holoscan.operators

This module provides a Python API to underlying C++ API Operators.

`holoscan.operators.AJASourceOp`	Operator to get a video stream from an AJA capture card.
`holoscan.operators.BayerDemosaicOp`	Bayer Demosaic operator.
`holoscan.operators.FormatConverterOp`	Format conversion operator.
`holoscan.operators.GXFCodeletOp`(fragment, ...)	GXF Codelet wrapper operator.

`holoscan.operators.HolovizOp`(fragment, *args)	Holoviz visualization operator using Holoviz module.
`holoscan.operators.InferenceOp`	Inference operator.
`holoscan.operators.InferenceProcessorOp`	Holoinfer Processing operator.
`holoscan.operators.PingRxOp`(fragment, *args, ...)	Simple receiver operator.

`holoscan.operators.PingTxOp` (fragment, *args, ...)	Simple transmitter operator.
`holoscan.operators.SegmentationPostprocessorOp`	Operator carrying out post-processing operations on segmentation outputs.
`holoscan.operators.V4L2VideoCaptureOp`	Operator to get a video stream from a V4L2 source.
`holoscan.operators.VideoStreamRecorderOp`	Operator class to record a video stream to a file.

holoscan.operators.VideoStreamReplayerOp	Operator class to replay a video stream from a file.

class holoscan.operators.AJASourceOp

Bases: holoscan.core._core.Operator

Operator to get a video stream from an AJA capture card.

==Named Inputs==

overlay_buffer_inputnvidia::gxf::VideoBuffer (optional)

The operator does not require a message on this input port in order for compute to be called. If a message is found, and enable_overlay is True, the image will be mixed with the image captured by the AJA card. If enable_overlay is False, any message on this port will be ignored.

==Named Outputs==

video_buffer_outputnvidia::gxf::VideoBuffer

The output video frame from the AJA capture card. If overlay_rdma is True, this video buffer will be on the device, otherwise it will be in pinned host memory.

overlay_buffer_outputnvidia::gxf::VideoBuffer (optional)

This output port will only emit a video buffer when enable_overlay is True. If overlay_rdma is True, this video buffer will be on the device, otherwise it will be in pinned host memory.

Parameters

fragment

The fragment that the operator belongs to.

device

The device to target (e.g., “0” for device 0). Default value is `"0"`.

channel

The camera `NTV2Channel` to use for output (e.g., `NTV2Channel.NTV2_CHANNEL1` (`0`) or “NTV2_CHANNEL1” (in YAML) for the first channel). Default value is `NTV2Channel.NTV2_CHANNEL1` (`"NTV2_CHANNEL1"` in YAML).

width

Width of the video stream. Default value is `1920`.

height

Height of the video stream. Default value is `1080`.

framerate

Frame rate of the video stream. Default value is `60`.

rdma

Boolean indicating whether RDMA is enabled. Default value is `False` (`"false"` in YAML).

enable_overlay

Boolean indicating whether a separate overlay channel is enabled. Default value is `False` (`"false"` in YAML).

overlay_channel

The camera NTV2Channel to use for overlay output. Default value is `NTV2Channel.NTV2_CHANNEL2` (`"NTV2_CHANNEL2"` in YAML).

overlay_rdma

Boolean indicating whether RDMA is enabled for the overlay. Default value is `False` (`"false"` in YAML).

name

The name of the operator. Default value is `"aja_source"`.

Attributes

`args`	The list of arguments associated with the component.
`conditions`	Conditions associated with the operator.
`description`	YAML formatted string describing the operator.
`fragment`	The fragment (`holoscan.core.Fragment`) that the operator belongs to.
`id`	The identifier of the component.
`name`	The name of the operator.
`operator_type`	The operator type.
`resources`	Resources associated with the operator.
`spec`	The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

Methods

`add_arg`(*args, **kwargs)	Overloaded function.
`compute`(self, arg0, arg1, arg2)	Operator compute method.
`initialize`(self)	Initialize the operator.
`setup`(self, spec)	Define the operator specification.
`start`(self)	Operator start method.
`stop`(self)	Operator stop method.

OperatorType	

class OperatorType

Bases: `pybind11_builtins.pybind11_object`

Enum class for operator types used by the executor.

- NATIVE: Native operator.
- GXF: GXF operator.

- VIRTUAL: Virtual operator. (for internal use, not intended for use by application authors)

Members:

NATIVE

GXF

VIRTUAL

Attributes

name	

value	

GXF = *<OperatorType.GXF: 1>*

NATIVE = *<OperatorType.NATIVE: 0>*

VIRTUAL = *<OperatorType.VIRTUAL: 2>*

__init__(*self: holoscan.core._core.Operator.OperatorType*, *value: int*)   None

property name

property value

__init__(*self: holoscan.operators.aja_source._aja_source.AJASourceOp*, *fragment: holoscan.core._core.Fragment*, **args*, *device: str = '0'*, *channel: holoscan.operators.aja_source._aja_source.NTV2Channel = <NTV2Channel.NTV2_CHANNEL1: 0>*, *width: int = 1920*, *height: int = 1080*, *framerate: int = 60*, *rdma: bool = False*, *enable_overlay: bool = False*, *overlay_channel: holoscan.operators.aja_source._aja_source.NTV2Channel = <NTV2Channel.NTV2_CHANNEL2: 1>*, *overlay_rdma: bool = True*, *name: str = 'aja_source'*)   None

Operator to get a video stream from an AJA capture card.

==Named Inputs==

overlay_buffer_inputnvidia::gxf::VideoBuffer (optional)

The operator does not require a message on this input port in order for `compute` to be called. If a message is found, and `enable_overlay` is `True`, the image will be mixed with the image captured by the AJA card. If `enable_overlay` is `False`, any message on this port will be ignored.

==Named Outputs==

video_buffer_outputnvidia::gxf::VideoBuffer

The output video frame from the AJA capture card. If `overlay_rdma` is `True`, this video buffer will be on the device, otherwise it will be in pinned host memory.

overlay_buffer_outputnvidia::gxf::VideoBuffer (optional)

This output port will only emit a video buffer when `enable_overlay` is `True`. If `overlay_rdma` is `True`, this video buffer will be on the device, otherwise it will be in pinned host memory.

Parameters

fragment

The fragment that the operator belongs to.

device

The device to target (e.g., “0” for device 0). Default value is `"0"`.

channel

The camera `NTV2Channel` to use for output (e.g., `NTV2Channel.NTV2_CHANNEL1` (`0`) or “NTV2_CHANNEL1” (in YAML) for the first channel). Default value is `NTV2Channel.NTV2_CHANNEL1` (`"NTV2_CHANNEL1"` in YAML).

width

Width of the video stream. Default value is `1920`.

height

Height of the video stream. Default value is `1080`.

framerate

Frame rate of the video stream. Default value is `60`.

rdma

Boolean indicating whether RDMA is enabled. Default value is `False` (`"false"` in YAML).

enable_overlay

Boolean indicating whether a separate overlay channel is enabled. Default value is `False` (`"false"` in YAML).

overlay_channel

The camera NTV2Channel to use for overlay output. Default value is `NTV2Channel.NTV2_CHANNEL2` (`"NTV2_CHANNEL2"` in YAML).

overlay_rdma

Boolean indicating whether RDMA is enabled for the overlay. Default value is `False` (`"false"` in YAML).

name

The name of the operator. Default value is `"aja_source"`.

add_arg(**args*, ***kwargs*)

Overloaded function.

1. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Arg) -> None

Add an argument to the component.

2. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.ArgList) -> None

Add a list of arguments to the component.

3. add_arg(self: holoscan.core._core.Operator, **kwargs) -> None

Add arguments to the component via Python kwargs.

4. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Condition) -> None

5. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Resource) -> None

Add a condition or resource to the Operator.

This can be used to add a condition or resource to an operator after it has already been constructed.

Parameters

arg

The condition or resource to add.

property args

The list of arguments associated with the component.

Returns

arglist

compute(*self: holoscan.core._core.Operator*, *arg0: holoscan.core._core.InputContext*, *arg1: holoscan.core._core.OutputContext*, *arg2: holoscan.core._core.ExecutionContext*)

None

Operator compute method. This method defines the primary computation to be executed by the operator.

property conditions

Conditions associated with the operator.

property description

YAML formatted string describing the operator.

property fragment

The fragment (`holoscan.core.Fragment`) that the operator belongs to.

property id

The identifier of the component.

The identifier is initially set to `-1`, and will become a valid value when the component is initialized.

With the default executor (*holoscan.gxf.GXFExecutor*), the identifier is set to the GXF component ID.

Returns

id

initialize(*self: holoscan.operators.aja_source._aja_source.AJASourceOp*) None

Initialize the operator.

This method is called only once when the operator is created for the first time, and uses a light-weight initialization.

property name

The name of the operator.

property operator_type

The operator type.

holoscan.core.Operator.OperatorType enum representing the type of the operator. The two types currently implemented are native and GXF.

property resources

Resources associated with the operator.

setup(*self: holoscan.operators.aja_source._aja_source.AJASourceOp*, *spec: holoscan.core._core.OperatorSpec*)   None

Define the operator specification.

Parameters

spec

The operator specification.

property spec

The operator spec (holoscan.core.OperatorSpec) associated with the operator.

start(*self: holoscan.core._core.Operator*)   None

Operator start method.

stop(*self: holoscan.core._core.Operator*)   None

Operator stop method.

class holoscan.operators.BayerDemosaicOp

Bases: holoscan.core._core.Operator

Bayer Demosaic operator.

==Named Inputs==

receivernvidia::gxf::Tensor or nvidia::gxf::VideoBuffer

The input video frame to process. If the input is a VideoBuffer it must be an 8-bit unsigned grayscale video (*nvidia::gxf::VideoFormat::GXF_VIDEO_FORMAT_GRAY*). If a video buffer is not found, the input port message is searched for a device tensor with the name specified by *in_tensor_name*. The tensor must have either 8-bit or 16-bit unsigned integer format. The tensor or video buffer may be in either host or device memory (a host->device copy is performed if needed).

==Named Outputs==

transmitternvidia::gxf::Tensor

The output video frame after demosaicing. This will be a 3-channel RGB image if `alpha_value` is `True`, otherwise it will be a 4-channel RGBA image. The data type will be either 8-bit or 16-bit unsigned integer (matching the bit depth of the input). The name of the tensor that is output is controlled by `out_tensor_name`.

==Device Memory Requirements==

When using this operator with a `holoscan.resources.BlockMemoryPool`, the minimum `block_size` is `(rows * columns * output_channels * element_size_bytes)` where `output_channels` is 4 when `generate_alpha` is `True` and 3 otherwise. If the input tensor or video buffer is already on the device, only a single memory block is needed. However, if the input is on the host, a second memory block will also be needed in order to make an internal copy of the input to the device. The memory buffer must be on device (`storage_type=1`).

Parameters

fragment

The fragment that the operator belongs to.

pool

Memory pool allocator used by the operator.

cuda_stream_pool

`holoscan.resources.CudaStreamPool` instance to allocate CUDA streams. Default value is `None`.

in_tensor_name

The name of the input tensor. Default value is `""` (empty string).

out_tensor_name

The name of the output tensor. Default value is `""` (empty string).

interpolation_mode

The interpolation model to be used for demosaicing. Values available at: https://docs.nvidia.com/cuda/npp/nppdefs.html?highlight=Two%20parameter%20cubic%20filter#c.NppiInterpolationMode

- NPPI_INTER_UNDEFINED (`0`): Undefined filtering interpolation mode.
- NPPI_INTER_NN (`1`): Nearest neighbor filtering.
- NPPI_INTER_LINEAR (`2`): Linear interpolation.
- NPPI_INTER_CUBIC (`4`): Cubic interpolation.
- NPPI_INTER_CUBIC2P_BSPLINE (`5`): Two-parameter cubic filter (B=1, C=0)
- NPPI_INTER_CUBIC2P_CATMULLROM (`6`): Two-parameter cubic filter (B=0, C=1/2)
- NPPI_INTER_CUBIC2P_B05C03 (`7`): Two-parameter cubic filter (B=1/2, C=3/10)
- NPPI_INTER_SUPER (`8`): Super sampling.
- NPPI_INTER_LANCZOS (`16`): Lanczos filtering.

- NPPI_INTER_LANCZOS3_ADVANCED (`17`): Generic Lanczos filtering with order 3.

- NPPI_SMOOTH_EDGE (`0x8000000`): Smooth edge filtering.

Default value is `0` (NPPI_INTER_UNDEFINED).

bayer_grid_pos

The Bayer grid position. Values available at: https://docs.nvidia.com/cuda/npp/nppdefs.html?highlight=Two%20parameter%20cubic%20filter#c.NppiBayerGridPosition

- NPPI_BAYER_BGGR (`0`): Default registration position BGGR.

- NPPI_BAYER_RGGB (`1`): Registration position RGGB.

- NPPI_BAYER_GBRG (`2`): Registration position GBRG.

- NPPI_BAYER_GRBG (`3`): Registration position GRBG.

Default value is `2` (NPPI_BAYER_GBRG).

generate_alpha

Generate alpha channel. Default value is `False`.

alpha_value

Alpha value to be generated if `generate_alpha` is set to `True`. Default value is `255`.

name

The name of the operator. Default value is `"bayer_demosaic"`.

Attributes

`args`	The list of arguments associated with the component.

`conditions`	Conditions associated with the operator.
`description`	YAML formatted string describing the operator.
`fragment`	The fragment (`holoscan.core.Fragment`) that the operator belongs to.
`id`	The identifier of the component.
`name`	The name of the operator.
`operator_type`	The operator type.
`resources`	Resources associated with the operator.
`spec`	The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

Methods

`add_arg` (*args, **kwargs)	Overloaded function.
`compute` (self, arg0, arg1, arg2)	Operator compute method.
`initialize`	Initialize the operator.

(self)	
`setup` (self, spec)	Define the operator specification.
`start` (self)	Operator start method.
`stop` (self)	Operator stop method.

OperatorType	

class OperatorType

Bases: `pybind11_builtins.pybind11_object`

Enum class for operator types used by the executor.

- NATIVE: Native operator.
- GXF: GXF operator.
- VIRTUAL: Virtual operator. (for internal use, not intended for use by application authors)

Members:

NATIVE

GXF

VIRTUAL

Attributes

name	

value	

GXF = *<OperatorType.GXF: 1>*

NATIVE = *<OperatorType.NATIVE: 0>*

VIRTUAL = *<OperatorType.VIRTUAL: 2>*

__init__(*self: holoscan.core._core.Operator.OperatorType*, *value: int*)   None

property name

property value

add_arg(**args*, ***kwargs*)

Overloaded function.

1. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Arg) -> None

Add an argument to the component.

2. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.ArgList) -> None

Add a list of arguments to the component.

3. add_arg(self: holoscan.core._core.Operator, **kwargs) -> None

Add arguments to the component via Python kwargs.

4. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Condition) -> None

5. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Resource) -> None

Add a condition or resource to the Operator.

This can be used to add a condition or resource to an operator after it has already been constructed.

Parameters

arg

The condition or resource to add.

property args

The list of arguments associated with the component.

Returns

arglist

compute(*self: holoscan.core._core.Operator*, *arg0: holoscan.core._core.InputContext*, *arg1: holoscan.core._core.OutputContext*, *arg2: holoscan.core._core.ExecutionContext*) None

Operator compute method. This method defines the primary computation to be executed by the operator.

property conditions

Conditions associated with the operator.

property description

YAML formatted string describing the operator.

property fragment

The fragment (`holoscan.core.Fragment`) that the operator belongs to.

property id

The identifier of the component.

The identifier is initially set to `-1`, and will become a valid value when the component is initialized.

With the default executor (*holoscan.gxf.GXFExecutor*), the identifier is set to the GXF component ID.

Returns

id

initialize(*self: holoscan.operators.bayer_demosaic._bayer_demosaic.BayerDemosaicOp*) None

Initialize the operator.

This method is called only once when the operator is created for the first time, and uses a light-weight initialization.

property name

The name of the operator.

property operator_type

The operator type.

holoscan.core.Operator.OperatorType enum representing the type of the operator. The two types currently implemented are native and GXF.

property resources

Resources associated with the operator.

setup(*self: holoscan.operators.bayer_demosaic._bayer_demosaic.BayerDemosaicOp*, *spec: holoscan.core._core.OperatorSpec*) None

Define the operator specification.

