void `cnstream::EventBus::ClearAllWatchers()`

Removes all bus watchers.

3.2.4 GetBusWatchers

`const` std::list<BusWatcher> &`cnstream::EventBus::GetBusWatchers() const`

Gets all bus watchers from the event bus.

Return A list with pairs of bus watcher and module.

3.2.5 IsRunning

bool `cnstream::EventBus::IsRunning()`

Checks if the event bus is running.

Return Returns true if the event bus is running. Otherwise, returns false.

3.2.6 PollEvent

Event `cnstream::EventBus::PollEvent()`

Polls an event from a bus [block].

Note This function is blocked until an event or a bus is stopped.

3.2.7 PostEvent

bool `cnstream::EventBus::PostEvent(`Event event`)`

Posts an event to a bus.

Parameters

- `event`: The event to be posted.

Return Returns true if this function run successfully. Otherwise, returns false.

3.2.8 Start

bool `cnstream::EventBus::Start()`

Starts an event bus thread.

3.2.9 Stop

void `cnstream::`*`EventBus`*`::`**`Stop`**`()`

Stops an event bus thread.

3.3 Frame

3.3.1 AddAttribute

bool `cnstream::`*`CNInferObject`*`::`**`AddAttribute`**`(const` std::pair<std::string, CNInferAttr> &attribute`)`

Adds the key pairs of an attribute to a specified object.

Parameters

- `attribute`: The attribute pair (key, value) to be added.

Return Returns true if the attribute has been added successfully. Returns false if the attribute has already existed.

Note This is a thread-safe function.

3.3.2 AddAttribute

bool `cnstream::`*`CNInferObject`*`::`**`AddAttribute`**`(const` std::string &key, `const` CNInferAttr &value`)`

Adds the key of an attribute to a specified object.

Parameters

- `key`: The Key of the attribute you want to add to. See GetAttribute().
- `value`: The value of the attribute.

Return Returns true if the attribute has been added successfully. Returns false if the attribute already existed.

Note This is a thread-safe function.

3.3.3 AddExtraAttribute

bool `cnstream::`*`CNInferObject`*`::`**`AddExtraAttribute`**`(const` std::string &key, `const` std::string &value`)`

Adds the key of the extended attribute to a specified object.

Parameters

- `key`: The key of an attribute. You can get this attribute by key. See GetExtraAttribute().
- `value`: The value of the attribute.

Return Returns true if the attribute has been added successfully. Returns false if the attribute has already existed in the object.

Note This is a thread-safe function.

3.3.4 AddExtraAttributes

bool `cnstream::CNInferObject::AddExtraAttributes(const std::vector<std::pair<std::string, std::string>> &attributes)`

Adds the key pairs of the extended attributes to a specified object.

Parameters

- `attributes`: Attributes to be added.

Return Returns true if the attribute has been added successfully. Returns false if the attribute has already existed.

Note This is a thread-safe function.

3.3.5 AddFeature

bool `cnstream::CNInferObject::AddFeature(const std::string &key, const CNInferFeature &feature)`

Adds the key of feature to a specified object.

Parameters

- `key`: The Key of feature you want to add the feature to. See GetFeature.
- `value`: The value of the feature.

Return Returns true if the feature is added successfully. Returns false if the feature identified by the key already exists.

Note This is a thread-safe function.

3.3.6 CNGetPlanes

int `cnstream::CNGetPlanes(CNDataFormat fmt)`

Gets image plane number by a specified image format.

Parameters

- `fmt`: The format of the image.

Return Value

- `0`: Unsupported image format.
- `>0`: Image plane number.

Return

3.3.7 CopyToSharedMem

void `cnstream::CNDataFrame::CopyToSharedMem(MemMapType type, std::string stream_id)`

Copies source-data to shared memory for multi-process.

Parameters

- `memory`: The type of the mapped or shared memory.

Return Void.

3.3.8 CopyToSyncMem

void `cnstream::`*`CNDataFrame`*`::`**`CopyToSyncMem`**(bool dst_mlu = true)

Synchronizes the source-data to CNSyncedMemory, inside the mlu device only.

3.3.9 CopyToSyncMemOnDevice

void `cnstream::`*`CNDataFrame`*`::`**`CopyToSyncMemOnDevice`**(int device_id)

Synchronizes source data to specific device, and resets ctx.dev_id to device_id when synced, for multi-device case.

Parameters

- `device_id`: The device id.

Return Void.

3.3.10 Create

std::shared_ptr<CNFrameInfo> `cnstream::`*`CNFrameInfo`*`::`**`Create`**(`const` std::string &stream_id, bool eos = false, std::shared_ptr<CNFrameInfo> payload = nullptr)

Creates a CNFrameInfo instance.

Parameters

- `stream_id`: The data stream alias. Identifies which data stream the frame data comes from.
- `eos`: Whether this is the end of the stream. This parameter is set to false by default to create a CNFrameInfo instance. If you set this parameter to true, CNDataFrame::flags will be set to `CN_FRAME_FLAG_EOS`. Then, the modules do not have permission to process this frame. This frame should be handed over to the pipeline for processing.

Return Returns `shared_ptr` of *`CNFrameInfo`* if this function has run successfully. Otherwise, returns NULL.

3.3.11 GetAttribute

CNInferAttr `cnstream::`*`CNInferObject`*`::`**`GetAttribute`**(`const` std::string &key)

Gets an attribute by key.

Parameters

- `key`: The key of an attribute you want to query. See AddAttribute().

Return Returns the attribute key. If the attribute does not exist, CNInferAttr::id will be set to -1.

Note This is a thread-safe function.

3.3.12 GetBytes

size_t `cnstream::`*`CNDataFrame`*`::`**`GetBytes`**`() const`

Gets the number of bytes in a frame.

Return Returns the number of bytes in a frame.

3.3.13 GetCNDataFramePtr

CNDataFramePtr `cnstream::`**`GetCNDataFramePtr`**`(`std::shared_ptr<CNFrameInfo> frameInfo`)`

3.3.14 GetExtraAttribute

std::string `cnstream::`*`CNInferObject`*`::`**`GetExtraAttribute`**`(const` std::string &key`)`

Gets an extended attribute by key.

Parameters

- `key`: The key of an identified attribute. See AddExtraAttribute().

Return Returns the attribute that is identified by the key. If the attribute does not exist, returns NULL.

Note This is a thread-safe function.

3.3.15 GetExtraAttributes

StringPairs `cnstream::`*`CNInferObject`*`::`**`GetExtraAttributes`**`()`

Gets all extended attributes of an object.

Return Returns all extended attributes.

Note This is a thread-safe function.

3.3.16 GetFeature

CNInferFeature `cnstream::`*`CNInferObject`*`::`**`GetFeature`**`(const` std::string &key`)`

Gets an feature by key.

Parameters

- `key`: The key of an feature you want to query. See AddFeature.

Return Return the feature of the key. If the feature identified by the key is not exists, CNInferFeature will be empty.

Note This is a thread-safe function.

3.3.17 GetFeatures

CNInferFeatures `cnstream::`*`CNInferObject`*`::GetFeatures()`

Gets the features of an object.

Return Returns the features of an object.

Note This is a thread-safe function.

3.3.18 GetCNInferDataPtr

CNInferDataPtr `cnstream::GetCNInferDataPtr(`std::shared_ptr<CNFrameInfo> frameInfo`)`

3.3.19 GetCNInferObjsPtr

CNInferObjsPtr `cnstream::GetCNInferObjsPtr(`std::shared_ptr<CNFrameInfo> frameInfo`)`

3.3.20 GetMediaImage

void *`cnstream::`*`ICNMediaImageMapper`*`::GetMediaImage()` = 0

Gets an image.

Return Returns the image address.

3.3.21 GetPlanes

int `cnstream::`*`CNDataFrame`*`::GetPlanes() const`

Gets plane count for a specified frame.

Return Returns the plane count of this frame.

3.3.22 GetPlaneBytes

size_t `cnstream::`*`CNDataFrame`*`::GetPlaneBytes(`int plane_idx`) const`

Gets the number of bytes in a specified plane.

Parameters

- `plane_idx`: The index of the plane. The index increments from 0.

Return Returns the number of bytes in the plane.

3.3.23 ImageBGR

cv::Mat *`cnstream::`*`CNDataFrame`*`::ImageBGR()`

Converts data from RGB to BGR. Called after CopyToSyncMem() is invoked.

If data is not RGB image but BGR, YUV420NV12 or YUV420NV21 image, its color mode will not be converted.

Return Returns data with opencv mat type.

3.3.24 ~ICNMediaImageMapper

`cnstream::`*`ICNMediaImageMapper`*`::~ICNMediaImageMapper()`

Destructor of class ICNMediaImageMapper.

3.3.25 IsEos

bool `cnstream::`*`CNFrameInfo`*`::IsEos()`

Whether DataFrame is end of stream (EOS) or not.

Return Returns true if the frame is EOS. Returns false if the frame is not EOS.

3.3.26 MmapSharedMem

void `cnstream::`*`CNDataFrame`*`::MmapSharedMem(`MemMapType type, std::string stream_id)

Maps shared memory for multi-process.

Parameters

- `memory`: The type of the mapped or shared memory.

Return Void.

3.3.27 ReleaseSharedMem

void `cnstream::`*`CNDataFrame`*`::ReleaseSharedMem(`MemMapType type, std::string stream_id)

Releases shared memory for multi-process.

Parameters

- `memory`: The type of the mapped or shared memory.

Return Void.

3.3.28 RemoveExtraAttribute

bool `cnstream::`*`CNInferObject`*`::`**`RemoveExtraAttribute`**`(`**`const`** std::string &key`)`

Removes an attribute by key.

Parameters

- `key`: The key of an attribute you want to remove. See AddAttribute.

Return Return true.

Note This is a thread-safe function.

3.3.29 SetStreamIndex

void `cnstream::`*`CNFrameInfo`*`::`**`SetStreamIndex`**`(`uint32_t index`)`

Sets index (usually the index is a number) to identify stream. This is only used for distributing each stream data to the appropriate thread. We do not recommend SDK users to use this API because it will be removed later.

Parameters

- `index`: Number to identify stream.

Return Returns true if the frame is EOS. Returns false if the frame is not EOS.

3.3.30 UnMapSharedMem

void `cnstream::`*`CNDataFrame`*`::`**`UnMapSharedMem`**`(`MemMapType type`)`

Unmaps the shared memory for multi-process.

Parameters

- `memory`: The type of the mapped or shared memory.

Return Void.

3.4 Inferencer

3.4.1 CheckParamSet

bool `cnstream::`*`Inferencer`*`::`**`CheckParamSet`**`(`**`const`** ModuleParamSet &param_set`)` **`const override`**

Checks parameters for a module.

Parameters

- `param_set`: Parameters of this module.

Return Returns true if this function has run successfully. Otherwise, returns false.

3.4.2 Close

void `cnstream::Inferencer::Close() override`

Called by pipeline when the pipeline is stopped.

Return Void.

3.4.3 Open

bool `cnstream::Inferencer::Open(ModuleParamSet paramSet) override`

Called by pipeline when the pipeline is started.

Parameters

- `paramSet`:
  - model_path: Required. The path of the offline model.
  - func_name: Required. The function name that is defined in the offline model. It could be found in Cambricon twins file. For most cases, it is "subnet0".
  - postproc_name: Required. The class name for postprocess. The class specified by this name must inherited from class cnstream::Postproc when [object_infer] is false, otherwise the class specified by this name must inherit from class cnstream::ObjPostproc.
  - preproc_name: Optional. The class name for preprocessing on CPU. The class specified by this name must inherited from class cnstream::Preproc when [object_infer] is false, otherwise the class specified by this name must inherit from class cnstream::ObjPreproc. Preprocessing will be done on MLU by ResizeYuv2Rgb (cambricon Bang op) when this parameter not set.
  - use_scaler: Optional. Whether use the scaler to preprocess the input. The scaler will not be used by default.
  - device_id: Optional. MLU device ordinal number. The default value is 0.
  - batching_timeout: Optional. The batching timeout. The default value is 3000.0[ms]. type[float]. unit[ms].
  - data_order: Optional. Data format. The default format is NHWC.
  - threshold: Optional. The threshold of the confidence. By default it is 0.
  - infer_interval: Optional. Process one frame for every `infer_interval` frames.
  - show_stats: Optional. Whether show inferencer performance statistics. It will not be shown by default.
  - stats_db_name: Required when show_stats is set to `true`. The directory to store the db file. e.g., `dir1/dir2/detect.db`.
  - object_infer: Optional. if object_infer is set to true, the detection target is used as the input to inferencing. if it is set to false, the video frame is used as the input to inferencing. False by default.
  - obj_filter_name: Optional. The class name for object filter. See cnstream::ObjFilter. This parameter is valid when object_infer is true. When this parameter not set, no object will be filtered. keep_aspect_ratio: Optional. As the mlu is used for image processing, the scale remains constant. model_input_pixel_format: Optional. As the mlu is used for image processing, set the pixel format of the model input image. RGBA32 by default. mem_on_mlu_for_postproc: Optional. Pass a batch mlu pointer

directly to post-processing function without making d2h copies. see `Postproc` for details. saving_infer_input: Optional. Save the data close to inferencing.

Return Returns ture if the inferencer has been opened successfully.

3.4.4 Process

int `cnstream::`*`Inferencer`*`::`**`Process`**`(`CNFrameInfoPtr data`)` **`final`**

Performs inference for each frame.

Parameters

- `data`: The information and data of frames.

Return Value

- `1`: The process has run successfully.
- `-1`: The process is failed.

3.5 Module

3.5.1 CheckParamSet

bool `cnstream::`*`Module`*`::`**`CheckParamSet`**`(`**`const`** ModuleParamSet &paramSet`)` **`const`**

Checks parameters for a module, including parameter name, type, value, validity, and so on.

Parameters

- `paramSet`: Parameters for this module.

Return Returns true if this function has run successfully. Otherwise, returns false.

3.5.2 Close

void `cnstream::`*`Module`*`::`**`Close`**`()` = 0

Closes resources for a module.

Return Void.

Note You do not need to call this function by yourself. This function is called by pipeline automatically when the pipeline is stopped. The pipeline calls the `Close` function of this module automatically after the `Open` and `Process` functions are done.

3.5.3 Create

Module *`cnstream::`*`ModuleFactory`*`::`**`Create`**`(`**`const`** std::string &strTypeName, **`const`** std::string &name`)`

Creates a module instance with `ModuleClassName` and `moduleName`.

Parameters

- `strTypeName`: The module class name.
- `name`: The `CreateFunction` of a Module object that has a parameter `moduleName`.

Return Returns the module instance if this function has run successfully. Otherwise, returns nullptr if failed.

3.5.4 Create

Module *`cnstream::`*`ModuleCreatorWorker`*`::`**`Create`**`(`**`const`** std::string &strTypeName, **`const`** std::string &name`)`

Creates a module instance with `ModuleClassName` and `moduleName`.

Parameters

- `strTypeName`: The module class name.
- `name`: The module name.

Return Returns the module instance if the module instance is created successfully. Returns nullptr if failed.

See ModuleFactory::Create

3.5.5 CreateObject

T *`cnstream::ModuleCreator::`**`CreateObject`**`(`**`const`** std::string &name`)`

Creates an instance of template (T) with specified instance name.

This is a template function.

Parameters

- `name`: The name of the instance.

Return Returns the instance of template (T).

3.5.6 DoProcess

int `cnstream::`*`Module`*`::`**`DoProcess`**`(`std::shared_ptr<CNFrameInfo> data`)`

Processes the data.

This function is called by a pipeline.

Parameters

- `data`: A pointer to the information of the frame.

Return Value

- `0`: The process has been run successfully. The data should be transmitted by framework then.
- `>0`: The process has been run successfully. The data has been handled by this module. The `hasTransmit_` must be set. The Pipeline::ProvideData should be called by Module to transmit data to the next modules in the pipeline.
- `<0`: Pipeline posts an event with the EVENT_ERROR event type and return number.

3.5.7 GetName

std::string `cnstream::`*`Module`*`::GetName() const`

Gets the name of this module.

Return Returns the name of this module.