Parameters

spec

The operator specification.

property spec

The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

start(*self: holoscan.core._core.Operator*) None

Operator start method.

stop(*self: holoscan.core._core.Operator*) None

Operator stop method.

class holoscan.operators.FormatConverterOp

Bases: `holoscan.core._core.Operator`

Format conversion operator.

==Named Inputs==

source_videonvidia::gxf::Tensor or nvidia::gxf::VideoBuffer

The input video frame to process. If the input is a VideoBuffer it must be in format GXF_VIDEO_FORMAT_RGBA, GXF_VIDEO_FORMAT_RGB or GXF_VIDEO_FORMAT_NV12. If a video buffer is not found, the input port message is searched for a tensor with the name specified by *in_tensor_name*. This must be a tensor in one of several supported formats (unsigned 8-bit int or float32 graycale, unsigned 8-bit int RGB or RGBA YUV420 or NV12). The tensor or video buffer may be in either host or device memory (a host->device copy is performed if needed).

==Named Outputs==

tensornvidia::gxf::Tensor

The output video frame after processing. The shape, data type and number of channels of this output tensor will depend on the specific parameters that were set for this operator. The name of the Tensor transmitted on this port is determined by `out_tensor_name`.

==Device Memory Requirements==

When using this operator with a `holoscan.resources.BlockMemoryPool`, between 1 and 3 device memory blocks (`storage_type=1`) will be required based on the input tensors and parameters:

- 1.) In all cases there is a memory block needed for the output tensor. The size of this

  block will be `out_height * out_width * out_channels * out_element_size_bytes` where `(out_height, out_width)` will either be `(in_height, in_width)` (or `(resize_height, resize_width)` a resize was specified). *out_element_size* is the element size in bytes (e.g. 1 for RGB888 or 4 for Float32).

- 2.) If a resize is being done, another memory block is required for this. This block will

  have size `resize_height * resize_width * in_channels * in_element_size_bytes`.

- 3.) If the input tensor will be in host memory, a memory block is needed to copy the input

  to the device. This block will have size `in_height * in_width * in_channels * in_element_size_bytes`.

Thus when declaring the memory pool, *num_blocks* will be between 1 and 3 and *block_size* must be greater or equal to the largest of the individual blocks sizes described above.

Parameters

fragment

The fragment that the operator belongs to.

pool

Memory pool allocator used by the operator.

out_dtype

Destination data type. The available options are:

- `"rgb888"`
- `"uint8"`
- `"float32"`
- `"rgba8888"`
- `"yuv420"`
- `"nv12"`

in_dtype

Source data type. The available options are:

- `"rgb888"`
- `"uint8"`
- `"float32"`
- `"rgba8888"`
- `"yuv420"`
- `"nv12"`

in_tensor_name

The name of the input tensor. Default value is `""` (empty string).

out_tensor_name

The name of the output tensor. Default value is `""` (empty string).

scale_min

Output will be clipped to this minimum value. Default value is `0.0`.

scale_max

Output will be clipped to this maximum value. Default value is `1.0`.

alpha_value

Unsigned integer in range [0, 255], indicating the alpha channel value to use when converting from RGB to RGBA. Default value is `255`.

resize_height

Desired height for the (resized) output. Height will be unchanged if `resize_height` is `0`. Default value is `0`.

resize_width

Desired width for the (resized) output. Width will be unchanged if `resize_width` is `0`. Default value is `0`.

resize_mode

Resize mode enum value corresponding to NPP's NppiInterpolationMode. Values available at: https://docs.nvidia.com/cuda/npp/nppdefs.html?highlight=Two%20parameter%20cubic%20filter#c.NppiInterpolationMode

- NPPI_INTER_UNDEFINED (`0`): Undefined filtering interpolation mode.
- NPPI_INTER_NN (`1`): Nearest neighbor filtering.
- NPPI_INTER_LINEAR (`2`): Linear interpolation.
- NPPI_INTER_CUBIC (`4`): Cubic interpolation.
- NPPI_INTER_CUBIC2P_BSPLINE (`5`): Two-parameter cubic filter (B=1, C=0)

- NPPI_INTER_CUBIC2P_CATMULLROM (`6`): Two-parameter cubic filter (B=0, C=1/2)
- NPPI_INTER_CUBIC2P_B05C03 (`7`): Two-parameter cubic filter (B=1/2, C=3/10)
- NPPI_INTER_SUPER (`8`): Super sampling.
- NPPI_INTER_LANCZOS (`16`): Lanczos filtering.
- NPPI_INTER_LANCZOS3_ADVANCED (`17`): Generic Lanczos filtering with order 3.
- NPPI_SMOOTH_EDGE (`0x8000000`): Smooth edge filtering.

Default value is `0` (NPPI_INTER_UNDEFINED) which would be equivalent to `4` (NPPI_INTER_CUBIC).
channel_order

Sequence of integers describing how channel values are permuted. Default value is `[0, 1, 2]` for 3-channel images and `[0, 1, 2, 3]` for 4-channel images.

cuda_stream_pool

holoscan.resources.CudaStreamPool instance to allocate CUDA streams. Default value is `None`.

name

The name of the operator. Default value is `"format_converter"`.

Attributes

`args`	The list of arguments associated with the component.
`conditions`	Conditions associated with the operator.

description	YAML formatted string describing the operator.
fragment	The fragment (holoscan.core.Fragment) that the operator belongs to.
id	The identifier of the component.
name	The name of the operator.
operator_type	The operator type.
resources	Resources associated with the operator.
spec	The operator spec (holoscan.core.OperatorSpec) associated with the operator.

Methods

add_arg (*args, **kwargs)	Overloaded function.
compute (self, arg0, arg1, arg2)	Operator compute method.
initialize (self)	Initialize the operator.

setup (self, spec)	Define the operator specification.
start (self)	Operator start method.
stop (self)	Operator stop method.

OperatorType	

class OperatorType

Bases: pybind11_builtins.pybind11_object

Enum class for operator types used by the executor.

- NATIVE: Native operator.
- GXF: GXF operator.
- VIRTUAL: Virtual operator. (for internal use, not intended for use by application authors)

Members:

NATIVE

GXF

VIRTUAL

Attributes

name	

value	

GXF = *<OperatorType.GXF: 1>*

NATIVE = *<OperatorType.NATIVE: 0>*

VIRTUAL = *<OperatorType.VIRTUAL: 2>*

__init__(*self: holoscan.core._core.Operator.OperatorType*, *value: int*)   None

property name

property value

__init__(*self: holoscan.operators.format_converter._format_converter.FormatConverterOp*, *fragment: holoscan.core._core.Fragment*, **args*, *pool: holoscan.resources._resources.Allocator*, *out_dtype: str*, *in_dtype: str = ''*, *in_tensor_name: str = ''*, *out_tensor_name: str = ''*, *scale_min: float = 0.0*, *scale_max: float = 1.0*, *alpha_value: int = 255*, *resize_height: int = 0*, *resize_width: int = 0*, *resize_mode: int = 0*, *out_channel_order: List[int] = []*, *cuda_stream_pool: holoscan.resources._resources.CudaStreamPool = None*, *name: str = 'format_converter'*)   None

Format conversion operator.

==Named Inputs==

source_videonvidia::gxf::Tensor or nvidia::gxf::VideoBuffer

The input video frame to process. If the input is a VideoBuffer it must be in format GXF_VIDEO_FORMAT_RGBA, GXF_VIDEO_FORMAT_RGB or GXF_VIDEO_FORMAT_NV12. If a video buffer is not found, the input port message is searched for a tensor with the name specified by *in_tensor_name*. This must be a tensor in one of several supported formats (unsigned 8-bit int or float32 graycale, unsigned 8-bit int RGB or RGBA YUV420 or NV12). The tensor or video buffer may be in either host or device memory (a host->device copy is performed if needed).

==Named Outputs==

tensornvidia::gxf::Tensor

The output video frame after processing. The shape, data type and number of channels of this output tensor will depend on the specific parameters that were set for this operator. The name of the Tensor transmitted on this port is determined by `out_tensor_name`.

==Device Memory Requirements==

When using this operator with a `holoscan.resources.BlockMemoryPool`, between 1 and 3 device memory blocks (`storage_type=1`) will be required based on the input tensors and parameters:

- 1.) In all cases there is a memory block needed for the output tensor. The size of this

  block will be `out_height * out_width * out_channels * out_element_size_bytes` where `(out_height, out_width)` will either be `(in_height, in_width)` (or `(resize_height, resize_width)` a resize was specified). *out_element_size* is the element size in bytes (e.g. 1 for RGB888 or 4 for Float32).

- 2.) If a resize is being done, another memory block is required for this. This block will

  have size `resize_height * resize_width * in_channels * in_element_size_bytes`.

- 3.) If the input tensor will be in host memory, a memory block is needed to copy the input

  to the device. This block will have size `in_height * in_width * in_channels * in_element_size_bytes`.

Thus when declaring the memory pool, *num_blocks* will be between 1 and 3 and *block_size* must be greater or equal to the largest of the individual blocks sizes described above.

Parameters

fragment

The fragment that the operator belongs to.

pool

Memory pool allocator used by the operator.

out_dtype

Destination data type. The available options are:

- `"rgb888"`
- `"uint8"`
- `"float32"`
- `"rgba8888"`
- `"yuv420"`
- `"nv12"`

in_dtype

Source data type. The available options are:

- `"rgb888"`
- `"uint8"`
- `"float32"`
- `"rgba8888"`
- `"yuv420"`
- `"nv12"`

in_tensor_name

The name of the input tensor. Default value is `""` (empty string).

out_tensor_name

The name of the output tensor. Default value is `""` (empty string).

scale_min

Output will be clipped to this minimum value. Default value is `0.0`.

scale_max

Output will be clipped to this maximum value. Default value is `1.0`.

alpha_value

Unsigned integer in range [0, 255], indicating the alpha channel value to use when converting from RGB to RGBA. Default value is `255`.

resize_height

Desired height for the (resized) output. Height will be unchanged if `resize_height` is `0`. Default value is `0`.

resize_width

Desired width for the (resized) output. Width will be unchanged if `resize_width` is `0`. Default value is `0`.

resize_mode

Resize mode enum value corresponding to NPP's NppiInterpolationMode. Values available at: https://docs.nvidia.com/cuda/npp/nppdefs.html?highlight=Two%20parameter%20cubic%20filter#c.NppiInterpolationMode

- NPPI_INTER_UNDEFINED (`0`): Undefined filtering interpolation mode.

- NPPI_INTER_NN (`1`): Nearest neighbor filtering.
- NPPI_INTER_LINEAR (`2`): Linear interpolation.
- NPPI_INTER_CUBIC (`4`): Cubic interpolation.
- NPPI_INTER_CUBIC2P_BSPLINE (`5`): Two-parameter cubic filter (B=1, C=0)
- NPPI_INTER_CUBIC2P_CATMULLROM (`6`): Two-parameter cubic filter (B=0, C=1/2)
- NPPI_INTER_CUBIC2P_B05C03 (`7`): Two-parameter cubic filter (B=1/2, C=3/10)
- NPPI_INTER_SUPER (`8`): Super sampling.
- NPPI_INTER_LANCZOS (`16`): Lanczos filtering.
- NPPI_INTER_LANCZOS3_ADVANCED (`17`): Generic Lanczos filtering with order 3.
- NPPI_SMOOTH_EDGE (`0x8000000`): Smooth edge filtering.

Default value is `0` (NPPI_INTER_UNDEFINED) which would be equivalent to `4` (NPPI_INTER_CUBIC).
channel_order

Sequence of integers describing how channel values are permuted. Default value is `[0, 1, 2]` for 3-channel images and `[0, 1, 2, 3]` for 4-channel images.

cuda_stream_pool

holoscan.resources.CudaStreamPool instance to allocate CUDA streams. Default value is `None`.

name

The name of the operator. Default value is `"format_converter"`.

add_arg(**args*, ***kwargs*)

Overloaded function.

1. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Arg) -> None

Add an argument to the component.

2. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.ArgList) -> None

Add a list of arguments to the component.

3. add_arg(self: holoscan.core._core.Operator, **kwargs) -> None

Add arguments to the component via Python kwargs.

4. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Condition) -> None

5. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Resource) -> None

Add a condition or resource to the Operator.

This can be used to add a condition or resource to an operator after it has already been constructed.

Parameters

arg

The condition or resource to add.

property args

The list of arguments associated with the component.

Returns

arglist

compute(*self: holoscan.core._core.Operator*, *arg0: holoscan.core._core.InputContext*, *arg1: holoscan.core._core.OutputContext*, *arg2: holoscan.core._core.ExecutionContext*) None

Operator compute method. This method defines the primary computation to be executed by the operator.

property conditions

Conditions associated with the operator.

property description

YAML formatted string describing the operator.

property fragment

The fragment (`holoscan.core.Fragment`) that the operator belongs to.

property id

The identifier of the component.

The identifier is initially set to `-1`, and will become a valid value when the component is initialized.

With the default executor (*holoscan.gxf.GXFExecutor*), the identifier is set to the GXF component ID.

Returns

id

initialize(*self: holoscan.operators.format_converter._format_converter.FormatConverterOp*) None

Initialize the operator.

This method is called only once when the operator is created for the first time, and uses a light-weight initialization.

property name

The name of the operator.

property operator_type

The operator type.

holoscan.core.Operator.OperatorType enum representing the type of the operator. The two types currently implemented are native and GXF.

property resources

Resources associated with the operator.

setup(*self: holoscan.operators.format_converter._format_converter.FormatConverterOp*, *spec: holoscan.core._core.OperatorSpec*)   None

Define the operator specification.

Parameters

spec

The operator specification.

property spec

The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

start(*self: holoscan.core._core.Operator*)   None

Operator start method.

stop(*self: holoscan.core._core.Operator*)   None

Operator stop method.

class holoscan.operators.GXFCodeletOp(*fragment*, **args*, ***kwargs*)

Bases: `holoscan.operators.gxf_codelet._gxf_codelet.GXFCodeletOp`

GXF Codelet wrapper operator.

==Named Inputs==

Input ports are automatically defined based on the parameters of the underlying GXF Codelet that include the `nvidia::gxf::Receiver` component handle.

To view the information about the operator, refer to the *description* property of this object.

==Named Outputs==

Output ports are automatically defined based on the parameters of the underlying GXF Codelet that include the `nvidia::gxf::Transmitter` component handle.

To view the information about the operator, refer to the `description` property of this object.

Parameters

fragment

The fragment that the operator belongs to.

gxf_typename

The GXF type name that identifies the specific GXF Codelet being wrapped.

***args**

Additional positional arguments (`holoscan.core.Condition` or `holoscan.core.Resource`).

name

The name of the operator. Default value is `"gxf_codelet"`.

****kwargs**

The additional keyword arguments that can be passed depend on the underlying GXF Codelet. The additional parameters are the parameters of the underlying GXF Codelet that are neither specifically part of the `nvidia::gxf::Receiver` nor the `nvidia::gxf::Transmitter` components. These parameters can provide further customization and functionality to the operator.

Attributes

`args`	The list of arguments associated with the component.
`conditions`	Conditions associated with the operator.
`description`	YAML formatted string describing the operator.
`fragment`	The fragment (`holoscan.core.Fragment`) that the operator belongs to.
`gxf_typename`	The GXF type name of the resource.
`id`	The identifier of the component.
`name`	The name of the operator.
`operator_type`	The operator type.
`resources`	Resources associated with the operator.
`spec`	The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

Methods

`add_arg`(*args, **kwargs)	Overloaded function.
`compute`(self, arg0, arg1, arg2)	Operator compute method.
`initialize`(self)	Initialize the operator.
`setup`(self, arg0)	Define the operator specification.
`start`(self)	Operator start method.
`stop`(self)	Operator stop method.

OperatorType	

class OperatorType

Bases: `pybind11_builtins.pybind11_object`

Enum class for operator types used by the executor.

- NATIVE: Native operator.
- GXF: GXF operator.
- VIRTUAL: Virtual operator. (for internal use, not intended for use by application authors)

Members:

NATIVE

GXF

VIRTUAL

Attributes

name	

value	

GXF = *<OperatorType.GXF: 1>*

NATIVE = *<OperatorType.NATIVE: 0>*

VIRTUAL = *<OperatorType.VIRTUAL: 2>*

__init__(*self: holoscan.core._core.Operator.OperatorType*, *value: int*)   None

property name

property value

add_arg(**args*, ***kwargs*)

Overloaded function.

1. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Arg) -> None

Add an argument to the component.

2. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.ArgList) -> None

Add a list of arguments to the component.

3. add_arg(self: holoscan.core._core.Operator, **kwargs) -> None

Add arguments to the component via Python kwargs.

4. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Condition) -> None

5. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Resource) -> None

Add a condition or resource to the Operator.

This can be used to add a condition or resource to an operator after it has already been constructed.

Parameters

arg

The condition or resource to add.

property args

The list of arguments associated with the component.

Returns

arglist

compute(*self: holoscan.core._core.Operator*, *arg0: holoscan.core._core.InputContext*, *arg1: holoscan.core._core.OutputContext*, *arg2: holoscan.core._core.ExecutionContext*) None

Operator compute method. This method defines the primary computation to be executed by the operator.

property conditions

Conditions associated with the operator.

property description

YAML formatted string describing the operator.

property fragment

The fragment (`holoscan.core.Fragment`) that the operator belongs to.

property gxf_typename

The GXF type name of the resource.

Returns

str

The GXF type name of the resource

property id

The identifier of the component.

The identifier is initially set to `-1`, and will become a valid value when the component is initialized.

With the default executor (*holoscan.gxf.GXFExecutor*), the identifier is set to the GXF component ID.

Returns

id

initialize(*self: holoscan.operators.gxf_codelet._gxf_codelet.GXFCodeletOp*) None

Initialize the operator.

This method is called only once when the operator is created for the first time, and uses a light-weight initialization.

property name

The name of the operator.

property operator_type

The operator type.

holoscan.core.Operator.OperatorType enum representing the type of the operator. The two types currently implemented are native and GXF.

property resources

Resources associated with the operator.

setup(*self: holoscan.operators.gxf_codelet._gxf_codelet.GXFCodeletOp*, *arg0: holoscan.core._core.OperatorSpec*) None

Define the operator specification.

Parameters

spec

The operator specification.

property spec

The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

start(*self: holoscan.core._core.Operator*) None

Operator start method.

stop(*self: holoscan.core._core.Operator*) None

Operator stop method.

class holoscan.operators.HolovizOp(*fragment*, **args*, *allocator=None*, *receivers=()*, *tensors=()*, *color_lut=()*, *window_title='Holoviz'*, *display_name='DP-0'*, *width=1920*, *height=1080*, *framerate=60*, *use_exclusive_display=False*, *fullscreen=False*, *headless=False*, *enable_render_buffer_input=False*, *enable_render_buffer_output=False*, *enable_camera_pose_output=False*, *camera_pose_output_type='projection_matrix'*, *camera_eye=(0.0, 0.0, 1.0)*, *camera_look_at=(0.0, 0.0, 0.0)*, *camera_up=(0.0, 1.0, 0.0)*, *font_path=''*, *cuda_stream_pool=None*, *name='holoviz_op'*)

Bases: `holoscan.operators.holoviz._holoviz.HolovizOp`

Holoviz visualization operator using Holoviz module.

This is a Vulkan-based visualizer.

==Named Inputs==

receiversmulti-receiver accepting nvidia::gxf::Tensor and/or nvidia::gxf::VideoBuffer

Any number of upstream ports may be connected to this `receivers` port. This port can accept either VideoBuffers or Tensors. These inputs can be in either host or device memory. Each tensor or video buffer will result in a layer. The operator autodetects the layer type for certain input types (e.g. a video buffer will result in an image layer). For other input types or more complex use cases, input specifications can be provided either at initialization time as a parameter or dynamically at run time (via `input_specs`). On each call to `compute`, tensors corresponding to all names specified in the `tensors` parameter must be found or an exception will be raised. Any extra, named tensors not present in the `tensors` parameter specification (or optional, dynamic `input_specs` input) will be ignored.

input_specslist[holoscan.operators.HolovizOp.InputSpec], optional

A list of `InputSpec` objects. This port can be used to dynamically update the overlay specification at run time. No inputs are required on this port in order for the operator to `compute`.

render_buffer_inputnvidia::gxf::VideoBuffer, optional

An empty render buffer can optionally be provided. The video buffer must have format GXF_VIDEO_FORMAT_RGBA and be in device memory. This input port only exists if `enable_render_buffer_input` was set to `True`, in which case `compute` will only be called when a message arrives on this input.

==Named Outputs==

render_buffer_outputnvidia::gxf::VideoBuffer, optional

Output for a filled render buffer. If an input render buffer is specified, it is using that one, else it allocates a new buffer. The video buffer will have format GXF_VIDEO_FORMAT_RGBA and will be in device memory. This output is useful for

offline rendering or headless mode. This output port only exists if `enable_render_buffer_output` was set to `True`.
camera_pose_outputstd::array<float, 16> or nvidia::gxf::Pose3D, optional

The camera pose. Depending on the value of `camera_pose_output_type` this outputs a 4x4 row major projection matrix (type `std::array&lt;float, 16&gt;`) or the camera extrinsics model (type `nvidia::gxf::Pose3D`). This output port only exists if `enable_camera_pose_output` was set to `True`.

==Device Memory Requirements==

If `render_buffer_input` is enabled, the provided buffer is used and no memory block will be allocated. Otherwise, when using this operator with a `holoscan.resources.BlockMemoryPool`, a single device memory block is needed (`storage_type=1`). The size of this memory block can be determined by rounding the width and height up to the nearest even size and then padding the rows as needed so that the row stride is a multiple of 256 bytes. C++ code to calculate the block size is as follows

```
def get_block_size(height, width): height_even = height + (height & 1)
width_even = width + (width & 1) row_bytes = width_even * 4; # 4 bytes per
pixel for 8-bit RGBA row_stride = (row_bytes % 256 == 0) ? row_bytes :
((row_bytes // 256 + 1) * 256) return height_even * row_stride
```

Parameters

fragment

The fragment that the operator belongs to.

allocator

Allocator used to allocate render buffer output. If `None`, will default to `holoscan.core.UnboundedAllocator`.

receivers

List of input receivers.

tensors

List of input tensors. Each tensor is defined by a dictionary where the `"name"` key must correspond to a tensor sent to the operator’s input. See the notes section below for further details on how the tensor dictionary is defined.

color_lut

Color lookup table for tensors of type `color_lut`. Should be shape `(n_colors, 4)`.

window_title

Title on window canvas. Default value is `"Holoviz"`.

display_name

In exclusive mode, name of display to use as shown with *xrandr* or *hwinfo –monitor*. Default value is `"DP-0"`.

width

Window width or display resolution width if in exclusive or fullscreen mode. Default value is `1920`.

height

Window height or display resolution width if in exclusive or fullscreen mode. Default value is `1080`.

framerate

Display framerate in Hz if in exclusive mode. Default value is `60.0`.

use_exclusive_display

Enable exclusive display. Default value is `False`.

fullscreen

Enable fullscreen window. Default value is `False`.

headless

Enable headless mode. No window is opened, the render buffer is output to port `render_buffer_output`. Default value is `False`.

enable_render_buffer_input

If `True`, an additional input port, named `"render_buffer_input"` is added to the operator. Default value is `False`.

enable_render_buffer_output

If `True`, an additional output port, named `"render_buffer_output"` is added to the operator. Default value is `False`.

enable_camera_pose_output

If `True`, an additional output port, named `"camera_pose_output"` is added to the operator. Default value is `False`.

camera_pose_output_type

Type of data output at `"camera_pose_output"`. Supported values are `projection_matrix` and `extrinsics_model`. Default value is `projection_matrix`.

camera_eye

Initial camera eye position. Default value is `(0.0, 0.0, 1.0)`.

camera_look_at

Initial camera look at position. Default value is `(0.0, 0.0, 0.0)`.

camera_up

Initial camera up vector. Default value is `(0.0, 1.0, 0.0)`.

font_path

File path for the font used for rendering text. Default value is `""`.

cuda_stream_pool

`holoscan.resources.CudaStreamPool` instance to allocate CUDA streams. Default value is `None`.

name

The name of the operator. Default value is `"holoviz_op"`.

Notes

The `tensors` argument is used to specify the tensors to display. Each tensor is defined using a dictionary, that must, at minimum include a ‘name’ key that corresponds to a tensor found on the operator’s input. A ‘type’ key should also be provided to indicate the type of entry to display. The ‘type’ key will be one of { `"color"`, `"color_lut"`, `"crosses"`, `"lines"`, `"lines_3d"`, `"line_strip"`, `"line_strip_3d"`, `"ovals"`, `"points"`, `"points_3d"`, `"rectangles"`, `"text"`, `"triangles"`, `"triangles_3d"`, `"depth_map"`, `"depth_map_color"`, `"unknown"`}. The default type is `"unknown"` which will attempt to guess the corresponding type based on the tensor dimensions. Concrete examples are given below.

To show a single 2D RGB or RGBA image, use a list containing a single tensor of type `"color"`.

```
tensors = [dict(name="video", type="color", opacity=1.0, priority=0)]
```

Here, the optional key `opacity` is used to scale the opacity of the tensor. The `priority` key is used to specify the render priority for layers. Layers with a higher priority will be rendered on top of those with a lower priority.

If we also had a ``"boxes"` `` tensor representing rectangular bounding boxes, we could display them on top of the image like this.

```
tensors = [ dict(name="video", type="color", priority=0), dict(name="boxes",
type="rectangles", color=[1.0, 0.0, 0.0], line_width=2, priority=1), ]
```

where the `color` and `line_width` keys specify the color and line width of the bounding box.

The details of the dictionary is as follows:

- **name**: name of the tensor containing the input data to display
    - type: `str`
- **type**: input type (default `"unknown"`)
    - type: `str`
    - possible values:
        - **unknown**: unknown type, the operator tries to guess the type by inspecting the tensor.
        - **color**: RGB or RGBA color 2d image.
        - **color_lut**: single channel 2d image, color is looked up.
        - **points**: point primitives, one coordinate (x, y) per primitive.
        - **lines**: line primitives, two coordinates (x0, y0) and (x1, y1) per primitive.
        - **line_strip**: line strip primitive, a line primitive i is defined by each coordinate (xi, yi) and the following (xi+1, yi+1).
        - **triangles**: triangle primitive, three coordinates (x0, y0), (x1, y1) and (x2, y2) per primitive.
        - **crosses**: cross primitive, a cross is defined by the center coordinate and the size (xi, yi, si).
        - **rectangles**: axis aligned rectangle primitive, each rectangle is defined by two coordinates (xi, yi) and (xi+1, yi+1).
        - **ovals**: oval primitive, an oval primitive is defined by the center coordinate and the axis sizes (xi, yi, sxi, syi).

        - **text**: text is defined by the top left coordinate and the size (x, y, s) per string, text strings are defined by InputSpec member **text**.

        - **depth_map**: single channel 2d array where each element represents a depth value. The data is rendered as a 3d object using points, lines or triangles. The color for the elements can be specified through `depth_map_color`. Supported format: 8-bit unsigned normalized format that has a single 8-bit depth component.

        - **depth_map_color**: RGBA 2d image, same size as the depth map. One color value for each element of the depth map grid. Supported format: 32-bit unsigned normalized format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, an 8-bit B component in byte 2, and an 8-bit A component in byte 3.

- **opacity**: layer opacity, 1.0 is fully opaque, 0.0 is fully transparent (default: `1.0`)

    - type: `float`

- **priority**: layer priority, determines the render order, layers with higher priority

  values are rendered on top of layers with lower priority values (default: `0`)

    - type: `int`

- **color**: RGBA color of rendered geometry (default: `[1.f, 1.f, 1.f, 1.f]`)

    - type: `List[float]`

- **line_width**: line width for geometry made of lines (default: `1.0`)

    - type: `float`

- **point_size**: point size for geometry made of points (default: `1.0`)

    - type: `float`

- **text**: array of text strings, used when `type` is text (default: `[]`)

    - type: `List[str]`

- **depth_map_render_mode**: depth map render mode (default: `points`)

    - type: `str`

    - possible values:

        - **points**: render as points

        - **lines**: render as lines

        - **triangles**: render as triangles

1. Displaying Color Images

   Image data can either be on host or device (GPU). Multiple image formats are supported

    - R 8 bit unsigned

    - R 16 bit unsigned

    - R 16 bit float

    - R 32 bit unsigned

    - R 32 bit float

    - RGB 8 bit unsigned

    - BGR 8 bit unsigned

    - RGBA 8 bit unsigned

    - BGRA 8 bit unsigned

    - RGBA 16 bit unsigned

    - RGBA 16 bit float

    - RGBA 32 bit float

   When the `type` parameter is set to `color_lut` the final color is looked up using the values from the `color_lut` parameter. For color lookups these image formats are supported

   - R 8 bit unsigned
   - R 16 bit unsigned
   - R 32 bit unsigned

2. Drawing Geometry

   In all cases, `x` and `y` are normalized coordinates in the range `[0, 1]`. The `x` and `y` correspond to the horizontal and vertical axes of the display, respectively. The origin `(0, 0)` is at the top left of the display. Geometric primitives outside of the visible area are clipped. Coordinate arrays are expected to have the shape `(N, C)` where `N` is the coordinate count and `C` is the component count for each coordinate.

   - Points are defined by a `(x, y)` coordinate pair.
   - Lines are defined by a set of two `(x, y)` coordinate pairs.
   - Lines strips are defined by a sequence of `(x, y)` coordinate pairs. The first two coordinates define the first line, each additional coordinate adds a line connecting to the previous coordinate.
   - Triangles are defined by a set of three `(x, y)` coordinate pairs.
   - Crosses are defined by `(x, y, size)` tuples. `size` specifies the size of the cross in the `x` direction and is optional, if omitted it’s set to `0.05`. The size in the `y` direction is calculated using the aspect ratio of the window to make the crosses square.
   - Rectangles (bounding boxes) are defined by a pair of 2-tuples defining the upper-left and lower-right coordinates of a box: `(x1, y1), (x2, y2)`.
   - Ovals are defined by `(x, y, size_x, size_y)` tuples. `size_x` and `size_y` are optional, if omitted they are set to `0.05`.

- Texts are defined by `(x, y, size)` tuples. `size` specifies the size of the text in `y` direction and is optional, if omitted it’s set to `0.05`. The size in the `x` direction is calculated using the aspect ratio of the window. The index of each coordinate references a text string from the `text` parameter and the index is clamped to the size of the text array. For example, if there is one item set for the `text` parameter, e.g. `text=["my_text"]` and three coordinates, then `my_text` is rendered three times. If `text=["first text", "second text"]` and three coordinates are specified, then `first text` is rendered at the first coordinate, `second text` at the second coordinate and then `second text` again at the third coordinate. The `text` string array is fixed and can’t be changed after initialization. To hide text which should not be displayed, specify coordinates greater than `(1.0, 1.0)` for the text item, the text is then clipped away.

- 3D Points are defined by a `(x, y, z)` coordinate tuple.

- 3D Lines are defined by a set of two `(x, y, z)` coordinate tuples.

- 3D Lines strips are defined by a sequence of `(x, y, z)` coordinate tuples. The first two coordinates define the first line, each additional coordinate adds a line connecting to the previous coordinate.

- 3D Triangles are defined by a set of three `(x, y, z)` coordinate tuples.

3. Displaying Depth Maps

   When `type` is `depth_map` the provided data is interpreted as a rectangular array of depth values. Additionally a 2d array with a color value for each point in the grid can be specified by setting `type` to `depth_map_color`.

   The type of geometry drawn can be selected by setting `depth_map_render_mode`.

   Depth maps are rendered in 3D and support camera movement. The camera is controlled using the mouse:

   - Orbit (LMB)

- Pan (LMB + CTRL | MMB)
- Dolly (LMB + SHIFT | RMB | Mouse wheel)
- Look Around (LMB + ALT | LMB + CTRL + SHIFT)
- Zoom (Mouse wheel + SHIFT)

4. Output

   By default a window is opened to display the rendering, but the extension can also be run in headless mode with the `headless` parameter.