3.5.8 GetRegisted

std::vector<std::string> `cnstream::`*`ModuleFactory`*`::GetRegisted()`

Gets all registered modules.

Return All registered module class names.

3.5.9 HasTransmit

bool `cnstream::`*`Module`*`::HasTransmit() const`

Checks if this module has permission to transmit data by itself.

Return Returns true if this module has permission to transmit data by itself. Otherwise, returns false.

See Process

3.5.10 Instance

ModuleFactory *`cnstream::`*`ModuleFactory`*`::Instance()`

Creates or gets the instance of the ModuleFactory class.

Return Returns the instance of the ModuleFactory class.

3.5.11 OnEos

void `cnstream::`*`Module`*`::OnEos(const` std::string &stream_id)

Notify flow-EOS arrives, the module should reset internal status if needed.

Please be noted: this function will be invoked when flow-EOS is forwarded by the framework

3.5.12 Open

bool `cnstream::`*`Module`*`::Open(`ModuleParamSet param_set) = 0

Opens resources for a module.

Parameters

- `param_set`: A set of parameters for this module.

Return Returns true if this function has run successfully. Otherwise, returns false.

Note You do not need to call this function by yourself. This function is called by pipeline automatically when the pipeline is started. The pipeline calls the `Process` function of this module automatically after the `Open` function is done.

3.5.13 PostEvent

bool `cnstream::`*`Module`*`::`**`PostEvent`**`(`Event e`)`

Posts an event to the pipeline.

Parameters

- *`Event`*: with event type, stream_id, message, module name and thread_id.

Return Returns true if this function has run successfully. Returns false if this module has not been added to the pipeline.

3.5.14 PostEvent

bool `cnstream::`*`Module`*`::`**`PostEvent`**`(`EventType type, **`const`** std::string &msg`)`

Posts an event to the pipeline.

Parameters

- `type`: The type of an event.
- `msg`: The event message string.

Return Returns true if this function has run successfully. Returns false if this module has not been added to the pipeline.

3.5.15 Process

int `cnstream::`*`Module`*`::`**`Process`**`(`std::shared_ptr<CNFrameInfo> data`)` = 0

Processes data.

Parameters

- `data`: The data to be processed by the module.

Return Value

- `0`: The data is processed successfully. The data should be transmitted in the framework then.
- `>0`: The data is processed successfully. The data has been handled by this module. The `hasTransmit_` must be set. The Pipeline::ProvideData should be called by Module to transmit data to the next modules in the pipeline.
- `<0`: Pipeline will post an event with the EVENT_ERROR event type and return number.

3.5.16 Regist

bool `cnstream::`*`ModuleFactory`*`::`**`Regist`**`(`**`const`** std::string &strTypeName, std::function<Module*`)` **`const`** std::string&

> pFuncRegisters `ModuleClassName` and `CreateFunction`.

Parameters

- `strTypeName`: The module class name.
- `pFunc`: The `CreateFunction` of a Module object that has a parameter `moduleName`.

Return Returns true if this function has run successfully.

3.5.17 SetContainer

void `cnstream::`*`Module`*`::`**`SetContainer`**`(`Pipeline *container`)`

Sets a container to this module and identifies which pipeline the module is added to.

Parameters

- `container`: A pipeline pointer to the container of this module.

Note This function is called automatically by the pipeline after this module is added into the pipeline. You do not need to call this function by yourself.

3.5.18 SetObserver

void `cnstream::`*`Module`*`::`**`SetObserver`**`(`IModuleObserver *observer`)`

Registers an observer to the module.

Parameters

- `observer`: An observer you defined.

Return Void.

3.5.19 TransmitData

bool `cnstream::`*`Module`*`::`**`TransmitData`**`(`std::shared_ptr<CNFrameInfo> data`)`

Transmits data to the following stages.

Valid when the module has permission to transmit data by itself.

Parameters

- `data`: A pointer to the information of the frame.

Return Returns true if the data has been transmitted successfully. Otherwise, returns false.

3.6 Pipelines

3.6.1 AddModule

bool `cnstream::`*`Pipeline`*`::`**`AddModule`**`(`std::shared_ptr<Module> module`)`

Adds the module to a pipeline.

Parameters

- `module`: The module instance to be added to this pipeline.

Return Returns true if this function has run successfully. Returns false if the module has been added to this pipeline.

3.6.2 AddModuleConfig

int `cnstream::`*`Pipeline`*`::`**`AddModuleConfig`**`(const` CNModuleConfig &config`)`

Adds module configurations in a pipeline.

Parameters

- `The`: configuration of a module.

Return Returns 0 if this function has run successfully. Otherwise, returns -1.

3.6.3 BuildPipeline

int `cnstream::`*`Pipeline`*`::`**`BuildPipeline`**`(const` std::vector<CNModuleConfig> &module_configs, `const` ProfilerConfig &profiler_config = ProfilerConfig()`)`

Builds a pipeline by module configurations.

Parameters

- `module_configs`: The configurations of a module.
- `profiler_config`: The configuration of profiler.

Return Returns 0 if this function has run successfully. Otherwise, returns -1.

3.6.4 BuildPipelineByJSONFile

int `cnstream::`*`Pipeline`*`::`**`BuildPipelineByJSONFile`**`(const` std::string &config_file`)`

Builds a pipeline from a JSON file.

```
{
  "source" : {
               "class_name" : "cnstream::DataSource",
               "parallelism" : 0,
               "next_modules" : ["detector"],
               "custom_params" : {
                 "decoder_type" : "mlu",
                 "device_id" : 0
               }
            },
   "detector" : {...}
}
```

Parameters

- `config_file`: The configuration file in JSON format.

Return Returns 0 if this function has run successfully. Otherwise, returns -1.

3.6.5 GetEndModule

Module *`cnstream::`*`Pipeline`*`::`**`GetEndModule`**`()`

Gets end module in pipeline(only valid when pipeline graph converged at end module).

Return Returns endmodule pointer when endmodule found and pipeline graph is converged at it, otherwise return nullptr.

3.6.6 GetEventBus

EventBus *`cnstream::`*`Pipeline`*`::`**`GetEventBus`**`() const`

Gets the event bus in the pipeline.

Return Returns the event bus.

3.6.7 GetModule

Module *`cnstream::`*`Pipeline`*`::`**`GetModule`**`(const` std::string &moduleName`)`

Gets a module in a pipeline by name.

Parameters

- `moduleName`: The module name specified in the module constructor.

Return Returns the module pointer if the module named `moduleName` has been added to the pipeline. Otherwise, returns nullptr.

3.6.8 GetModuleConfig

CNModuleConfig `cnstream::`*`Pipeline`*`::`**`GetModuleConfig`**`(const` std::string &module_name`)`

Gets the module configuration by the module name.

Parameters

- `module_name`: The module name specified in module constructor.

Return Returns module configuration if this function has run successfully. Returns NULL if the module specified by `module_name` has not been added to this pipeline.

3.6.9 GetModuleParamSet

ModuleParamSet `cnstream::`*`Pipeline`*`::`**`GetModuleParamSet`**`(const` std::string &moduleName`)`

Gets parameter set of a module. Module parameter set is used in Module::Open. It provides the ability for modules to customize parameters.

Parameters

- `moduleName`: The module name specified in the module constructor.

Return Returns the customized parameters of the module. If the module does not have customized parameters or the module has not been added to this pipeline, then the value of size (ModuleParamSet::size) is 0.

See Module::Open.

3.6.10 GetStreamMsgObserver

StreamMsgObserver *`cnstream::Pipeline::GetStreamMsgObserver() const`

Gets the stream message observer that has been bound with this pipeline.

Return Returns the stream message observer that has been bound with this pipeline.

See Pipeline::SetStreamMsgObserver.

3.6.11 IsLeafNode

bool `cnstream::Pipeline::IsLeafNode(const` std::string &node_name) `const`

Return if module is leaf node of pipeline.

Parameters

- `node_name`: module name.

Return True for yes, false for no.

3.6.12 IsProfilingEnabled

bool `cnstream::Pipeline::IsProfilingEnabled() const`

Is profiling enabled.

Return Returns true if profiling is enabled.

3.6.13 IsRunning

bool `cnstream::Pipeline::IsRunning() const`

The running status of a pipeline.