   Using a display in exclusive mode is also supported with the `use_exclusive_display` parameter. This reduces the latency by avoiding the desktop compositor.

   The rendered framebuffer can be output to `render_buffer_output`.

Attributes

`args`	The list of arguments associated with the component.
`conditions`	Conditions associated with the operator.
`description`	YAML formatted string describing the operator.
`fragment`	The fragment (`holoscan.core.Fragment`) that the operator belongs to.
`id`	The identifier of the component.
`name`	The name of the operator.
`operator_type`	The operator type.

resources	Resources associated with the operator.
spec	The operator spec (holoscan.core.OperatorSpec) associated with the operator.

Methods

InputSpec	InputSpec for the HolovizOp operator.
add_arg(*args, **kwargs)	Overloaded function.
compute(self, arg0, arg1, arg2)	Operator compute method.
initialize(self)	Initialize the operator.
setup(self, spec)	Define the operator specification.
start(self)	Operator start method.
stop(self)	Operator stop method.

DepthMapRenderMode	
InputType	

OperatorType	

class DepthMapRenderMode

Bases: `pybind11_builtins.pybind11_object`

Members:

POINTS

LINES

TRIANGLES

Attributes

`name`	

value	

LINES = *<DepthMapRenderMode.LINES: 1>*

POINTS = *<DepthMapRenderMode.POINTS: 0>*

TRIANGLES = *<DepthMapRenderMode.TRIANGLES: 2>*

__init__(*self: holoscan.operators.holoviz._holoviz.HolovizOp.DepthMapRenderMode*, *value: int*) None

property name

property value

class InputSpec

Bases: `pybind11_builtins.pybind11_object`

InputSpec for the HolovizOp operator.

Parameters

tensor_name

The tensor name for this input.

type

The type of data that this tensor represents.

Attributes

type	(holoscan.operators.HolovizOp.InputType) The type of data that this tensor represents.
opacity	(float) The opacity of the object. Must be in range [0.0, 1.0] where 1.0 is fully opaque.
priority	(int) Layer priority, determines the render order. Layers with higher priority values are rendered on top of layers with lower priority.
color	(4-tuple of float) RGBA values in range [0.0, 1.0] for rendered geometry.
line_width	(float) Line width for geometry made of lines.
point_size	(float) Point size for geometry made of points.
text	(sequence of str) Sequence of strings used when type is *HolovizOp.InputType.TEXT*.
depth_map_render_mode	(holoscan.operators.HolovizOp.DepthMapRenderMode) The depth map render mode. Used only if *type* is *HolovizOp.InputType.DEPTH_MAP* or *HolovizOp.InputType.DEPTH_MAP_COLOR*.
views	(list of HolovizOp.InputSpec.View) Sequence of layer views. By default a layer will fill the whole window. When using a view, the layer can be

	placed freely within the window. When multiple views are specified, the layer is drawn multiple times using the specified layer views.

Methods

`View`	View for the InputSpec of a HolovizOp operator.
`description`(self)	Returns

class View

Bases: `pybind11_builtins.pybind11_object`

View for the InputSpec of a HolovizOp operator.

Notes

Layers can also be placed in 3D space by specifying a 3D transformation *matrix*. Note that for geometry layers there is a default matrix which allows coordinates in the range of [0 … 1] instead of the Vulkan [-1 … 1] range. When specifying a matrix for a geometry layer, this default matrix is overwritten.

When multiple views are specified, the layer is drawn multiple times using the specified layer views.

It’s possible to specify a negative term for height, which flips the image. When using a negative height, one should also adjust the y value to point to the lower left corner of the viewport instead of the upper left corner.

Attributes

offset_x, offset_y	(float) Offset of top-left corner of the view. (0, 0) is the upper left and (1, 1) is the lower right.
width	(float) Normalized width (range [0.0, 1.0]).
height	(float) Normalized height (range [0.0, 1.0]).

matrix	(sequence of float) 16-elements representing a 4x4 transformation matrix.

__init__(*self: holoscan.operators.holoviz._holoviz.HolovizOp.InputSpec.View*) None

View for the InputSpec of a HolovizOp operator.

Notes

Layers can also be placed in 3D space by specifying a 3D transformation *matrix*. Note that for geometry layers there is a default matrix which allows coordinates in the range of [0 … 1] instead of the Vulkan [-1 … 1] range. When specifying a matrix for a geometry layer, this default matrix is overwritten.

When multiple views are specified, the layer is drawn multiple times using the specified layer views.

It’s possible to specify a negative term for height, which flips the image. When using a negative height, one should also adjust the y value to point to the lower left corner of the viewport instead of the upper left corner.

Attributes

offset_x, offset_y	(float) Offset of top-left corner of the view. (0, 0) is the upper left and (1, 1) is the lower right.
width	(float) Normalized width (range [0.0, 1.0]).
height	(float) Normalized height (range [0.0, 1.0]).
matrix	(sequence of float) 16-elements representing a 4x4 transformation matrix.

property height

property matrix

property offset_x

property offset_y

property width

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: holoscan.operators.holoviz._holoviz.HolovizOp.InputSpec, arg0: str, arg1: holoscan.operators.holoviz._holoviz.HolovizOp.InputType) -> None

InputSpec for the HolovizOp operator.

Parameters

tensor_name

The tensor name for this input.

type

The type of data that this tensor represents.

Attributes

type	(holoscan.operators.HolovizOp.InputType) The type of data that this tensor represents.
opacity	(float) The opacity of the object. Must be in range [0.0, 1.0] where 1.0 is fully opaque.
priority	(int) Layer priority, determines the render order. Layers with higher priority values are rendered on top of layers with lower priority.
color	(4-tuple of float) RGBA values in range [0.0, 1.0] for rendered geometry.
line_width	(float) Line width for geometry made of lines.
point_size	(float) Point size for geometry made of points.
text	(sequence of str) Sequence of strings used when type is *HolovizOp.InputType.TEXT*.

depth_map_render_mode	(holoscan.operators.HolovizOp.DepthMapRenderMode) The depth map render mode. Used only if *type* is *HolovizOp.InputType.DEPTH_MAP* or *HolovizOp.InputType.DEPTH_MAP_COLOR*.
views	(list of HolovizOp.InputSpec.View) Sequence of layer views. By default a layer will fill the whole window. When using a view, the layer can be placed freely within the window. When multiple views are specified, the layer is drawn multiple times using the specified layer views.
2. __init__(self: holoscan.operators.holoviz._holoviz.HolovizOp.InputSpec, arg0: str, arg1: str) -> None	

property color

property depth_map_render_mode

description(*self: holoscan.operators.holoviz._holoviz.HolovizOp.InputSpec*)   str

Returns

description

YAML string representation of the InputSpec class.

property line_width

property opacity

property point_size

property priority

property text

property type

property views

class InputType

Bases: `pybind11_builtins.pybind11_object`

Members:

UNKNOWN

COLOR

COLOR_LUT

POINTS

LINES

LINE_STRIP

TRIANGLES

CROSSES

RECTANGLES

OVALS

TEXT

DEPTH_MAP

DEPTH_MAP_COLOR

POINTS_3D

LINES_3D

LINE_STRIP_3D

TRIANGLES_3D

Attributes

name	

value	

COLOR = *<InputType.COLOR: 1>*

COLOR_LUT = *<InputType.COLOR_LUT: 2>*

CROSSES = *<InputType.CROSSES: 7>*

DEPTH_MAP = *<InputType.DEPTH_MAP: 11>*

DEPTH_MAP_COLOR = *<InputType.DEPTH_MAP_COLOR: 12>*

LINES = *<InputType.LINES: 4>*

LINES_3D = *<InputType.LINES_3D: 14>*

LINE_STRIP = *<InputType.LINE_STRIP: 5>*

LINE_STRIP_3D = *<InputType.LINE_STRIP_3D: 15>*

OVALS = *<InputType.OVALS: 9>*

POINTS = *<InputType.POINTS: 3>*

POINTS_3D = *<InputType.POINTS_3D: 13>*

RECTANGLES = *<InputType.RECTANGLES: 8>*

TEXT = *<InputType.TEXT: 10>*

TRIANGLES = *<InputType.TRIANGLES: 6>*

TRIANGLES_3D = *<InputType.TRIANGLES_3D: 16>*

UNKNOWN = *<InputType.UNKNOWN: 0>*

__init__(*self: holoscan.operators.holoviz._holoviz.HolovizOp.InputType*, *value: int*) None

property name

property value

class OperatorType

Bases: `pybind11_builtins.pybind11_object`

Enum class for operator types used by the executor.

- NATIVE: Native operator.
- GXF: GXF operator.
- VIRTUAL: Virtual operator. (for internal use, not intended for use by application authors)

Members:

NATIVE

GXF

VIRTUAL

Attributes

`name`	

value	

GXF = *<OperatorType.GXF: 1>*

NATIVE = *<OperatorType.NATIVE: 0>*

VIRTUAL = *<OperatorType.VIRTUAL: 2>*

__init__(*self: holoscan.core._core.Operator.OperatorType*, *value: int*) None

property name

property value

__init__(*self: holoscan.operators.holoviz._holoviz.HolovizOp, fragment: holoscan.core._core.Fragment, *args, allocator: holoscan.resources._resources.Allocator, receivers: List[holoscan.core._core.IOSpec] = [], tensors: List[holoscan::ops::HolovizOp::InputSpec] = [], color_lut: List[List[float]] = [], window_title: str = 'Holoviz', display_name: str = 'DP-0', width: int = 1920, height: int = 1080, framerate: int = 60, use_exclusive_display: bool = False, fullscreen: bool = False, headless: bool = False, enable_render_buffer_input: bool = False, enable_render_buffer_output: bool = False, enable_camera_pose_output: bool = False, camera_pose_output_type: str = 'projection_matrix', camera_eye: Annotated[List[float], FixedSize(3)] = [0.0, 0.0, 1.0], camera_look_at: Annotated[List[float], FixedSize(3)] = [0.0, 0.0, 0.0], camera_up: Annotated[List[float], FixedSize(3)] = [0.0, 1.0, 1.0], font_path: str = '', cuda_stream_pool: holoscan.resources._resources.CudaStreamPool = None, name: str = 'holoviz_op'*)
None

Holoviz visualization operator using Holoviz module.

This is a Vulkan-based visualizer.

==Named Inputs==

receiversmulti-receiver accepting nvidia::gxf::Tensor and/or nvidia::gxf::VideoBuffer

Any number of upstream ports may be connected to this `receivers` port. This port can accept either VideoBuffers or Tensors. These inputs can be in either host or device memory. Each tensor or video buffer will result in a layer. The operator autodetects the layer type for certain input types (e.g. a video buffer will result in an image layer). For other input types or more complex use cases, input specifications can be provided either at initialization time as a parameter or dynamically at run time (via `input_specs`). On each call to `compute`, tensors corresponding to all names specified in the `tensors` parameter must be found or an exception will be raised. Any extra, named tensors not present in the `tensors` parameter specification (or optional, dynamic `input_specs` input) will be ignored.

input_specslist[holoscan.operators.HolovizOp.InputSpec], optional

A list of `InputSpec` objects. This port can be used to dynamically update the overlay specification at run time. No inputs are required on this port in order for the operator to `compute`.

render_buffer_inputnvidia::gxf::VideoBuffer, optional

An empty render buffer can optionally be provided. The video buffer must have format GXF_VIDEO_FORMAT_RGBA and be in device memory. This input port only exists if `enable_render_buffer_input` was set to `True`, in which case `compute` will only be called when a message arrives on this input.

==Named Outputs==

render_buffer_outputnvidia::gxf::VideoBuffer, optional

Output for a filled render buffer. If an input render buffer is specified, it is using that one, else it allocates a new buffer. The video buffer will have format GXF_VIDEO_FORMAT_RGBA and will be in device memory. This output is useful for offline rendering or headless mode. This output port only exists if `enable_render_buffer_output` was set to `True`.

camera_pose_outputstd::array<float, 16> or nvidia::gxf::Pose3D, optional

The camera pose. Depending on the value of `camera_pose_output_type` this outputs a 4x4 row major projection matrix (type `std::array&lt;float, 16&gt;`) or the camera extrinsics model (type `nvidia::gxf::Pose3D`). This output port only exists if `enable_camera_pose_output` was set to `True`.

==Device Memory Requirements==

If `render_buffer_input` is enabled, the provided buffer is used and no memory block will be allocated. Otherwise, when using this operator with a `holoscan.resources.BlockMemoryPool`, a single device memory block is needed (`storage_type=1`). The size of this memory block can be determined by rounding the width and height up to the nearest even size and then padding the rows as needed so that the row stride is a multiple of 256 bytes. C++ code to calculate the block size is as follows

```
def get_block_size(height, width): height_even = height + (height & 1)
width_even = width + (width & 1) row_bytes = width_even * 4; # 4 bytes
per pixel for 8-bit RGBA row_stride = (row_bytes % 256 == 0) ? row_bytes :
((row_bytes // 256 + 1) * 256) return height_even * row_stride
```

Parameters

fragment

The fragment that the operator belongs to.

allocator

Allocator used to allocate render buffer output. If `None`, will default to `holoscan.core.UnboundedAllocator`.

receivers

List of input receivers.

tensors

List of input tensors. Each tensor is defined by a dictionary where the `"name"` key must correspond to a tensor sent to the operator’s input. See the notes section below for further details on how the tensor dictionary is defined.

color_lut

Color lookup table for tensors of type `color_lut`. Should be shape `(n_colors, 4)`.

window_title

Title on window canvas. Default value is `"Holoviz"`.

display_name

In exclusive mode, name of display to use as shown with *xrandr* or *hwinfo –monitor*. Default value is `"DP-0"`.

width

Window width or display resolution width if in exclusive or fullscreen mode. Default value is `1920`.

height

Window height or display resolution width if in exclusive or fullscreen mode. Default value is `1080`.

framerate

Display framerate in Hz if in exclusive mode. Default value is `60.0`.

use_exclusive_display

Enable exclusive display. Default value is `False`.

fullscreen

Enable fullscreen window. Default value is `False`.

headless

Enable headless mode. No window is opened, the render buffer is output to port `render_buffer_output`. Default value is `False`.

enable_render_buffer_input

If `True`, an additional input port, named `"render_buffer_input"` is added to the operator. Default value is `False`.

enable_render_buffer_output

If `True`, an additional output port, named `"render_buffer_output"` is added to the operator. Default value is `False`.

enable_camera_pose_output

If `True`, an additional output port, named `"camera_pose_output"` is added to the operator. Default value is `False`.

camera_pose_output_type

Type of data output at `"camera_pose_output"`. Supported values are `projection_matrix` and `extrinsics_model`. Default value is `projection_matrix`.

camera_eye

Initial camera eye position. Default value is `(0.0, 0.0, 1.0)`.

camera_look_at

Initial camera look at position. Default value is `(0.0, 0.0, 0.0)`.

camera_up

Initial camera up vector. Default value is `(0.0, 1.0, 0.0)`.

font_path

File path for the font used for rendering text. Default value is `""`.

cuda_stream_pool

`holoscan.resources.CudaStreamPool` instance to allocate CUDA streams. Default value is `None`.

name

The name of the operator. Default value is `"holoviz_op"`.

Notes

The `tensors` argument is used to specify the tensors to display. Each tensor is defined using a dictionary, that must, at minimum include a ‘name’ key that corresponds to a tensor found on the operator’s input. A ‘type’ key should also be provided to indicate the type of entry to display. The ‘type’ key will be one of {`"color"`, `"color_lut"`, `"crosses"`, `"lines"`, `"lines_3d"`, `"line_strip"`, `"line_strip_3d"`, `"ovals"`, `"points"`, `"points_3d"`, `"rectangles"`, `"text"`, `"triangles"`, `"triangles_3d"`, `"depth_map"`, `"depth_map_color"`, `"unknown"` }. The default type is `"unknown"` which will attempt to guess the corresponding type based on the tensor dimensions. Concrete examples are given below.

To show a single 2D RGB or RGBA image, use a list containing a single tensor of type `"color"`.

```
tensors = [dict(name="video", type="color", opacity=1.0, priority=0)]
```

Here, the optional key `opacity` is used to scale the opacity of the tensor. The `priority` key is used to specify the render priority for layers. Layers with a higher priority will be rendered on top of those with a lower priority.

If we also had a `` "boxes"` `` tensor representing rectangular bounding boxes, we could display them on top of the image like this.

```
tensors = [ dict(name="video", type="color", priority=0),
dict(name="boxes", type="rectangles", color=[1.0, 0.0, 0.0], line_width=2,
priority=1), ]
```

where the `color` and `line_width` keys specify the color and line width of the bounding box.

The details of the dictionary is as follows:

- **name**: name of the tensor containing the input data to display
  - type: `str`
- **type**: input type (default `"unknown"`)
  - type: `str`
  - possible values:

- **unknown**: unknown type, the operator tries to guess the type by inspecting the tensor.

- **color**: RGB or RGBA color 2d image.

- **color_lut**: single channel 2d image, color is looked up.

- **points**: point primitives, one coordinate (x, y) per primitive.

- **lines**: line primitives, two coordinates (x0, y0) and (x1, y1) per primitive.

- **line_strip**: line strip primitive, a line primitive i is defined by each coordinate (xi, yi) and the following (xi+1, yi+1).

- **triangles**: triangle primitive, three coordinates (x0, y0), (x1, y1) and (x2, y2) per primitive.

- **crosses**: cross primitive, a cross is defined by the center coordinate and the size (xi, yi, si).

- **rectangles**: axis aligned rectangle primitive, each rectangle is defined by two coordinates (xi, yi) and (xi+1, yi+1).

- **ovals**: oval primitive, an oval primitive is defined by the center coordinate and the axis sizes (xi, yi, sxi, syi).

- **text**: text is defined by the top left coordinate and the size (x, y, s) per string, text strings are defined by InputSpec member **text**.

- **depth_map**: single channel 2d array where each element represents a depth value. The data is rendered as a 3d object using points, lines or triangles. The color for the elements can be specified through `depth_map_color`. Supported format: 8-bit unsigned normalized format that has a single 8-bit depth component.

- **depth_map_color**: RGBA 2d image, same size as the depth map. One color value for each element of the depth map grid. Supported format: 32-bit unsigned normalized format that has an 8-bit R component in byte 0, an 8-bit G component in byte

        1, an 8-bit B component in byte 2, and an 8-bit A component in byte 3.

- **opacity**: layer opacity, 1.0 is fully opaque, 0.0 is fully transparent (default: `1.0`)
    - type: `float`
- **priority**: layer priority, determines the render order, layers with higher priority

  values are rendered on top of layers with lower priority values (default: `0`)
    - type: `int`
- **color**: RGBA color of rendered geometry (default: `[1.f, 1.f, 1.f, 1.f]`)
    - type: `List[float]`
- **line_width**: line width for geometry made of lines (default: `1.0`)
    - type: `float`
- **point_size**: point size for geometry made of points (default: `1.0`)
    - type: `float`
- **text**: array of text strings, used when `type` is text (default: `[]`)
    - type: `List[str]`
- **depth_map_render_mode**: depth map render mode (default: `points`)
    - type: `str`
    - possible values:
        - **points**: render as points

        - **lines**: render as lines

        - **triangles**: render as triangles

1. Displaying Color Images

   Image data can either be on host or device (GPU). Multiple image formats are supported

   - R 8 bit unsigned
   - R 16 bit unsigned
   - R 16 bit float
   - R 32 bit unsigned
   - R 32 bit float
   - RGB 8 bit unsigned
   - BGR 8 bit unsigned
   - RGBA 8 bit unsigned
   - BGRA 8 bit unsigned
   - RGBA 16 bit unsigned
   - RGBA 16 bit float
   - RGBA 32 bit float

   When the `type` parameter is set to `color_lut` the final color is looked up using the values from the `color_lut` parameter. For color lookups these image formats are supported

   - R 8 bit unsigned
   - R 16 bit unsigned
   - R 32 bit unsigned

2. Drawing Geometry

   In all cases, `x` and `y` are normalized coordinates in the range `[0, 1]`. The `x` and `y` correspond to the horizontal and vertical axes of the display, respectively. The origin `(0, 0)` is at the top left of the display. Geometric primitives outside of the visible area are clipped. Coordinate arrays are expected to have the shape `(N, C)` where `N` is the coordinate count and `C` is the component count for each coordinate.

   - Points are defined by a `(x, y)` coordinate pair.

   - Lines are defined by a set of two `(x, y)` coordinate pairs.

   - Lines strips are defined by a sequence of `(x, y)` coordinate pairs. The first two coordinates define the first line, each additional coordinate adds a line connecting to the previous coordinate.

   - Triangles are defined by a set of three `(x, y)` coordinate pairs.

   - Crosses are defined by `(x, y, size)` tuples. `size` specifies the size of the cross in the `x` direction and is optional, if omitted it’s set to `0.05`. The size in the `y` direction is calculated using the aspect ratio of the window to make the crosses square.

   - Rectangles (bounding boxes) are defined by a pair of 2-tuples defining the upper-left and lower-right coordinates of a box: `(x1, y1), (x2, y2)`.

   - Ovals are defined by `(x, y, size_x, size_y)` tuples. `size_x` and `size_y` are optional, if omitted they are set to `0.05`.

   - Texts are defined by `(x, y, size)` tuples. `size` specifies the size of the text in `y` direction and is optional, if omitted it’s set to `0.05`. The size in the `x` direction is calculated using the aspect ratio of the window. The index of each coordinate references a text string from the `text` parameter and the index is clamped to the size of the text array. For example, if there is one item set for the `text` parameter, e.g. `text=["my_text"]` and three coordinates, then `my_text` is rendered three times. If `text=["first text", "second text"]` and three coordinates are specified, then `first text` is rendered at the first coordinate, `second text` at the second coordinate and then `second text` again at the third coordinate. The `text` string array is fixed and can't be changed after initialization. To hide text which should not be displayed, specify coordinates greater than `(1.0, 1.0)` for the text item, the text is then clipped away.

    - 3D Points are defined by a `(x, y, z)` coordinate tuple.

    - 3D Lines are defined by a set of two `(x, y, z)` coordinate tuples.

    - 3D Lines strips are defined by a sequence of `(x, y, z)` coordinate tuples. The first two coordinates define the first line, each additional coordinate adds a line connecting to the previous coordinate.

    - 3D Triangles are defined by a set of three `(x, y, z)` coordinate tuples.

3. Displaying Depth Maps

   When `type` is `depth_map` the provided data is interpreted as a rectangular array of depth values. Additionally a 2d array with a color value for each point in the grid can be specified by setting `type` to `depth_map_color`.

   The type of geometry drawn can be selected by setting `depth_map_render_mode`.

   Depth maps are rendered in 3D and support camera movement. The camera is controlled using the mouse:

    - Orbit (LMB)

    - Pan (LMB + CTRL | MMB)

    - Dolly (LMB + SHIFT | RMB | Mouse wheel)

  - Look Around (LMB + ALT | LMB + CTRL + SHIFT)
  - Zoom (Mouse wheel + SHIFT)

4. Output

   By default a window is opened to display the rendering, but the extension can also be run in headless mode with the `headless` parameter.

   Using a display in exclusive mode is also supported with the `use_exclusive_display` parameter. This reduces the latency by avoiding the desktop compositor.

   The rendered framebuffer can be output to `render_buffer_output`.

add_arg(**args*, ***kwargs*)

Overloaded function.

1. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Arg) -> None

Add an argument to the component.

2. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.ArgList) -> None

Add a list of arguments to the component.

3. add_arg(self: holoscan.core._core.Operator, **kwargs) -> None

Add arguments to the component via Python kwargs.

4. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Condition) -> None

5. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Resource) -> None

Add a condition or resource to the Operator.

This can be used to add a condition or resource to an operator after it has already been constructed.

Parameters

arg

The condition or resource to add.

property args

The list of arguments associated with the component.

Returns

arglist

compute(*self: holoscan.core._core.Operator*, *arg0: holoscan.core._core.InputContext*, *arg1: holoscan.core._core.OutputContext*, *arg2: holoscan.core._core.ExecutionContext*) None

Operator compute method. This method defines the primary computation to be executed by the operator.

property conditions

Conditions associated with the operator.

property description

YAML formatted string describing the operator.

property fragment

The fragment (`holoscan.core.Fragment`) that the operator belongs to.

property id

The identifier of the component.

The identifier is initially set to `-1`, and will become a valid value when the component is initialized.

With the default executor (*holoscan.gxf.GXFExecutor*), the identifier is set to the GXF component ID.

Returns

id

initialize(*self: holoscan.operators.holoviz._holoviz.HolovizOp*) None

Initialize the operator.

This method is called only once when the operator is created for the first time, and uses a light-weight initialization.

property name

The name of the operator.

property operator_type

The operator type.

holoscan.core.Operator.OperatorType enum representing the type of the operator. The two types currently implemented are native and GXF.

property resources

Resources associated with the operator.

setup(*self: holoscan.operators.holoviz._holoviz.HolovizOp*, *spec: holoscan.core._core.OperatorSpec*) None

Define the operator specification.

Parameters

spec

The operator specification.

property spec

The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

start(*self: holoscan.core._core.Operator*)   None

Operator start method.

stop(*self: holoscan.core._core.Operator*)   None

Operator stop method.

class holoscan.operators.InferenceOp

Bases: `holoscan.core._core.Operator`

Inference operator.

==Named Inputs==

receiversmulti-receiver accepting nvidia::gxf::Tensor(s)

Any number of upstream ports may be connected to this `receivers` port. The operator will search across all messages for tensors matching those specified in `in_tensor_names`. These are the set of input tensors used by the models in `inference_map`.

==Named Outputs==

transmitternvidia::gxf::Tensor(s)

A message containing tensors corresponding to the inference results from all models will be emitted. The names of the tensors transmitted correspond to those in `out_tensor_names`.

==Device Memory Requirements==

When using this operator with a `holoscan.resources.BlockMemoryPool`, `num_blocks` must be greater than or equal to the number of output tensors that will be produced. The `block_size` in bytes must be greater than or equal to the largest output tensor (in bytes). If `output_on_cuda` is `True`, the blocks should be in device memory (`storage_type=1`), otherwise they should be CUDA pinned host memory (`storage_type=0`).

For more details on `InferenceOp` parameters, see [Customizing the Inference Operator](https://docs.nvidia.com/holoscan/sdk-user-guide/examples/byom.html#customizing-the-inference-operator) or refer to [Inference](https://docs.nvidia.com/holoscan/sdk-user-guide/inference.html).

Parameters

fragment

The fragment that the operator belongs to.

backend

Backend to use for inference. Set `"trt"` for TensorRT, `"torch"` for LibTorch and `"onnxrt"` for the ONNX runtime.

allocator

Memory allocator to use for the output.

inference_map

Tensor to model map.

model_path_map

Path to the ONNX model to be loaded.

pre_processor_map

Pre processed data to model map.

device_map

Mapping of model to GPU ID for inference.

temporal_map

Mapping of model to frame delay for inference.

backend_map

Mapping of model to backend type for inference. Backend options: `"trt"` or `"torch"`

in_tensor_names

Input tensors.

out_tensor_names

Output tensors.

infer_on_cpu

Whether to run the computation on the CPU instead of GPU. Default value is `False`.

parallel_inference

Whether to enable parallel execution. Default value is `True`.

input_on_cuda

Whether the input buffer is on the GPU. Default value is `True`.

output_on_cuda

Whether the output buffer is on the GPU. Default value is `True`.

transmit_on_cuda

Whether to transmit the message on the GPU. Default value is `True`.

enable_fp16

Use 16-bit floating point computations. Default value is `False`.

is_engine_path

Whether the input model path mapping is for trt engine files. Default value is `False`.

cuda_stream_pool

`holoscan.resources.CudaStreamPool` instance to allocate CUDA streams. Default value is `None`.

name

The name of the operator. Default value is `"inference"`.

Attributes

`args`	The list of arguments associated with the component.
`conditions`	Conditions associated with the operator.
`description`	YAML formatted string describing the operator.
`fragment`	The fragment (`holoscan.core.Fragment`) that the operator belongs to.
`id`	The identifier of the component.
`name`	The name of the operator.
`operator_type`	The operator type.
`resources`	Resources associated with the operator.
`spec`	The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

Methods

add_arg(*args, **kwargs)	Overloaded function.
compute(self, arg0, arg1, arg2)	Operator compute method.
initialize(self)	Initialize the operator.
setup(self, spec)	Define the operator specification.
start(self)	Operator start method.
stop(self)	Operator stop method.

DataMap	
DataVecMap	
OperatorType	

class DataMap

Bases: pybind11_builtins.pybind11_object

Methods

get_map (self)	
insert (self)	

__init__(*self: holoscan.operators.inference._inference.InferenceOp.DataMap*) None

get_map(*self: holoscan.operators.inference._inference.InferenceOp.DataMap*) Dict[str, str]

insert(*self: holoscan.operators.inference._inference.InferenceOp.DataMap*) Dict[str, str]

class DataVecMap

Bases: pybind11_builtins.pybind11_object

Methods

get_map (self)	
insert (self)	

__init__(*self: holoscan.operators.inference._inference.InferenceOp.DataVecMap*) None

get_map(*self: holoscan.operators.inference._inference.InferenceOp.DataVecMap*) Dict[str, List[str]]

insert(*self: holoscan.operators.inference._inference.InferenceOp.DataVecMap*) Dict[str, List[str]]

class OperatorType

Bases: `pybind11_builtins.pybind11_object`

Enum class for operator types used by the executor.

- NATIVE: Native operator.
- GXF: GXF operator.
- VIRTUAL: Virtual operator. (for internal use, not intended for use by application authors)

Members:

NATIVE

GXF

VIRTUAL

Attributes

`name`	

value	

GXF = *<OperatorType.GXF: 1>*

NATIVE = *<OperatorType.NATIVE: 0>*

VIRTUAL = *<OperatorType.VIRTUAL: 2>*

__init__(*self: holoscan.core._core.Operator.OperatorType*, *value: int*) None

property name

property value

__init__(*self: holoscan.operators.inference._inference.InferenceOp*, *fragment: holoscan.core._core.Fragment*, **args*, *backend: str*, *allocator:*

holoscan.resources._resources.Allocator, inference_map: dict, model_path_map: dict, pre_processor_map: dict, device_map: dict = {}, temporal_map: dict = {}, backend_map: dict = {}, in_tensor_names: List[str] = [], out_tensor_names: List[str] = [], infer_on_cpu: bool = False, parallel_inference: bool = True, input_on_cuda: bool = True, output_on_cuda: bool = True, transmit_on_cuda: bool = True, enable_fp16: bool = False, is_engine_path: bool = False, cuda_stream_pool: holoscan.resources._resources.CudaStreamPool = None, name: str = 'inference') None

Inference operator.

==Named Inputs==

receiversmulti-receiver accepting nvidia::gxf::Tensor(s)

Any number of upstream ports may be connected to this `receivers` port. The operator will search across all messages for tensors matching those specified in `in_tensor_names`. These are the set of input tensors used by the models in `inference_map`.

==Named Outputs==

transmitternvidia::gxf::Tensor(s)

A message containing tensors corresponding to the inference results from all models will be emitted. The names of the tensors transmitted correspond to those in `out_tensor_names`.

==Device Memory Requirements==

When using this operator with a `holoscan.resources.BlockMemoryPool`, `num_blocks` must be greater than or equal to the number of output tensors that will be produced. The `block_size` in bytes must be greater than or equal to the largest output tensor (in bytes). If `output_on_cuda` is `True`, the blocks should be in device memory (`storage_type=1`), otherwise they should be CUDA pinned host memory (`storage_type=0`).

For more details on `InferenceOp` parameters, see [Customizing the Inference Operator](https://docs.nvidia.com/holoscan/sdk-user-guide/examples/byom.html#customizing-the-inference-operator) or refer to [Inference](https://docs.nvidia.com/holoscan/sdk-user-guide/inference.html).