Return Returns true if the pipeline is running. Returns false if the pipeline is not running.

3.6.14 IsRootNode

bool `cnstream::Pipeline::IsRootNode(const` std::string &node_name) `const`

Return if module is root node of pipeline.

Parameters

- `node_name`: module name.

Return True for yes, false for no.

3.6.15 IsTracingEnabled

bool `cnstream::Pipeline::IsTracingEnabled() const`

Is tracing enabled

Return Returns true if tracing is enabled.

3.6.16 LinkModules

std::string `cnstream::Pipeline::LinkModules(`std::shared_ptr<Module> up_node, std::shared_ptr<Module> down_node`)`

Links two modules. The upstream node will process data before the downstream node.

Parameters

- `up_node`: The upstream module.
- `down_node`: The downstream module.

Return Returns the link-index if this function has run successfully. The link-index can used to query link status between `up_node` and `down_node`. See Pipeline::QueryStatus for details. Returns NULL if one of the two nodes has not been added to this pipeline.

Note Both `up_node` and `down_node` should be added to this pipeline before calling this function.

See Pipeline::QueryStatus.

3.6.17 ProvideData

bool `cnstream::Pipeline::ProvideData(const` Module *module, std::shared_ptr<CNFrameInfo> data`)`

Provides data for this pipeline that is used in source module or the module transmission by itself.

Parameters

- `module`: The module that provides data.
- `data`: The data that is transmitted to the pipeline.

Return Returns true if this function has run successfully. Returns false if the module is not added in the pipeline or the pipeline has been stopped.

See Module::Process.

3.6.18 QueryLinkStatus

bool `cnstream::Pipeline::QueryLinkStatus(`LinkStatus *status, `const` std::string &link_id`)`

Queries the link status by link-index. link-index is returned by Pipeline::LinkModules.

Parameters

- `status`: The link status to query.
- `link_id`: The Link-index returned by Pipeline::LinkModules.

Return Returns true if this function has run successfully. Otherwise, returns false.

See Pipeline::LinkModules.

3.6.19 RegistIPCFrameDoneCallBack

void `cnstream::`*`Pipeline`*`::`**`RegistIPCFrameDoneCallBack`**(std::function<void)std::shared_ptr<CNFrameInfo>
> callbackRegisters a callback to be called after the frame process is done.

Return Void.

3.6.20 SetModuleAttribute

bool `cnstream::`*`Pipeline`*`::`**`SetModuleAttribute`**(std::shared_ptr<Module> module, uint32_t parallelism, size_t queue_capacity = 20)

Sets the parallelism and conveyor capacity attributes of the module.

The SetModuleParallelism function is deprecated. Please use the SetModuleAttribute function instead.

Parameters

- `module`: The module to be configured.
- `parallelism`: Module parallelism, as well as Module' s conveyor number of input connector.
- `queue_capacity`: The queue capacity of the Module input conveyor.

Return Returns true if this function has run successfully. Returns false if this module has not been added to this pipeline.

Note You must call this function before calling Pipeline::Start.

See CNModuleConfig::parallelism.

3.6.21 SetStreamMsgObserver

void `cnstream::`*`Pipeline`*`::`**`SetStreamMsgObserver`**(StreamMsgObserver *observer)

Binds the stream message observer with this pipeline to receive stream message from this pipeline.

Parameters

- `observer`: The stream message observer.

Return Void.

See StreamMsgObserver.

3.6.22 Start

bool `cnstream::`*`Pipeline`*`::`**`Start`**()

Starts a pipeline. Starts data transmission in a pipeline. Calls the `Open` function for all modules. See Module::Open. Links modules.

Return Returns true if this function has run successfully. Returns false if the `Open` function did not run successfully in one of the modules, or the link modules failed.

3.6.23 Stop

bool `cnstream::`*`Pipeline`*`::`**`Stop`**`()`

Stops data transmissions in a pipeline.

Return Returns true if this function has run successfully. Otherwise, returns false.

3.7 Profiler

3.7.1 ModuleProfiler

ModuleProfiler

`cnstream::`*`ModuleProfiler`*`::`**`ModuleProfiler`**`(const` ProfilerConfig &config, `const` std::string &module_name, PipelineTracer *tracer)

Constructor of ModuleProfiler.

Parameters

- `config`: Profiler config.
- `module_name`: Module name.
- `tracer`: Tool for tracing.

RegisterProcessName

bool `cnstream::`*`ModuleProfiler`*`::`**`RegisterProcessName`**`(const` std::string &process_name)

Registers process named by `process_name` for this profiler.

Parameters

- `process_name`: The process name is the unique identification of a function or a piece of code that needs to do profiling.

Return True for Register successed. False will be returned when the process named by `process_name` has already been registered.

RecordProcessStart

bool `cnstream::`*`ModuleProfiler`*`::`**`RecordProcessStart`**`(const` std::string &process_name, `const` RecordKey &key)

Records the start of a process named `process_name`.

Parameters

- `process_name`: The name of a process. process_name is registed by `RegisterProcessName`.
- `key`: Unique identifier of a CNFrameInfo instance.

Return Ture for record successed. False will be returned when the process named by `process_name` has not been registered by `RegisterProcessName`.

See RegisterProcessName

See RecordKey

RecordProcessEnd

bool `cnstream::ModuleProfiler::RecordProcessEnd(const` std::string &process_name, `const` RecordKey &key)

Records the end of a process named `process_name`.

Parameters

- `process_name`: The name of a process. process_name is registed by `RegisterProcessName`.
- `key`: Unique identifier of a CNFrameInfo instance.

Return Ture for record successed. False will be returned when the process named by `process_name` has not been registered by `RegisterProcessName`.

See RegisterProcessName

See RecordKey

OnStreamEos

void `cnstream::ModuleProfiler::OnStreamEos(const` std::string &stream_name)

Tells the profiler to clear datas of stream named by `stream_name`.

Parameters

- `stream_name`: Stream name. Usually it is comes from `CNFrameInfo::stream_id`.

Return void.

GetName

std::string `cnstream::ModuleProfiler::GetName() const`

Gets name of module.

GetProfile

ModuleProfile `cnstream::ModuleProfiler::GetProfile()`

Gets profiling results of the whole run time.

Return Returns the profiling results.

GetProfile

ModuleProfile `cnstream::ModuleProfiler::GetProfile(const` ModuleTrace &trace)

Gets profiling results according to the trace datas.

Parameters

- `trace`: Trace datas.

Return Returns the profiling results.

3.7.2 PipelineProfiler

PipelineProfiler

`cnstream::PipelineProfiler::PipelineProfiler(const ProfilerConfig &config, const std::string &pipeline_name, const std::vector<std::shared_ptr<Module>> &modules)`

Constructor of ModuleProfiler.

Parameters

- `config`: Profiler config.
- `pipeline_name`: Pipeline name.
- `modules`: modules in the pipeline named `pipeline_name`.

GetName

`std::string cnstream::PipelineProfiler::GetName() const`

Gets name of pipeline.

GetTracer

`PipelineTracer *cnstream::PipelineProfiler::GetTracer() const`

Gets tracer.

Return Returns tracer.

GetModuleProfiler

`ModuleProfiler *cnstream::PipelineProfiler::GetModuleProfiler(const std::string &module_name) const`

Gets module profiler by module name.

Parameters

- `module_name`: Name of module.

Return Returns module profiler.

GetProfile

`PipelineProfile cnstream::PipelineProfiler::GetProfile()`

Gets profiling results of the whole run time.

Return Returns the profiling results.

GetProfile

PipelineProfile `cnstream::`*`PipelineProfiler`*`::`**`GetProfile`**`(const` Time &start, `const` Time &end)

Gets profiling results from `start` to `end`.

Parameters

- `start`: Start time.
- `end`: End time.

Return Returns the profiling results.

GetProfileBefore

PipelineProfile `cnstream::`*`PipelineProfiler`*`::`**`GetProfileBefore`**`(const` Time &end, `const` Duration &duration)

Gets profiling results for a specified period time.

Parameters

- `end`: End time.
- `duration`: Length of time before `end`.

Return Returns the profiling results.