Parameters

fragment

The fragment that the operator belongs to.

backend

Backend to use for inference. Set `"trt"` for TensorRT, `"torch"` for LibTorch and `"onnxrt"` for the ONNX runtime.

allocator

Memory allocator to use for the output.

inference_map

Tensor to model map.

model_path_map

Path to the ONNX model to be loaded.

pre_processor_map

Pre processed data to model map.

device_map

Mapping of model to GPU ID for inference.

temporal_map

Mapping of model to frame delay for inference.

backend_map

Mapping of model to backend type for inference. Backend options: `"trt"` or `"torch"`

in_tensor_names

Input tensors.

out_tensor_names

Output tensors.

infer_on_cpu

Whether to run the computation on the CPU instead of GPU. Default value is `False`.

parallel_inference

Whether to enable parallel execution. Default value is `True`.

input_on_cuda

Whether the input buffer is on the GPU. Default value is `True`.

output_on_cuda

Whether the output buffer is on the GPU. Default value is `True`.

transmit_on_cuda

Whether to transmit the message on the GPU. Default value is `True`.

enable_fp16

Use 16-bit floating point computations. Default value is `False`.

is_engine_path

Whether the input model path mapping is for trt engine files. Default value is `False`.

cuda_stream_pool

`holoscan.resources.CudaStreamPool` instance to allocate CUDA streams. Default value is `None`.

name

The name of the operator. Default value is `"inference"`.

add_arg(**args*, ***kwargs*)

Overloaded function.

1. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Arg) -> None

Add an argument to the component.

2. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.ArgList) -> None

Add a list of arguments to the component.

3. add_arg(self: holoscan.core._core.Operator, **kwargs) -> None

Add arguments to the component via Python kwargs.

4. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Condition) -> None

5. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Resource) -> None

Add a condition or resource to the Operator.

This can be used to add a condition or resource to an operator after it has already been constructed.

Parameters

arg

The condition or resource to add.

property args

The list of arguments associated with the component.

Returns

arglist

compute(*self: holoscan.core._core.Operator*, *arg0: holoscan.core._core.InputContext*, *arg1: holoscan.core._core.OutputContext*, *arg2: holoscan.core._core.ExecutionContext*) None

Operator compute method. This method defines the primary computation to be executed by the operator.

property conditions

Conditions associated with the operator.

property description

YAML formatted string describing the operator.

property fragment

The fragment (`holoscan.core.Fragment`) that the operator belongs to.

property id

The identifier of the component.

The identifier is initially set to `-1`, and will become a valid value when the component is initialized.

With the default executor (*holoscan.gxf.GXFExecutor*), the identifier is set to the GXF component ID.

Returns

id

initialize(*self: holoscan.operators.inference._inference.InferenceOp*) None

Initialize the operator.

This method is called only once when the operator is created for the first time, and uses a light-weight initialization.

property name

The name of the operator.

property operator_type

The operator type.

holoscan.core.Operator.OperatorType enum representing the type of the operator. The two types currently implemented are native and GXF.

property resources

Resources associated with the operator.

setup(*self: holoscan.operators.inference._inference.InferenceOp*, *spec: holoscan.core._core.OperatorSpec*) None

Define the operator specification.

Parameters

spec

The operator specification.

property spec

The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

start(*self: holoscan.core._core.Operator*) None

Operator start method.

stop(*self: holoscan.core._core.Operator*) None

Operator stop method.

class holoscan.operators.InferenceProcessorOp

Bases: holoscan.core._core.Operator

Holoinfer Processing operator.

==Named Inputs==

receiversmulti-receiver accepting nvidia::gxf::Tensor(s)

Any number of upstream ports may be connected to this receivers port. The operator will search across all messages for tensors matching those specified in in_tensor_names. These are the set of input tensors used by the processing operations specified in process_map.

==Named Outputs==

transmitternvidia::gxf::Tensor(s)

A message containing tensors corresponding to the processed results from operations will be emitted. The names of the tensors transmitted correspond to those in out_tensor_names.

==Device Memory Requirements==

When using this operator with a holoscan.resources.BlockMemoryPool, num_blocks must be greater than or equal to the number of output tensors that will be produced. The block_size in bytes must be greater than or equal to the largest output tensor (in bytes). If output_on_cuda is True, the blocks should be in device memory (storage_type=1), otherwise they should be CUDA pinned host memory (storage_type=0).

Parameters

fragment

The fragment that the operator belongs to.

allocator

Memory allocator to use for the output.

process_operations

Operations in sequence on tensors.

processed_map

Input-output tensor mapping.

in_tensor_names

Names of input tensors in the order to be fed into the operator.

out_tensor_names

Names of output tensors in the order to be fed into the operator.

input_on_cuda

Whether the input buffer is on the GPU. Default value is `False`.

output_on_cuda

Whether the output buffer is on the GPU. Default value is `False`.

transmit_on_cuda

Whether to transmit the message on the GPU. Default value is `False`.

cuda_stream_pool

`holoscan.resources.CudaStreamPool` instance to allocate CUDA streams. Default value is `None`.

config_path

File path to the config file. Default value is `""`.

disable_transmitter

If `True`, disable the transmitter output port of the operator. Default value is `False`.

name

The name of the operator. Default value is `"postprocessor"`.

Attributes

`args`	The list of arguments associated with the component.
`conditions`	Conditions associated with the operator.
`description`	YAML formatted string describing the operator.
`fragment`	The fragment (`holoscan.core.Fragment`) that the operator belongs to.
`id`	The identifier of the component.
`name`	The name of the operator.
`operator_type`	The operator type.
`resources`	Resources associated with the operator.

spec	The operator spec (holoscan.core.OperatorSpec) associated with the operator.

Methods

add_arg (*args, **kwargs)	Overloaded function.
compute (self, arg0, arg1, arg2)	Operator compute method.
initialize (self)	Initialize the operator.
setup (self, spec)	Define the operator specification.
start (self)	Operator start method.
stop (self)	Operator stop method.

DataMap	
DataVecMap	
OperatorType	

class DataMap

Bases: pybind11_builtins.pybind11_object

Methods

get_map (self)	
insert (self)	

__init__(*self: holoscan.operators.inference_processor._inference_processor.InferenceProcessorOp*
None

get_map(*self: holoscan.operators.inference_processor._inference_processor.InferenceProcessorOp*
Dict[str, str]

insert(*self: holoscan.operators.inference_processor._inference_processor.InferenceProcessorOp*
Dict[str, str]

class DataVecMap

Bases: pybind11_builtins.pybind11_object

Methods

get_map (self)	
insert (self)	

__init__(*self: holoscan.operators.inference_processor._inference_processor.InferenceProcessorOp*
None

get_map(*self: holoscan.operators.inference_processor._inference_processor.InferenceProcessorOp*
Dict[str, List[str]]

insert(*self: holoscan.operators.inference_processor._inference_processor.InferenceProcessorOp*
Dict[str, List[str]]

class OperatorType

Bases: `pybind11_builtins.pybind11_object`

Enum class for operator types used by the executor.

- NATIVE: Native operator.
- GXF: GXF operator.
- VIRTUAL: Virtual operator. (for internal use, not intended for use by application authors)

Members:

NATIVE

GXF

VIRTUAL

Attributes

`name`	

value	

GXF = *<OperatorType.GXF: 1>*

NATIVE = *<OperatorType.NATIVE: 0>*

VIRTUAL = *<OperatorType.VIRTUAL: 2>*

__init__(*self: holoscan.core._core.Operator.OperatorType*, *value: int*)   None

property name

property value

__init__(*self: holoscan.operators.inference_processor._inference_processor.InferenceProcessorOp*, *fragment: holoscan.core._core.Fragment*, **args*, *allocator: holoscan.resources._resources.Allocator*, *process_operations: dict = {}*, *processed_map: dict = {}*, *in_tensor_names: List[str] = []*, *out_tensor_names: List[str] = []*, *input_on_cuda: bool = False*, *output_on_cuda: bool = False*, *transmit_on_cuda: bool = False*, *disable_transmitter: bool = False*, *cuda_stream_pool: holoscan.resources._resources.CudaStreamPool = None*, *config_path: str = ''*, *name: str = 'postprocessor'*)   None

Holoinfer Processing operator.

==Named Inputs==

receiversmulti-receiver accepting nvidia::gxf::Tensor(s)

Any number of upstream ports may be connected to this `receivers` port. The operator will search across all messages for tensors matching those specified in `in_tensor_names`. These are the set of input tensors used by the processing operations specified in `process_map`.

==Named Outputs==

transmitternvidia::gxf::Tensor(s)

A message containing tensors corresponding to the processed results from operations will be emitted. The names of the tensors transmitted correspond to those in `out_tensor_names`.

==Device Memory Requirements==

When using this operator with a `holoscan.resources.BlockMemoryPool`, `num_blocks` must be greater than or equal to the number of output tensors that will be produced. The `block_size` in bytes must be greater than or equal to the largest output tensor (in bytes). If `output_on_cuda` is `True`, the blocks should be in device memory (`storage_type=1`), otherwise they should be CUDA pinned host memory (`storage_type=0`).

Parameters

fragment

The fragment that the operator belongs to.

allocator

Memory allocator to use for the output.

process_operations

Operations in sequence on tensors.

processed_map

Input-output tensor mapping.

in_tensor_names

Names of input tensors in the order to be fed into the operator.

out_tensor_names

Names of output tensors in the order to be fed into the operator.

input_on_cuda

Whether the input buffer is on the GPU. Default value is `False`.

output_on_cuda

Whether the output buffer is on the GPU. Default value is `False`.

transmit_on_cuda

Whether to transmit the message on the GPU. Default value is `False`.

cuda_stream_pool

`holoscan.resources.CudaStreamPool` instance to allocate CUDA streams. Default value is `None`.

config_path

File path to the config file. Default value is `""`.

disable_transmitter

If `True`, disable the transmitter output port of the operator. Default value is `False`.

name

The name of the operator. Default value is `"postprocessor"`.

add_arg(**args*, ***kwargs*)

Overloaded function.

1. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Arg) -> None

Add an argument to the component.

2. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.ArgList) -> None

Add a list of arguments to the component.

3. add_arg(self: holoscan.core._core.Operator, **kwargs) -> None

Add arguments to the component via Python kwargs.

4. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Condition) -> None

5. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Resource) -> None

Add a condition or resource to the Operator.

This can be used to add a condition or resource to an operator after it has already been constructed.

Parameters

arg

The condition or resource to add.

property args

The list of arguments associated with the component.

Returns

arglist

compute(*self: holoscan.core._core.Operator*, *arg0: holoscan.core._core.InputContext*, *arg1: holoscan.core._core.OutputContext*, *arg2: holoscan.core._core.ExecutionContext*) None

Operator compute method. This method defines the primary computation to be executed by the operator.

property conditions

Conditions associated with the operator.

property description

YAML formatted string describing the operator.

property fragment

The fragment (`holoscan.core.Fragment`) that the operator belongs to.

property id

The identifier of the component.

The identifier is initially set to `-1`, and will become a valid value when the component is initialized.

With the default executor (*holoscan.gxf.GXFExecutor*), the identifier is set to the GXF component ID.

Returns

id

initialize(*self: holoscan.operators.inference_processor._inference_processor.InferenceProcessorOp*) None

Initialize the operator.

This method is called only once when the operator is created for the first time, and uses a light-weight initialization.

property name

The name of the operator.

property operator_type

The operator type.

holoscan.core.Operator.OperatorType enum representing the type of the operator. The two types currently implemented are native and GXF.

property resources

Resources associated with the operator.

setup(*self: holoscan.operators.inference_processor._inference_processor.InferenceProcessorOp*, *spec: holoscan.core._core.OperatorSpec*)   None

Define the operator specification.

Parameters

spec

The operator specification.

property spec

The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

start(*self: holoscan.core._core.Operator*) None

Operator start method.

stop(*self: holoscan.core._core.Operator*) None

Operator stop method.

class holoscan.operators.NTV2Channel

Bases: `pybind11_builtins.pybind11_object`

Members:

NTV2_CHANNEL1

NTV2_CHANNEL2

NTV2_CHANNEL3

NTV2_CHANNEL4

NTV2_CHANNEL5

NTV2_CHANNEL6

NTV2_CHANNEL7

NTV2_CHANNEL8

NTV2_MAX_NUM_CHANNELS

NTV2_CHANNEL_INVALID

Attributes

name	

value	

NTV2_CHANNEL1 = *<NTV2Channel.NTV2_CHANNEL1: 0>*

NTV2_CHANNEL2 = *<NTV2Channel.NTV2_CHANNEL2: 1>*

NTV2_CHANNEL3 = *<NTV2Channel.NTV2_CHANNEL3: 2>*

NTV2_CHANNEL4 = *<NTV2Channel.NTV2_CHANNEL4: 3>*

NTV2_CHANNEL5 = *<NTV2Channel.NTV2_CHANNEL5: 4>*

NTV2_CHANNEL6 = *<NTV2Channel.NTV2_CHANNEL6: 5>*

NTV2_CHANNEL7 = *<NTV2Channel.NTV2_CHANNEL7: 6>*

NTV2_CHANNEL8 = *<NTV2Channel.NTV2_CHANNEL8: 7>*

NTV2_CHANNEL_INVALID = *<NTV2Channel.NTV2_MAX_NUM_CHANNELS: 8>*

NTV2_MAX_NUM_CHANNELS = *<NTV2Channel.NTV2_MAX_NUM_CHANNELS: 8>*

__init__(*self: holoscan.operators.aja_source._aja_source.NTV2Channel*, *value: int*) None

property name

property value

class holoscan.operators.PingRxOp(*fragment*, **args*, ***kwargs*)

Bases: holoscan.core.Operator

Simple receiver operator.

This is an example of a native operator with one input port. On each tick, it receives an integer from the “in” port.

==Named Inputs==

inany

A received value.

Attributes

`args`	The list of arguments associated with the component.
`conditions`	Conditions associated with the operator.
`description`	YAML formatted string describing the operator.
`fragment`	The fragment (`holoscan.core.Fragment`) that the operator belongs to.
`id`	The identifier of the component.
`name`	The name of the operator.
`operator_type`	The operator type.
`resources`	Resources associated with the operator.
`spec`	The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

Methods

`add_arg` (*args,	Overloaded function.

**kwargs)	
`compute`(op_input, op_output, context)	Default implementation of compute
`initialize`()	Default implementation of initialize
`setup`(spec)	Default implementation of setup method.
`start`()	Default implementation of start
`stop`()	Default implementation of stop

OperatorType	

class OperatorType

Bases: `pybind11_builtins.pybind11_object`

Enum class for operator types used by the executor.

- NATIVE: Native operator.
- GXF: GXF operator.
- VIRTUAL: Virtual operator. (for internal use, not intended for use by application authors)

Members:

NATIVE

GXF

VIRTUAL

Attributes

name	

value	

GXF = *<OperatorType.GXF: 1>*

NATIVE = *<OperatorType.NATIVE: 0>*

VIRTUAL = *<OperatorType.VIRTUAL: 2>*

__init__(*self: holoscan.core._core.Operator.OperatorType*, *value: int*)   None

property name

property value

__init__(*self: holoscan.core._core.Operator*, *arg0: object*, *arg1: holoscan::Fragment*, **args*, ***kwargs*)   None

Operator class.

Can be initialized with any number of Python positional and keyword arguments.

If a *name* keyword argument is provided, it must be a *str* and will be used to set the name of the operator.

Condition classes will be added to `self.conditions`, *Resource* classes will be added to `self.resources`, and any other arguments will be cast from a Python argument type to a C++ *Arg* and stored in `self.args`. (For details on how the casting is done, see the *py_object_to_arg* utility). When a Condition or Resource is provided via a kwarg, it's name will be automatically be updated to the name of the kwarg.

Parameters

fragment

The *holoscan.core.Fragment* (or *holoscan.core.Application*) to which this Operator will belong.

***args**

Positional arguments.

****kwargs**

Keyword arguments.

Raises

RuntimeError

If *name* kwarg is provided, but is not of *str* type. If multiple arguments of type *Fragment* are provided. If any other arguments cannot be converted to *Arg* type via *py_object_to_arg*.

add_arg(**args*, ***kwargs*)

Overloaded function.

1. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Arg) -> None

Add an argument to the component.

2. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.ArgList) -> None

Add a list of arguments to the component.

3. add_arg(self: holoscan.core._core.Operator, **kwargs) -> None

Add arguments to the component via Python kwargs.

4. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Condition) -> None

5. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Resource) -> None

Add a condition or resource to the Operator.

This can be used to add a condition or resource to an operator after it has already been constructed.

Parameters

arg

The condition or resource to add.

property args

The list of arguments associated with the component.