GetProfileAfter

PipelineProfile `cnstream::`*`PipelineProfiler`*`::`**`GetProfileAfter`**`(const` Time &start, `const` Duration &duration)

Gets profiling results for a specified period time.

Parameters

- `start`: Start time.
- `duration`: Length of time after `start`.

Return Returns the profiling results.

RecordInput

void `cnstream::`*`PipelineProfiler`*`::`**`RecordInput`**`(const` RecordKey &key)

Record the time when the data enters the pipeline.

Parameters

- `key`: Unique identifier of a CNFrameInfo instance.

Return void.

See RecordKey

RecordOutput

void `cnstream::PipelineProfiler::RecordOutput(const RecordKey &key)`

Record the time when the data exits the pipeline.

Parameters

- `key`: Unique identifier of a CNFrameInfo instance.

Return void.

See RecordKey

OnStreamEos

void `cnstream::PipelineProfiler::OnStreamEos(const std::string &stream_name)`

Tells the profiler to clear datas of stream named by `stream_name`.

Parameters

- `stream_name`: Stream name. Usually it is comes from `CNFrameInfo::stream_id`.

Return void.

3.7.3 PipelineTracer

PipelineTracer

`cnstream::PipelineTracer::PipelineTracer(size_t capacity = 100000)`

Constructor of PipelineTracer.

It used to do tracing and store trace events.

Parameters

- `capacity`: Capacity to store trace events.

RecordEvent

void `cnstream::PipelineTracer::RecordEvent(TraceEvent &&event)`

Records trace event.

Parameters

- `event`: Trace event.

Return void.

RecordEvent

void `cnstream::PipelineTracer::RecordEvent(const TraceEvent &event)`

Records trace event.

Parameters

- `event`: Trace event.

Return void.

GetTrace

PipelineTrace `cnstream::PipelineTracer::GetTrace(const Time &start, const Time &end) const`

Gets trace data of pipeline for a specified period of time.

Parameters
- `start`: Start time.
- `end`: End time.

Return Returns trace data of pipeline.

GetTraceBefore

PipelineTrace `cnstream::PipelineTracer::GetTraceBefore(const Time &end, const Duration &duration) const`

Gets trace data of pipeline for a specified period of time.

Parameters
- `end`: End time
- `duration`: Length of time before `end`.

Return Returns trace data of pipeline.

GetTraceAfter

PipelineTrace `cnstream::PipelineTracer::GetTraceAfter(const Time &start, const Duration &duration) const`

Gets trace data of pipeline for a specified period of time.

Parameters
- `start`: Start time.
- `duration`: Length of time after `start`.

Return Returns trace data of pipeline.

3.7.4 ProcessProfiler

ProcessProfiler

`cnstream::ProcessProfiler::ProcessProfiler(const ProfilerConfig &config, const std::string &process_name, PipelineTracer *tracer)`

Constructor of ProcessProfiler.

Parameters
- `config`: Profiler config.
- `process_name`: The name of a process.
- `tracer`: The tracer.

SetModuleName

ProcessProfiler &`cnstream::`*`ProcessProfiler`*`::`**`SetModuleName`**`(const` std::string &module_name)

Set the module name to identify which module this profiler belongs to. The module name takes effect when trace level is TraceEvent::MODULE. Trace level can be set by SetTraceLevel.

Parameters

- `module_name`: The name of module.

Return Returns this profiler itself.

SetTraceLevel

ProcessProfiler &`cnstream::`*`ProcessProfiler`*`::`**`SetTraceLevel`**`(const` TraceEvent::Level &level)

Set the trace level for this profiler. Trace level identifies whether this profiler belongs to a module or a pipeline.

Parameters

- `level`: Trace level.

Return Returns this profiler itself.

See TraceEvent::Level.

RecordStart

void `cnstream::`*`ProcessProfiler`*`::`**`RecordStart`**`(const` RecordKey &key)

Records process start.

Parameters

- `key`: Unique identifier of a CNFrameInfo instance.

Return void.

See RecordKey.

RecordEnd

void `cnstream::`*`ProcessProfiler`*`::`**`RecordEnd`**`(const` RecordKey &key)

Records process end.

Parameters

- `key`: Unique identifier of a CNFrameInfo instance.

Return void.

See RecordKey.

GetName

std::string `cnstream::`*`ProcessProfiler`*`::`**`GetName`**`() const`

Gets process name set by constructor.

Return The name of process set by constructor.

GetProfile

ProcessProfile `cnstream::`*`ProcessProfiler`*`::`**`GetProfile`**`()`

Gets profiling results of the whole run time.

Return Returns the profiling results.

GetProfile

ProcessProfile `cnstream::`*`ProcessProfiler`*`::`**`GetProfile`**`(const` ProcessTrace &trace) `const`

Gets profiling results according to the trace datas.

Parameters

- `trace`: Trace datas.

Return Returns the profiling results.

OnStreamEos

void `cnstream::`*`ProcessProfiler`*`::`**`OnStreamEos`**`(const` std::string &stream_name)

Tells the profiler to clear datas of stream named by `stream_name`.

Parameters

- `stream_name`: Stream name. Usually it is comes from `CNFrameInfo::stream_id`.

Return void.

3.7.5 StreamProfiler

StreamProfiler

`cnstream::`*`StreamProfiler`*`::`**`StreamProfiler`**`(const` std::string &stream_name)

StreamProfiler constructor.

Parameters

- `stream_name`: Stream name.

AddLatency

StreamProfiler &`cnstream::`*`StreamProfiler`*`::`**`AddLatency`**`(const` Duration &latency`)`

Accumulate latency data.

Parameters

- `latency`: Latency.

Return Returns a lvalue reference to the current instance.

UpdatePhysicalTime

StreamProfiler &`cnstream::`*`StreamProfiler`*`::`**`UpdatePhysicalTime`**`(const` Duration &time`)`

Update pyhsical time this stream used.

Parameters

- `time`: The pyhsical time this stream used.

Return Returns a lvalue reference to the current instance.

AddDropped

StreamProfiler &`cnstream::`*`StreamProfiler`*`::`**`AddDropped`**`(`uint64_t dropped`)`

Accumulate drop frame count.

Parameters

- `dropped`: drop frame count.

Return Returns a lvalue reference to the current instance.

AddCompleted

StreamProfiler &`cnstream::`*`StreamProfiler`*`::`**`AddCompleted`**`()`

Accumulate completed frame count with 1.

Return Returns a lvalue reference to the current instance.

GetName

std::string `cnstream::`*`StreamProfiler`*`::`**`GetName`**`() const`

Gets stream name.

Return Returns stream name.

GetProfile

StreamProfile `cnstream::StreamProfiler::GetProfile()`

Gets statistical performance data for this stream.

Return Returns statistical performance data for this stream.

3.7.6 TraceSerializeHelper

DeserializeFromJSONStr

bool `cnstream::TraceSerializeHelper::DeserializeFromJSONStr(const` std::string &jsonstr, TraceSerializeHelper *pout)

Deserialize from json string.

Parameters

- `jsonstr`: Json string.
- `pout`: Output pointer.

Return True for deserialized successfully. False for deserialized failed.

DeserializeFromJSONFile

bool `cnstream::TraceSerializeHelper::DeserializeFromJSONFile(const` std::string &filename, TraceSerializeHelper *pout)

Deserialize from json file.

Parameters

- `jsonstr`: Json file path.
- `pout`: Output pointer.

Return True for deserialized successfully. False for deserialized failed.

TraceSerializeHelper

`cnstream::TraceSerializeHelper::TraceSerializeHelper()`

TraceSerializeHelper constructor.

TraceSerializeHelper

`cnstream::TraceSerializeHelper::TraceSerializeHelper(`TraceSerializeHelper &&t)

TraceSerializeHelper move constructor.

Parameters

- `t`: which instance move from.

TraceSerializeHelper

`cnstream::`*`TraceSerializeHelper`*`::`**`TraceSerializeHelper`**`(const` TraceSerializeHelper &t)

TraceSerializeHelper copy constructor.

Parameters

- `t`: which instance copy from.

operator=

TraceSerializeHelper &`cnstream::`*`TraceSerializeHelper`*`::`**`operator=`**(TraceSerializeHelper &&t)

TraceSerializeHelper operator =.

Parameters

- `t`: Which instance move from.

Return Returns a lvalue reference to the current instance.

operator=

TraceSerializeHelper &`cnstream::`*`TraceSerializeHelper`*`::`**`operator=`**`(const` TraceSerializeHelper &t)

TraceSerializeHelper operator =.

Parameters

- `t`: Which instance copy from.

Return Returns a lvalue reference to the current instance.

Serialize

void `cnstream::`*`TraceSerializeHelper`*`::`**`Serialize`**`(const` PipelineTrace &pipeline_trace)

Serialize trace data.

Parameters

- `pipeline_trace`: Trace data, you can get it by pipeline.GetTracer()->GetTrace().

Merge

void `cnstream::`*`TraceSerializeHelper`*`::`**`Merge`**`(const` TraceSerializeHelper &t)

Merge a trace serialize helper tool’ s data.

Parameters

- `t`: the trace serialize helper tool to be merged.

ToJsonStr

std::string `cnstream::`*`TraceSerializeHelper`*`::`**`ToJsonStr`**`() const`

Serialize to json string.

Return Return a json string.

ToFile

bool `cnstream::`*`TraceSerializeHelper`*`::`**`ToFile`**`(const` std::string &filename`) const`

Serialize to json file.

Parameters

- `filename`: Json file name.

Return True for success, false for failed(The possible reason is that there is no write file permission).

Reset

void `cnstream::`*`TraceSerializeHelper`*`::`**`Reset`**`()`

Reset serialize helper. Clear datas and free up memory.

3.8 RTSP Sink

3.8.1 CheckParamSet

bool `cnstream::`*`RtspSink`*`::`**`CheckParamSet`**`(const` ModuleParamSet &paramSet`) const override`

Checks ParamSet for a module.

Parameters

- `paramSet`: Parameters for this module.

Return Returns true if this API run successfully. Otherwise, returns false.

3.8.2 Close

void `cnstream::`*`RtspSink`*`::`**`Close`**`() override`

Called by pipeline when pipeline stopped.

3.8.3 Open

bool `cnstream::`*`RtspSink`*`::`**`Open`**`(`ModuleParamSet paramSet`)` **`override`**

Called by pipeline when pipeline start.

Parameters

- `paramSet`: : The parameter set.

Return Returns ture if module open succeeded, otherwise returns false.

3.8.4 Process

int `cnstream::`*`RtspSink`*`::`**`Process`**`(`CNFrameInfoPtr data`)` **`override`**

Encode each frame.

Parameters

- `data`: : Data to be processed.

Return Value

- `0`: Succeeded and did not intercept data.
- `<0`: Failed.

Return Whether the process is succeeded.

3.9 Syncmem

3.9.1 CNSyncedMemory

`cnstream::`*`CNSyncedMemory`*`::`**`CNSyncedMemory`**`(`size_t size`)`

Constructor.

Parameters

- `size`: The size of the memory.

3.9.2 CNSyncedMemory

`cnstream::`*`CNSyncedMemory`*`::`**`CNSyncedMemory`**`(`size_t size, int mlu_dev_id, int mlu_ddr_chn = -1`)`

Constructor.

Parameters

- `size`: The size of the memory.
- `mlu_dev_id`: MLU device ID that is incremented from 0.
- `mlu_ddr_chn`: The MLU DDR channel that is greater than or equal to 0, and is less than 4. It specifies which piece of DDR channel the memory allocated on.

3.9.3 GetCpuData

`const void *cnstream::CNSyncedMemory::GetCpuData()`

Gets the CPU data.

Return Returns the CPU data pointer.

Note If the size is 0, nullptr is always returned.

3.9.4 GetHead

`SyncedHead cnstream::CNSyncedMemory::GetHead() const`

Gets synchronized head.

Return Returns synchronized head.

3.9.5 GetMluData

`const void *cnstream::CNSyncedMemory::GetMluData()`

Gets the MLU data.

Return Returns the MLU data pointer.

Note If the size is 0, nullptr is always returned.

3.9.6 GetMluDdrChnId

`int cnstream::CNSyncedMemory::GetMluDdrChnId() const`

Gets the channel ID of the MLU DDR.

Return Returns the DDR channel ID that the MLU memory allocated on.

3.9.7 GetMluDevId

`int cnstream::CNSyncedMemory::GetMluDevId() const`

Gets the MLU device ID.

Return Returns the device that the MLU memory allocated on.

3.9.8 GetMutableCpuData

`void *cnstream::CNSyncedMemory::GetMutableCpuData()`

Gets the mutable CPU data.

Return Returns the CPU data pointer.

3.9.9 GetMutableMluData

void *`cnstream::`*`CNSyncedMemory`*`::`**`GetMutableMluData`**`()`

Gets the mutable MLU data.

Return Returns the MLU data pointer.

3.9.10 GetSize

size_t `cnstream::`*`CNSyncedMemory`*`::`**`GetSize`**`()` **`const`**

Gets data bytes.

Return Returns data bytes.

3.9.11 SetCpuData

void `cnstream::`*`CNSyncedMemory`*`::`**`SetCpuData`**(void *data)

Sets the CPU data.

Parameters

- `data`: The data pointer on CPU.

Return Void.

3.9.12 SetMluData

void `cnstream::`*`CNSyncedMemory`*`::`**`SetMluData`**(void *data)

Sets the MLU data.

Parameters

- `data`: The data pointer on MLU.

3.9.13 SetMluDevContext

void `cnstream::`*`CNSyncedMemory`*`::`**`SetMluDevContext`**(int dev_id, int ddr_chn = -1)

Sets the MLU device context.

Parameters

- `dev_id`: The MLU device ID that is incremented from 0.
- `ddr_chn`: The MLU DDR channel ID that is greater than or equal to 0, and less than a. It specifies which piece of DDR channel the memory allocated on.

Note You need to call this API before all getters and setters.

3.9.14 ToCpu

void `cnstream::`*`CNSyncedMemory`*`::`**`ToCpu`**`()`

Synchronizes the memory data to CPU.

3.9.15 ToMlu

void `cnstream::`*`CNSyncedMemory`*`::`**`ToMlu`**`()`

Synchronizes the memory data to MLU.

3.10 Tracker

3.10.1 CheckParamSet

bool `cnstream::`*`Tracker`*`::`**`CheckParamSet`**`(const` ModuleParamSet &paramSet) `const override`

Checks parameters for a module.

Parameters

- `paramSet`: Parameters for this module.

Return Returns true if this function has run successfully. Otherwise, returns false.

3.10.2 Close

void `cnstream::`*`Tracker`*`::`**`Close`**`() override`

Called by pipeline when pipeline is stopped.

Return None.

3.10.3 Open

bool `cnstream::`*`Tracker`*`::`**`Open`**(ModuleParamSet paramSet) `override`

Called by pipeline when pipeline is started.

Parameters

- `paramSet`:
  - track_name: Optional. Class name for track. It is “FeatureMatch” by default.
  - model_path: Optional. The path of the offline model.
  - func_name: Optional. The function name defined in the offline model. It can be found in the Cambricon twins description file. It is “subnet0” for the most cases.

Return Returns true if the module has been opened successfully.

3.10.4 Process

int `cnstream::`*`Tracker`*`::Process`(std::shared_ptr<CNFrameInfo> data) **`override`**

Processes each frame.

Parameters

- `data`: : Pointer to the frame information.

Return Value

- `0`: The process has run successfully and has no intercepted data.
- `<0`: The process is failed.

Return Whether the process succeed.

4 Release Notes

This release notes outlines CNStream API updates and documentation updates in CNStream Developer Guide.

4.1 CNStream Release 2021-01-25 (Version 5.3.0)

4.1.1 API Updates

This section lists API functions and fields that were added, changed, or removed.