Returns

arglist

compute(*op_input*, *op_output*, *context*)

Default implementation of compute

property conditions

Conditions associated with the operator.

property description

YAML formatted string describing the operator.

property fragment

The fragment (`holoscan.core.Fragment`) that the operator belongs to.

property id

The identifier of the component.

The identifier is initially set to `-1`, and will become a valid value when the component is initialized.

With the default executor (*holoscan.gxf.GXFExecutor*), the identifier is set to the GXF component ID.

Returns

id

initialize()

Default implementation of initialize

property name

The name of the operator.

property operator_type

The operator type.

holoscan.core.Operator.OperatorType enum representing the type of the operator. The two types currently implemented are native and GXF.

property resources

Resources associated with the operator.

setup(*spec: holoscan.core._core.PyOperatorSpec*)

Default implementation of setup method.

property spec

The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

start()

Default implementation of start

stop()

Default implementation of stop

class holoscan.operators.PingTxOp(*fragment*, **args*, ***kwargs*)

Bases: holoscan.core.Operator

Simple transmitter operator.

On each tick, it transmits an integer to the “out” port.

==Named Outputs==

outint

An index value that increments by one on each call to *compute*. The starting value is 1.

Attributes

args	The list of arguments associated with the component.
conditions	Conditions associated with the operator.
description	YAML formatted string describing the operator.
fragment	The fragment (holoscan.core.Fragment) that the operator belongs to.
id	The identifier of the component.
name	The name of the operator.
operator_type	The operator type.

`resources`	Resources associated with the operator.
`spec`	The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

Methods

`add_arg` (*args, **kwargs)	Overloaded function.
`compute` (op_input, op_output, context)	Default implementation of compute
`initialize` ()	Default implementation of initialize
`setup` (spec)	Default implementation of setup method.
`start` ()	Default implementation of start
`stop` ()	Default implementation of stop

OperatorType	

class OperatorType

Bases: `pybind11_builtins.pybind11_object`

Enum class for operator types used by the executor.

- NATIVE: Native operator.
- GXF: GXF operator.
- VIRTUAL: Virtual operator. (for internal use, not intended for use by application authors)

Members:

NATIVE

GXF

VIRTUAL

Attributes

name	

value	

GXF = *<OperatorType.GXF: 1>*

NATIVE = *<OperatorType.NATIVE: 0>*

VIRTUAL = *<OperatorType.VIRTUAL: 2>*

__init__(*self: holoscan.core._core.Operator.OperatorType*, *value: int*)   None

property name

property value

__init__(*self: holoscan.core._core.Operator*, *arg0: object*, *arg1: holoscan::Fragment*, **args*, ***kwargs*)   None

Operator class.

Can be initialized with any number of Python positional and keyword arguments.

If a *name* keyword argument is provided, it must be a *str* and will be used to set the name of the operator.

Condition classes will be added to `self.conditions`, *Resource* classes will be added to `self.resources`, and any other arguments will be cast from a Python argument type to a C++ *Arg* and stored in `self.args`. (For details on how the casting is done, see the *py_object_to_arg* utility). When a Condition or Resource is provided via a kwarg, it's name will be automatically be updated to the name of the kwarg.

Parameters

fragment

The *holoscan.core.Fragment* (or *holoscan.core.Application*) to which this Operator will belong.

***args**

Positional arguments.

****kwargs**

Keyword arguments.

Raises

RuntimeError

If *name* kwarg is provided, but is not of *str* type. If multiple arguments of type *Fragment* are provided. If any other arguments cannot be converted to *Arg* type via *py_object_to_arg*.

add_arg(**args*, ***kwargs*)

Overloaded function.

1. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Arg) -> None

Add an argument to the component.

2. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.ArgList) -> None

Add a list of arguments to the component.

3. add_arg(self: holoscan.core._core.Operator, **kwargs) -> None

Add arguments to the component via Python kwargs.

4. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Condition) -> None

5. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Resource) -> None

Add a condition or resource to the Operator.

This can be used to add a condition or resource to an operator after it has already been constructed.

Parameters

arg

The condition or resource to add.

property args

The list of arguments associated with the component.

Returns

arglist

compute(*op_input*, *op_output*, *context*)

Default implementation of compute

property conditions

Conditions associated with the operator.

property description

YAML formatted string describing the operator.

property fragment

The fragment (`holoscan.core.Fragment`) that the operator belongs to.

property id

The identifier of the component.

The identifier is initially set to `-1`, and will become a valid value when the component is initialized.

With the default executor (*holoscan.gxf.GXFExecutor*), the identifier is set to the GXF component ID.

Returns

id

initialize()

Default implementation of initialize

property name

The name of the operator.

property operator_type

The operator type.

holoscan.core.Operator.OperatorType enum representing the type of the operator. The two types currently implemented are native and GXF.

property resources

Resources associated with the operator.

setup(*spec: holoscan.core._core.PyOperatorSpec*)

Default implementation of setup method.

property spec

The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

start()

Default implementation of start

stop()

Default implementation of stop

class holoscan.operators.SegmentationPostprocessorOp

Bases: `holoscan.core._core.Operator`

Operator carrying out post-processing operations on segmentation outputs.

==Named Inputs==

in_tensornvidia::gxf::Tensor

Expects a message containing a 32-bit floating point device tensor with name `in_tensor_name`. The expected data layout of this tensor is HWC, NCHW or NHWC format as specified via `data_format`.

==Named Outputs==

out_tensornvidia::gxf::Tensor

Emits a message containing a device tensor named “out_tensor” that contains the segmentation labels. This tensor will have unsigned 8-bit integer data type and shape (H, W, 1).

==Device Memory Requirements==

When used with a `holoscan.resources.BlockMemoryPool`, this operator requires only a single device memory block (`storage_type=1`) of size `height * width` bytes.

Parameters

fragment

The fragment that the operator belongs to.

allocator

Memory allocator to use for the output.

in_tensor_name

Name of the input tensor. Default value is `""`.

network_output_type

Network output type (e.g. ‘softmax’). Default value is `"softmax"`.

data_format

Data format of network output. Default value is `"hwc"`.

cuda_stream_pool

`holoscan.resources.CudaStreamPool` instance to allocate CUDA streams. Default value is `None`.

name

The name of the operator. Default value is `"segmentation_postprocessor"`.

Attributes

`args`	The list of arguments associated with the component.

`conditions`	Conditions associated with the operator.
`description`	YAML formatted string describing the operator.
`fragment`	The fragment (`holoscan.core.Fragment`) that the operator belongs to.
`id`	The identifier of the component.
`name`	The name of the operator.
`operator_type`	The operator type.
`resources`	Resources associated with the operator.
`spec`	The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

Methods

`add_arg` (*args, **kwargs)	Overloaded function.
`compute` (self, arg0, arg1, arg2)	Operator compute method.
`initialize`	Operator initialization method.

(self)	
`setup` (self, spec)	Define the operator specification.
`start` (self)	Operator start method.
`stop` (self)	Operator stop method.

OperatorType	

class OperatorType

Bases: `pybind11_builtins.pybind11_object`

Enum class for operator types used by the executor.

- NATIVE: Native operator.
- GXF: GXF operator.
- VIRTUAL: Virtual operator. (for internal use, not intended for use by application authors)

Members:

NATIVE

GXF

VIRTUAL

Attributes

name	

value	

GXF = *<OperatorType.GXF: 1>*

NATIVE = *<OperatorType.NATIVE: 0>*

VIRTUAL = *<OperatorType.VIRTUAL: 2>*

__init__(*self: holoscan.core._core.Operator.OperatorType*, *value: int*) None

property name

property value

__init__(*self: holoscan.operators.segmentation_postprocessor._segmentation_postprocessor.Segmenta* *fragment: holoscan.core._core.Fragment*, **args*, *allocator: holoscan.resources._resources.Allocator*, *in_tensor_name: str = ''*, *network_output_type: str = 'softmax'*, *data_format: str = 'hwc'*, *cuda_stream_pool: holoscan.resources._resources.CudaStreamPool = None*, *name: str = 'segmentation_postprocessor'*) None

Operator carrying out post-processing operations on segmentation outputs.

==Named Inputs==

in_tensornvidia::gxf::Tensor

Expects a message containing a 32-bit floating point device tensor with name `in_tensor_name`. The expected data layout of this tensor is HWC, NCHW or NHWC format as specified via `data_format`.

==Named Outputs==

out_tensornvidia::gxf::Tensor

Emits a message containing a device tensor named “out_tensor” that contains the segmentation labels. This tensor will have unsigned 8-bit integer data type and shape (H, W, 1).

==Device Memory Requirements==

When used with a `holoscan.resources.BlockMemoryPool`, this operator requires only a single device memory block (`storage_type=1`) of size `height * width` bytes.

Parameters

fragment

The fragment that the operator belongs to.

allocator

Memory allocator to use for the output.

in_tensor_name

Name of the input tensor. Default value is `""`.

network_output_type

Network output type (e.g. ‘softmax’). Default value is `"softmax"`.

data_format

Data format of network output. Default value is `"hwc"`.

cuda_stream_pool

`holoscan.resources.CudaStreamPool` instance to allocate CUDA streams. Default value is `None`.

name

The name of the operator. Default value is `"segmentation_postprocessor"`.

add_arg(**args*, ***kwargs*)

Overloaded function.

1. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Arg) -> None

Add an argument to the component.

2. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.ArgList) -> None

Add a list of arguments to the component.

3. add_arg(self: holoscan.core._core.Operator, **kwargs) -> None

Add arguments to the component via Python kwargs.

4. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Condition) -> None

5. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Resource) -> None

Add a condition or resource to the Operator.

This can be used to add a condition or resource to an operator after it has already been constructed.

Parameters

arg

The condition or resource to add.

property args

The list of arguments associated with the component.

Returns

arglist

compute(*self: holoscan.core._core.Operator*, *arg0: holoscan.core._core.InputContext*, *arg1: holoscan.core._core.OutputContext*, *arg2: holoscan.core._core.ExecutionContext*) None

Operator compute method. This method defines the primary computation to be executed by the operator.

property conditions

Conditions associated with the operator.

property description

YAML formatted string describing the operator.

property fragment

The fragment (`holoscan.core.Fragment`) that the operator belongs to.

property id

The identifier of the component.

The identifier is initially set to `-1`, and will become a valid value when the component is initialized.

With the default executor (*holoscan.gxf.GXFExecutor*), the identifier is set to the GXF component ID.

Returns

id

initialize(*self: holoscan.core._core.Operator*) None

Operator initialization method.

property name

The name of the operator.

property operator_type

The operator type.

holoscan.core.Operator.OperatorType enum representing the type of the operator. The two types currently implemented are native and GXF.

property resources

Resources associated with the operator.

setup(*self: holoscan.operators.segmentation_postprocessor._segmentation_postprocessor.Segmenta* *spec: holoscan.core._core.OperatorSpec*) None

Define the operator specification.

Parameters

spec

The operator specification.

property spec

The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

start(*self: holoscan.core._core.Operator*) None

Operator start method.

stop(*self: holoscan.core._core.Operator*) None

Operator stop method.

class holoscan.operators.V4L2VideoCaptureOp

Bases: `holoscan.core._core.Operator`

Operator to get a video stream from a V4L2 source.

https://www.kernel.org/doc/html/v4.9/media/uapi/v4l/v4l2.html

Inputs a video stream from a V4L2 node, including USB cameras and HDMI IN.

- Input stream is on host. If no pixel format is specified in the yaml configuration file, the pixel format will be automatically selected. However, only `AB24`, `YUYV`, and `MJPG` are then supported. If a pixel format is specified in the yaml file, then this format will be used. However, note that the operator then expects that this format can be encoded as RGBA32. If not, the behavior is undefined.

- Output stream is on host. Always RGBA32 at this time.

Use `holoscan.operators.FormatConverterOp` to move data from the host to a GPU device.

==Named Outputs==

signalnvidia::gxf::VideoBuffer

A message containing a video buffer on the host with format GXF_VIDEO_FORMAT_RGBA.

==Device Memory Requirements==

When using this operator with a `holoscan.resources.BlockMemoryPool`, a single device memory block is needed (`storage_type=1`). The size of this memory block can be determined by rounding the width and height up to the nearest even size and then padding the rows as needed so that the row stride is a multiple of 256 bytes. C++ code to calculate the block size is as follows:

```
def get_block_size(height, width): height_even = height + (height & 1)
width_even = width + (width & 1) row_bytes = width_even * 4; # 4 bytes per
```

> pixel for 8-bit RGBA row_stride = (row_bytes % 256 == 0) ? row_bytes : ((row_bytes // 256 + 1) * 256) return height_even * row_stride

Parameters

fragment

The fragment that the operator belongs to.

allocator

Memory allocator to use for the output.

device

The device to target (e.g. “/dev/video0” for device 0). Default value is `"/dev/video0"`.

width

Width of the video stream. Default value is `0`.

height

Height of the video stream. Default value is `0`.

num_buffers

Number of V4L2 buffers to use. Default value is `4`.

pixel_format

Video stream pixel format (little endian four character code (fourcc)). Default value is `"auto"`.

name

The name of the operator. Default value is `"v4l2_video_capture"`.

exposure_time

Exposure time of the camera sensor in multiples of 100 μs (e.g. setting exposure_time to 100 is 10 ms). Default: auto exposure, or camera sensor default. Use *v4l2-ctl -d /dev/<your_device> -L* for a range of values supported by your device. When not set by the user, V4L2_CID_EXPOSURE_AUTO is set to V4L2_EXPOSURE_AUTO, or to V4L2_EXPOSURE_APERTURE_PRIORITY if the former is not supported. When set by the user, V4L2_CID_EXPOSURE_AUTO is set to V4L2_EXPOSURE_SHUTTER_PRIORITY, or to V4L2_EXPOSURE_MANUAL if the former is not supported. The provided value is then used to set V4L2_CID_EXPOSURE_ABSOLUTE.

gain

Gain of the camera sensor. Default: auto gain, or camera sensor default. Use *v4l2-ctl -d /dev/<your_device> -L* for a range of values supported by your device. When not set by the user, V4L2_CID_AUTOGAIN is set to true (if supported). When set by the user, V4L2_CID_AUTOGAIN is set to false (if supported). The provided value is then used to set V4L2_CID_GAIN.

Attributes

`args`	The list of arguments associated with the component.
`conditions`	Conditions associated with the operator.
`description`	YAML formatted string describing the operator.
`fragment`	The fragment (`holoscan.core.Fragment`) that the operator belongs to.
`id`	The identifier of the component.
`name`	The name of the operator.
`operator_type`	The operator type.
`resources`	Resources associated with the operator.

spec	The operator spec (holoscan.core.OperatorSpec) associated with the operator.

Methods

add_arg (*args, **kwargs)	Overloaded function.
compute (self, arg0, arg1, arg2)	Operator compute method.
initialize (self)	Initialize the operator.
setup (self, spec)	Define the operator specification.
start (self)	Operator start method.
stop (self)	Operator stop method.

OperatorType	

class OperatorType

Bases: pybind11_builtins.pybind11_object

Enum class for operator types used by the executor.

- NATIVE: Native operator.
- GXF: GXF operator.
- VIRTUAL: Virtual operator. (for internal use, not intended for use by application authors)

Members:

NATIVE

GXF

VIRTUAL

Attributes

name	

value	

GXF = *<OperatorType.GXF: 1>*

NATIVE = *<OperatorType.NATIVE: 0>*

VIRTUAL = *<OperatorType.VIRTUAL: 2>*

__init__(*self: holoscan.core._core.Operator.OperatorType*, *value: int*) None

property name

property value

__init__(*self: holoscan.operators.v4l2_video_capture._v4l2_video_capture.V4L2VideoCaptureOp*, *fragment: holoscan.core._core.Fragment*, **args*, *allocator: holoscan.resources._resources.Allocator*, *device: str = '0'*, *width: int = 0*, *height: int = 0*, *num_buffers: int = 4*, *pixel_format: str = 'auto'*, *name: str = 'v4l2_video_capture'*, *exposure_time: Optional[int] = None*, *gain: Optional[int] = None*) None

Operator to get a video stream from a V4L2 source.

https://www.kernel.org/doc/html/v4.9/media/uapi/v4l/v4l2.html

Inputs a video stream from a V4L2 node, including USB cameras and HDMI IN.