- Changes on the Frame and FrameVa frameworks are as follows:
  - Add Parameter `CN_FRAME_FLAG_REMOVED` to `CNFrameFlag` enum for identifying the stream to which the frame belongs is removed.
  - Changed the struct `CNFrameInfo` to a class and privately inherits from class NonCopyable.
  - Added the new `payload` parameter to the `Create` API, the default value of which is `nullptr`.
  - Added the new `IsRemoved` API for checking whether the stream to which the frame belongs is removed.
  - Changed the struct `CNDataFrame` to a class and privately inherits from class NonCopyable.
  - Added the new `dst_mlu` parameter to the `CopyToSyncMem` API, the default value of which is `true`.
  - Added the new struct `CNInferObjs` for holding objects inference result.
  - Added the new struct `InferData` contains the inputs, the outputs and the information of inference.
  - Added the new struct `CNInferData` for holding all `InferData` of one frame.
  - Added the new `GetCNDataFramePtr` API for getting the `CNDataFramePtr` object of one frame.
  - Added the new `GetCNInferObjsPtr` API for getting the `CNInferObjsPtr` object of one frame.
  - Added the new `GetCNInferDataPtr` API for getting the `CNInferDataPtr` object of one frame.
- Changes on the Module framework are as follows:
  - Added the new virtual `OnEos` API to notify the module that the EOS is arrived.
  - Added the new `GetContainer` API to get the container of the module.
  - Added the new `GetProfiler` API to get the profiler of the module.
  - Removed the `RecordTime` API due to the PerfManager has been replaced to Profiler.
  - Removed the `GetPerfManager` API due to the PerfManager has been replaced to Profiler.
- Changes on the Pipeline framework are as follows:
  - Added the new `GetName` API to get the name of the pipeline.
  - Added the new `profiler_config` parameter to the `BuildPipeline` API, the default value of which is a `ProfilerConfig` object created by `ProfilerConfig` constructor.
  - The following APIs are removed due to the PerfManager has been replaced by Profiler:
    - The `CreatePerfManager` API.

    - The `RemovePerfManager` API.
    - The `AddPerfManager` API.
    - The `PerfSqlCommitLoop` API.
    - The `CalculatePerfStats` API.
    - The `CalculateModulePerfStats` API.
    - The `CalculatePipelinePerfStats` API.
    - The `GetPerfManagers` API.
  - Added the new `IsProfilingEnabled` API to check if profiling function is enabled.
  - Added the new `IsTracingEnabled` API to check if tracing function is enabled.
  - Added the new `GetProfiler` API to get the profiler.
  - Added the new `GetTracer` API to get the tracer.
  - Added the new `IsRootNode` API to check if the module is the root node of the pipeline.
  - Added the new `IsLeafNode` API to check if the module is the leaf node of the pipeline.
- Supported the Profiler with the related APIs.
- Replaced the PerfManager and PerfCalculator by Profiler.
- Changes on the SyncMem are as follows:
  - Removed the `CNStreamMallocHost` API.
  - Removed the `CNSyncedMemory` constructor.
  - Set the parameter `mlu_ddr_chn` with default value `-1` of the `CNSyncedMemory` constructor.
  - Changed the default value of parameter `mlu_ddr_chn` of the `SetMluDevContext` API, from `0` to `-1`.
  - Removed the `SetMluCpuData` API which is used on MLU220_SOC platform.
- Supported the Inferencer2 module with the related APIs.
- Changes on the DataSource module are as follows:
  - Changes on the `RawImgMemHandler` class are as follows:
    - Removed the `Write` API with one parameter `cv::Mat* mat_data`.
    - Removed the `Write` API with five parameters `unsigned char *data, int size, int width = 0, int height = 0, CNDataFormat pixel_fmt = CN_INVALID`.
    - Changed the parameters from `cv::Mat* mat_data, uint64_t pts` to `const cv::Mat* mat_data, const uint64_t pts` of the `Write` API.
    - Changed the parameters from `unsigned char *data, int size, uint64_t pts, int width = 0, int height = 0, CNDataFormat pixel_fmt = CN_INVALID` to `const uint8_t *data, const int size, const uint64_t pts, const int width = 0, const int height = 0, const CNDataFormat pixel_fmt = CN_INVALID` of the `Write` API.
  - Removed the `UsbHandler` class.

4.2 CNStream Release 2020-09-18 (Version 5.2.0)

4.2.1 API Updates

This section lists API functions and fields that were added, changed, or removed.

- Changes on the FrameVa are as follows:
  - Added the new `HasBGRImage` API for checking whether data frame is converted to BGR format and saved to CV format.
  - Added the new `RemoveExtraAttribute` API for removing an attribute by key.
  - Added the new `GetExtraAttributes` API for retrieving all extended attributes of an object.

  - Added the new `GetFeature` API for retrieving the feature of an object by key.
  - Added the new `key` parameter to the `AddFeature` API.
  - Renamed the `AddExtraAttribute` to `AddExtraAttributes`.
  - Changed the return type of the `AddFeature` API from `void` to `bool`.
  - Changed the return type of the `GetFeatures` API from `ThreadSafeVector<CNInferFeature>` to `CNInferFeatures`.
  - Added the new `CNInferFeatures` type.
  - Added the new `StringPairs` type.
  - Changed the struct `CNInferFeature` to `vector<float>` type.
  - Changed the type of variable `datas` in struct `CNInferObject` from `ThreadSafeUnorderedMap<int, any>` to `std::unordered_map<int, any>`.
- Changes on the Frame framework are as follows:
  - Changed the type of variable `datas` in struct `CNFrameInfo` from `ThreadSafeUnorderedMap<int, any>` to `std::unordered_map<int, any>`.
- Changes on the Pipeline framework are as follows:
  - Added the new `GetEndModule` API for retrieving the end module of a pipeline.
- Changes on the PerfCalculator are as follows:
  - Added the new `total_time` variable in struct `PerfStats`.
- Changes on the PerfManager are as follows:
  - Added the new `CreateDir` API for creating directory.

4.3 CNStream Release 2020-07-10 (Version 5.0.0)

4.3.1 API Updates

This section lists API functions and fields that were added, changed, or removed.

- Changes on the DataSource module are as follows:
  - The following new data types are supported:
    * Added the new `ESPacket` struct.
    * Added the new `FileHandler` class.
    * Added the new `RtspHandler` class.
    * Added the new `ESMemHandler` class.
  - Parameter changes in `DataSourceParam` struct.
  - The following data type and API are removed due to function changes:
    * The `SourceType` enum.
    * The `CreateSource` API.
- Changes on the EventBus framework are as follows:
  - The `cnstream_eventbus.hpp` file is moved from the `modules/core/include` directory to the `framework/core/include` directory.
  - Added the new `Start` and `Stop` APIs to support starting and stopping an event bus thread.
  - Parameter changes in `Event` struct.
  - Removed the `module` parameter from the BusWatcher API.
  - Removed the `watch_module` parameter from the `AddBusWatch` API.
  - Removed `EventType` enum due to function changes.
- Changes on the Frame framework are as follows:

  - The `cnstream_frame.hpp` file is moved from the `modules/core/include` directory to the `framework/core/include` directory.
  - Added the new `IsEos` API to check if this is an eos frame.
  - Added the new `SetStreamIndex` API to support setting stream index.
  - Parameter changes in `CNFrameInfo` struct.
  - The following enums, structs, classes, and APIs are moved from the `cnstream_frame.hpp` file to the `cnstream_frame_va.hpp` file:
    * The `CNDataFormat` enum.
    * The `DevContext` struct.
    * The `MemMapType` enum.
    * The `CNGetPlanes` API.
    * The `IDataDeallocator` class.
    * The `ICNMediaImageMapper` class.
    * The `CNDataFrame` struct.
    * The `CNInferBoundingBox` struct.
    * The `CNInferAttr` struct.
    * The `CNInferFeature` struct.
    * The `CNInferObject` struct.
  - Added the new `stream_id` parameter to the `MmapSharedMem`, `CopyToSharedMem`, and `ReleaseSharedMem` APIs.
  - Parameter changed in `CNDataFrame` struct.
  - Changed `CNInferFeature` from a type to struct.
  - Changed return value type of `GetFeatures` API.
- Changes on the Module framework are as follows:
  - The `cnstream_module.hpp` file is moved from the `modules/core/include` directory to the `framework/core/include` directory.
  - Added the new `IModuleObserver` class to support observing modules.
  - Added the new `SetObserver`, `ParseByJSONStr`, `ParseByJSONFile`, and `ConfigsFromJsonFile` APIs.
  - Removed `SetPerfManagers` and `ClearPerfManagers` APIs due to function changes.
  - The following enums, structs, classes, and APIs are moved from the `cnstream_module.hpp` file to the `cnstream_config.hpp` file:
    * The `ParamRegister` class.
    * The `ParametersChecker` class.
    * The `ModuleParamSet` struct.
    * The `GetPathRelativeToTheJSONFile` API.
    * The `Register` API.
    * The `GetParams` API.
    * The `IsRegisted` API.
    * The `SetModuleDesc` API.
- Changes on the Pipeline framework are as follows:
  - The `cnstream_pipeline.hpp` file is moved from the `modules/core/include` directory to the `framework/core/include` directory.
  - Added the new `IdxManager` class to support managing stream index.
  - Added the new `final_print` parameter to the `CalculateModulePerfStats` and `CalculatePipelinePerfStats` APIs.
  - Parameters are changed in `StreamMsg` struct.

  - Removed the following APIs due to the function changes:
    * The `Open` API.
    * The `Close` API.
    * The `Process` API.
    * The `GetLinkIds` API.
    * The `GetModuleParallelism` API.
    * The `NotifyStreamMsg` API.
  - Moved the `CNModuleConfig` struct from the `cnstream_pipeline.hpp` file to the `cnstream_config.hpp` file.
- Changes on the PerfManager are as follows:
  - The `perf_manager.hpp` file is moved from the `modules/core/include` directory to the `framework/core/include` directory.
  - The following new APIs are supported:
    * Added the new `GetSql` API to support getting SQL handler.
    * Added the new `GetKeys` API to support generating keys.
    * Added the new `GetEndTimeSuffix` API to support getting the end time suffix.
    * Added the new `GetStartTimeSuffix` API to support getting the start time suffix.
    * Added the new `GetPrimaryKey` API to support getting the default primary key.
    * Added the new `GetDefaultType` API to support getting the default perf type.
  - Removed the following data types and APIs due to the function changes:
    * The `PerfInfo` struct.
    * The `Init` API that contains the `db_name`, `module_names`, `start_node` and `end_nodes` parameters.
    * The `RegisterPerfType` API that contains the `type` parameter.
    * The `CalculatePipelinePerfStats` API.
    * The `GetCalculator` API.
    * The `SetModuleNames` API.
    * The `SetStartNode` API.
    * The `SetEndNodes` API.
    * All `CreatePerfCalculator` APIs.
    * All `CalculatePerfStats` APIs.
    * All `CalculateThroughput` APIs.
- Changes on the PerfCalculator are as follows:
  - The `perf_calculator.hpp` file is moved from the `modules/core/include` directory to the `framework/core/include` directory.
  - The following new data types, classes, APIs are supported:
    * The `PerfCalculatorForModule`, `PerfCalculatorForPipeline`, and `PerfCalculatorForInfer` classes, which inherits from `PerfCalculator` class.
    * The `PerfCalculationMethod` class.
    * The `PerfUtils` class.
    * The `PrintStreamId` API to print stream id.
    * The `PrintStr` API to print string.
    * The `PrintTitle` API to print title.
    * The `PrintTitleForLatestThroughput` API to print title for latest throughput.
    * The `PrintTitleForAverageThroughput` API to print title for average throughput.
    * The `PrintTitleForTotal` API to print ‘total’.
    * The `SetPerfUtils` API to set the PerfUtils for getting data from database.

    * The `GetPerfUtils` API to get the PerfUtils.
    * The `CalcAvgThroughput` API to calculate average throughput.
    * The `GetAvgThroughput` API to get average throughput.
    * The `CalculateFinalThroughput` API to calculate final throughput.
    * The virtual `CalcLatency` API to calculate latency.
    * The virtual `CalcThroughput` API to calculate throughput.
    * The `SetPrintThroughput` API to set whether print throughput inside perf calculator.
  - Added the new width parameter to the PrintLatency API.
  - Added the new width parameter to the PrintThroughput API.
  - Added the new sql_name and perf_type parameters to the GetLatency API.
  - Added the new sql_name and perf_type parameters to the GetThroughput API.
  - Parameter changes in `PerfStats` struct.
  - Changed the return type of the `GetThroughput` API from `PerfStats` to `std::vector<PerfStats>`.
  - The following APIs are removed due to function changes:
    * The `PrintPerfStats` API.
    * The `CalcLatency` API.
    * The `CalcThroughputByTotalTime` API.
    * The `CalcThroughputByEachFrameTime` API.
    * The `SearchFromDatabase` API.

4.4 CNStream Release 2020-05-25 (Version 4.5.0)

4.4.1 API Updates

This section lists API functions and fields that were added, changed, or removed.

- The following API is supported in the Frame framework:
  - Added the new `CopyToSyncMemOnDevice` API to synchronize source data to a specified device.
- The following APIs are supported in the Module framework:
  - Added the new `ClearPerfManagers` API to clear all performance managers.
- Supported the RtspSink module with the related APIs.

4.5 Release 2020-04-16 (Version 4.4.0)

4.5.1 API Updates

This section lists API functions and fields that were added, changed, or removed.

- The following APIs are supported in Frame framework for multi-process function:
  - Added the new `MmapSharedMem` API to map shared memory.
  - Added the new `UnMapSharedMem` API to unmap shared memory.
  - Added the new `CopyToSharedMem` API to copy source-data to shared memory.
  - Added the new `ReleaseSharedMem` API to release shared memory.
- The following APIs are supported in Module framework for the performance measurement function:
  - Added the new `SetPerfManagers` API to set PerfManagers.

  - Added the new `GetPerfManager` API to retrieve PerfManager by stream id.
  - Added the new `ClearPerfManagers` API to clear PerfManagers.
- The following APIs are supported in Pipeline framework for the performance measurement function:
  - Added the new `CreatePerfManager` API to create PerfManager for each stream to measure performance of modules and pipeline.
  - Added the new `PerfSqlCommitLoop` API to commit sqlite events to increase the speed of inserting data to the database.
  - Added the new `CalculatePerfStats` API to calculate performance of modules and pipeline, and print performance statistics.
  - Added the new `CalculateModulePerfStats` API to calculate performance of modules, and print performance statistics.
  - Added the new `CalculatePipelinePerfStats` API to calculate performance of pipeline, and print performance statistics.
  - Removed the `PrintPerformanceInformation` API due to function changes.

4.5.2 Doc Updates

This section lists the documentation updates that were made in this version:

- Optimized the description of the APIs.
- Added the missing description of APIs and data types.

4.6 Release 2020-02-24

4.6.1 API Updates

This section lists API functions and fields that were added, changed, or removed.

- The following APIs are supported in Frame framework:
  - Supported the new virtual `GetMediaImage` API.
  - Supported the new virtual `GetPitch` API.
  - Supported the new virtual `GetCpuAddress` API.
  - Supported the new virtual `GetDevAddress` API.
  - Supported the new virtual `~ICNMediaImageMapper` API.
  - Parameter changes in `DevContext` struct.
- The following APIs are supported in SyncMem:
  - Supported the new `SetMluCpuData` API to set the CPU and MLU data for MLU220SOC only.
  - Supported the new `mlu_data` and `cpu_data` parameters to the `SetMluCpuData` API.

4.7 Release 2019-12-31

4.7.1 API Updates

This section lists API functions and fields that were added, changed, or removed.

- The following APIs are supported in Module framework:
  - Supported the new `IsRegisted` API for checking if a module parameter is registered or not.
  - Supported the new `SetModuleDesc` API for setting module description.
  - Supported the new `GetModuleDesc` API for getting module description.
  - Supported the new `CheckParamSet` API for checking ParamSet in a module.
  - Supported the new `GetRegisted` API for getting all registered modules name.
  - Supported the new `CheckPath` API for checking path of a configuration file.
  - Supported the new `IsNum` API for checking if a parameter is a number.
- The following APIs are supported in Inferencer module:
  - Supported the new `CheckParamSet` API for checking ParamSet in Inferencer module.
- The following APIs are supported in DataSource module:
  - Supported the new `CheckParamSet` API for checking ParamSet in DataSource module.
- The following APIs are supported in Tracker module:
  - Supported the new `CheckParamSet` API for checking ParamSet in Tracker module.