- Input stream is on host. If no pixel format is specified in the yaml configuration file, the pixel format will be automatically selected. However, only `AB24`, `YUYV`, and `MJPG` are then supported. If a pixel format is specified in the yaml file, then this format will be used. However, note that the operator then expects that this format can be encoded as RGBA32. If not, the behavior is undefined.
- Output stream is on host. Always RGBA32 at this time.

Use `holoscan.operators.FormatConverterOp` to move data from the host to a GPU device.

==Named Outputs==

signalnvidia::gxf::VideoBuffer

A message containing a video buffer on the host with format GXF_VIDEO_FORMAT_RGBA.

==Device Memory Requirements==

When using this operator with a `holoscan.resources.BlockMemoryPool`, a single device memory block is needed (`storage_type=1`). The size of this memory block can be determined by rounding the width and height up to the nearest even size and then padding the rows as needed so that the row stride is a multiple of 256 bytes. C++ code to calculate the block size is as follows:

```
def get_block_size(height, width): height_even = height + (height & 1)
width_even = width + (width & 1) row_bytes = width_even * 4; # 4 bytes
```

> per pixel for 8-bit RGBA row_stride = (row_bytes % 256 == 0) ? row_bytes : ((row_bytes // 256 + 1) * 256) return height_even * row_stride

Parameters

fragment

The fragment that the operator belongs to.

allocator

Memory allocator to use for the output.

device

The device to target (e.g. “/dev/video0” for device 0). Default value is `"/dev/video0"`.

width

Width of the video stream. Default value is `0`.

height

Height of the video stream. Default value is `0`.

num_buffers

Number of V4L2 buffers to use. Default value is `4`.

pixel_format

Video stream pixel format (little endian four character code (fourcc)). Default value is `"auto"`.

name

The name of the operator. Default value is `"v4l2_video_capture"`.

exposure_time

Exposure time of the camera sensor in multiples of 100 μs (e.g. setting exposure_time to 100 is 10 ms). Default: auto exposure, or camera sensor default. Use *v4l2-ctl -d /dev/<your_device> -L* for a range of values supported by your device. When not set by the user, V4L2_CID_EXPOSURE_AUTO is set to V4L2_EXPOSURE_AUTO, or to V4L2_EXPOSURE_APERTURE_PRIORITY if the former is not supported. When set by the user, V4L2_CID_EXPOSURE_AUTO is set to V4L2_EXPOSURE_SHUTTER_PRIORITY, or to V4L2_EXPOSURE_MANUAL if the former is not supported. The provided value is then used to set V4L2_CID_EXPOSURE_ABSOLUTE.

gain

Gain of the camera sensor. Default: auto gain, or camera sensor default. Use *v4l2-ctl -d /dev/<your_device> -L* for a range of values supported by your device. When not set by the user, V4L2_CID_AUTOGAIN is set to true (if supported). When set by the user, V4L2_CID_AUTOGAIN is set to false (if supported). The provided value is then used to set V4L2_CID_GAIN.

add_arg(**args*, ***kwargs*)

Overloaded function.

1. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Arg) -> None

Add an argument to the component.

2. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.ArgList) -> None

Add a list of arguments to the component.

3. add_arg(self: holoscan.core._core.Operator, **kwargs) -> None

Add arguments to the component via Python kwargs.

4. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Condition) -> None

5. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Resource) -> None

Add a condition or resource to the Operator.

This can be used to add a condition or resource to an operator after it has already been constructed.

Parameters

arg

The condition or resource to add.

property args

The list of arguments associated with the component.

Returns

arglist

compute(*self: holoscan.core._core.Operator*, *arg0: holoscan.core._core.InputContext*, *arg1: holoscan.core._core.OutputContext*, *arg2: holoscan.core._core.ExecutionContext*) None

Operator compute method. This method defines the primary computation to be executed by the operator.

property conditions

Conditions associated with the operator.

property description

YAML formatted string describing the operator.

property fragment

The fragment (`holoscan.core.Fragment`) that the operator belongs to.

property id

The identifier of the component.

The identifier is initially set to `-1`, and will become a valid value when the component is initialized.

With the default executor (*holoscan.gxf.GXFExecutor*), the identifier is set to the GXF component ID.

Returns

id

initialize(*self: holoscan.operators.v4l2_video_capture._v4l2_video_capture.V4L2VideoCaptureOp*) None

Initialize the operator.

This method is called only once when the operator is created for the first time, and uses a light-weight initialization.

property name

The name of the operator.

property operator_type

The operator type.

holoscan.core.Operator.OperatorType enum representing the type of the operator. The two types currently implemented are native and GXF.

property resources

Resources associated with the operator.

setup(*self: holoscan.operators.v4l2_video_capture._v4l2_video_capture.V4L2VideoCaptureOp*, *spec: holoscan.core._core.OperatorSpec*) None

Define the operator specification.

Parameters

spec : `holoscan.core.OperatorSpec`

The operator specification.

property spec

The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

start(*self: holoscan.core._core.Operator*) None

Operator start method.

stop(*self: holoscan.core._core.Operator*) None

Operator stop method.

class holoscan.operators.VideoStreamRecorderOp

Bases: `holoscan.core._core.Operator`

Operator class to record a video stream to a file.

==Named Inputs==

inputnvidia::gxf::Tensor

A message containing a video frame to serialize to disk. The input tensor can be on either the CPU or GPU. This data location will be recorded as part of the metadata serialized to disk and if the data is later read back in via *VideoStreamReplayerOp*, the tensor output of that operator will be on the same device (CPU or GPU).

Parameters

fragment

The fragment that the operator belongs to.

directory

Directory path for storing files.

basename

User specified file name without extension.

flush_on_tick

Flushes output buffer on every tick when `True`. Default value is `False`.

name

The name of the operator. Default value is `"video_stream_recorder"`.

Attributes

`args`	The list of arguments associated with the component.
`conditions`	Conditions associated with the operator.
`description`	YAML formatted string describing the operator.
`fragment`	The fragment (`holoscan.core.Fragment`) that the operator belongs to.
`id`	The identifier of the component.
`name`	The name of the operator.
`operator_type`	The operator type.
`resources`	Resources associated with the operator.
`spec`	The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

Methods

add_arg (*args, **kwargs)	Overloaded function.
compute (self, arg0, arg1, arg2)	Operator compute method.
initialize (self)	Initialize the operator.
setup (self, spec)	Define the operator specification.
start (self)	Operator start method.
stop (self)	Operator stop method.

OperatorType	

class OperatorType

Bases: pybind11_builtins.pybind11_object

Enum class for operator types used by the executor.

- NATIVE: Native operator.
- GXF: GXF operator.
- VIRTUAL: Virtual operator. (for internal use, not intended for use by application authors)

Members:

NATIVE

GXF

VIRTUAL

Attributes

name	

value	

GXF = *<OperatorType.GXF: 1>*

NATIVE = *<OperatorType.NATIVE: 0>*

VIRTUAL = *<OperatorType.VIRTUAL: 2>*

__init__(*self: holoscan.core._core.Operator.OperatorType*, *value: int*) None

property name

property value

__init__(*self: holoscan.operators.video_stream_recorder._video_stream_recorder.VideoStreamRecorder* *fragment: holoscan.core._core.Fragment*, **args*, *directory: str*, *basename: str*, *flush_on_tick: bool = False*, *name: str = 'recorder'*) None

Operator class to record a video stream to a file.

==Named Inputs==

inputnvidia::gxf::Tensor

A message containing a video frame to serialize to disk. The input tensor can be on either the CPU or GPU. This data location will be recorded as part of the metadata serialized to disk and if the data is later read back in via *VideoStreamReplayerOp*, the tensor output of that operator will be on the same device (CPU or GPU).

Parameters

fragment

The fragment that the operator belongs to.

directory

Directory path for storing files.

basename

User specified file name without extension.

flush_on_tick

Flushes output buffer on every tick when `True`. Default value is `False`.

name

The name of the operator. Default value is `"video_stream_recorder"`.

add_arg(**args*, ***kwargs*)

Overloaded function.

1. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Arg) -> None

Add an argument to the component.

2. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.ArgList) -> None

Add a list of arguments to the component.

3. add_arg(self: holoscan.core._core.Operator, **kwargs) -> None

Add arguments to the component via Python kwargs.

4. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Condition) -> None

5. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Resource) -> None

Add a condition or resource to the Operator.

This can be used to add a condition or resource to an operator after it has already been constructed.

Parameters

arg

The condition or resource to add.

property args

The list of arguments associated with the component.

Returns

arglist

compute(*self: holoscan.core._core.Operator*, *arg0: holoscan.core._core.InputContext*, *arg1: holoscan.core._core.OutputContext*, *arg2: holoscan.core._core.ExecutionContext*) None

Operator compute method. This method defines the primary computation to be executed by the operator.

property conditions

Conditions associated with the operator.

property description

YAML formatted string describing the operator.

property fragment

The fragment (`holoscan.core.Fragment`) that the operator belongs to.

property id

> The identifier of the component.
>
> The identifier is initially set to `-1`, and will become a valid value when the component is initialized.
>
> With the default executor (*holoscan.gxf.GXFExecutor*), the identifier is set to the GXF component ID.
>
> Returns
>
> **id**

initialize(*self: holoscan.operators.video_stream_recorder._video_stream_recorder.VideoStreamRecorder*) → None

Initialize the operator.

This method is called only once when the operator is created for the first time, and uses a light-weight initialization.

property name

The name of the operator.

property operator_type

The operator type.

holoscan.core.Operator.OperatorType enum representing the type of the operator. The two types currently implemented are native and GXF.

property resources

Resources associated with the operator.

setup(*self: holoscan.operators.video_stream_recorder._video_stream_recorder.VideoStreamRecorder* *spec: holoscan.core._core.OperatorSpec*)   None

Define the operator specification.

Parameters

spec

The operator specification.

property spec

The operator spec (holoscan.core.OperatorSpec) associated with the operator.

start(*self: holoscan.core._core.Operator*)   None

Operator start method.

stop(*self: holoscan.core._core.Operator*)   None

Operator stop method.

class holoscan.operators.VideoStreamReplayerOp

Bases: holoscan.core._core.Operator

Operator class to replay a video stream from a file.

==Named Outputs==

outputnvidia::gxf::Tensor

A message containing a video frame deserialized from disk. Depending on the metadata in the file being read, this tensor could be on either CPU or GPU. For the data used in examples distributed with the SDK, the tensor will be an unnamed GPU tensor (name == “”).

Parameters

fragment

The fragment that the operator belongs to.

directory

Directory path for reading files from.

basename

User specified file name without extension.

batch_size

Number of entities to read and publish for one tick. Default value is `1`.

ignore_corrupted_entities

If an entity could not be deserialized, it is ignored by default; otherwise a failure is generated. Default value is `True`.

frame_rate

Frame rate to replay. If zero value is specified, it follows timings in timestamps. Default value is `0.0`.

realtime

Playback video in realtime, based on frame_rate or timestamps. Default value is `True`.

repeat

Repeat video stream in a loop. Default value is `False`.

count

Number of frame counts to playback. If zero value is specified, it is ignored. If the count is less than the number of frames in the video, it would finish early. Default value is `0`.

name

The name of the operator. Default value is `"video_stream_replayer"`.

Attributes

`args`	The list of arguments associated with the component.
`conditions`	Conditions associated with the operator.
`description`	YAML formatted string describing the operator.
`fragment`	The fragment (`holoscan.core.Fragment`) that the operator belongs to.
`id`	The identifier of the component.
`name`	The name of the operator.
`operator_type`	The operator type.
`resources`	Resources associated with the operator.
`spec`	The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

Methods

`add_arg`(*args, **kwargs)	Overloaded function.
`compute`(self, arg0, ar	Operator compute method.

g1, arg2)	
`initialize` (self)	Initialize the operator.
`setup` (self, spec)	Define the operator specification.
`start` (self)	Operator start method.
`stop` (self)	Operator stop method.

OperatorType	

class OperatorType

Bases: `pybind11_builtins.pybind11_object`

Enum class for operator types used by the executor.

- NATIVE: Native operator.
- GXF: GXF operator.
- VIRTUAL: Virtual operator. (for internal use, not intended for use by application authors)

Members:

NATIVE

GXF

VIRTUAL

Attributes

nam e	

value	

GXF = *<OperatorType.GXF: 1>*

NATIVE = *<OperatorType.NATIVE: 0>*

VIRTUAL = *<OperatorType.VIRTUAL: 2>*

__init__(*self: holoscan.core._core.Operator.OperatorType*, *value: int*) None

property name

property value

__init__(*self: holoscan.operators.video_stream_replayer._video_stream_replayer.VideoStreamReplayer*
fragment: holoscan.core._core.Fragment, **args*, *directory: str*, *basename: str*, *batch_size: int = 1*, *ignore_corrupted_entities: bool = True*, *frame_rate: float = 1.0*, *realtime: bool = True*, *repeat: bool = False*, *count: int = 0*, *name: str = 'video_stream_replayer'*) None

Operator class to replay a video stream from a file.

==Named Outputs==

outputnvidia::gxf::Tensor

A message containing a video frame deserialized from disk. Depending on the metadata in the file being read, this tensor could be on either CPU or GPU. For the data used in examples distributed with the SDK, the tensor will be an unnamed GPU tensor (name == “”).

Parameters

fragment

The fragment that the operator belongs to.

directory

Directory path for reading files from.

basename

User specified file name without extension.

batch_size

Number of entities to read and publish for one tick. Default value is `1`.

ignore_corrupted_entities

If an entity could not be deserialized, it is ignored by default; otherwise a failure is generated. Default value is `True`.

frame_rate

Frame rate to replay. If zero value is specified, it follows timings in timestamps. Default value is `0.0`.

realtime

Playback video in realtime, based on frame_rate or timestamps. Default value is `True`.

repeat

Repeat video stream in a loop. Default value is `False`.

count

Number of frame counts to playback. If zero value is specified, it is ignored. If the count is less than the number of frames in the video, it would finish early. Default value is `0`.

name

The name of the operator. Default value is `"video_stream_replayer"`.

add_arg(**args*, ***kwargs*)

Overloaded function.

1. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Arg) -> None

Add an argument to the component.

2. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.ArgList) -> None

Add a list of arguments to the component.

3. add_arg(self: holoscan.core._core.Operator, **kwargs) -> None

Add arguments to the component via Python kwargs.

4. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Condition) -> None

5. add_arg(self: holoscan.core._core.Operator, arg: holoscan.core._core.Resource) -> None

Add a condition or resource to the Operator.

This can be used to add a condition or resource to an operator after it has already been constructed.

Parameters

arg

The condition or resource to add.

property args

The list of arguments associated with the component.

Returns

arglist

compute(*self: holoscan.core._core.Operator*, *arg0: holoscan.core._core.InputContext*, *arg1: holoscan.core._core.OutputContext*, *arg2: holoscan.core._core.ExecutionContext*) None

Operator compute method. This method defines the primary computation to be executed by the operator.

property conditions

Conditions associated with the operator.

property description

YAML formatted string describing the operator.

property fragment

The fragment (`holoscan.core.Fragment`) that the operator belongs to.

property id

The identifier of the component.

The identifier is initially set to `-1`, and will become a valid value when the component is initialized.

With the default executor (*holoscan.gxf.GXFExecutor*), the identifier is set to the GXF component ID.

Returns

id

initialize(*self: holoscan.operators.video_stream_replayer._video_stream_replayer.VideoStreamReplayer(* None

Initialize the operator.

This method is called only once when the operator is created for the first time, and uses a light-weight initialization.

property name

The name of the operator.

property operator_type

The operator type.

holoscan.core.Operator.OperatorType enum representing the type of the operator. The two types currently implemented are native and GXF.

property resources

Resources associated with the operator.

setup(*self: holoscan.operators.video_stream_replayer._video_stream_replayer.VideoStreamReplayerC* *spec: holoscan.core._core.OperatorSpec*) None

Define the operator specification.

Parameters

spec

The operator specification.

property spec

The operator spec (`holoscan.core.OperatorSpec`) associated with the operator.

start(*self: holoscan.core._core.Operator*) None

Operator start method.

stop(*self: holoscan.core._core.Operator*) None

Operator stop method.