Class GXFExecutor

Table of contents

- Defined in File gxf_executor.hpp

Inheritance Relationships

Base Type

- `public holoscan::Executor` (Class Executor)

Class Documentation

class GXFExecutor : public holoscan::Executor

Executor for GXF.

Public Functions

GXFExecutor() = delete

explicit GXFExecutor(holoscan::Fragment *app, bool create_gxf_context = true)

~GXFExecutor() override

virtual void run(OperatorGraph &graph) override

> Initialize the graph and run the graph.
>
> This method calls `compose()` to compose the graph, and runs the graph.
>
> Parameters
>
> **graph** – The reference to the graph.

virtual std::future<void> run_async(OperatorGraph &graph) override

> Initialize the graph and run the graph asynchronously.
>
> This method calls `compose()` to compose the graph, and runs the graph asynchronously. The graph is executed in a separate thread and returns a future object.

Parameters

graph – The reference to the graph.

Returns

The future object.

virtual void interrupt() override

Interrupt the execution.

This method calls GxfGraphInterrupt() to interrupt the execution.

virtual void context(void *context) override

Set the context.

For GXF, GXFExtensionManager(gxf_extension_manager_) is initialized with the context.

Parameters

context – The context.

virtual std::shared_ptr<ExtensionManager> extension_manager() override

Get GXF extension manager.

GXFExtensionManager

Returns

The GXF extension manager.

inline void op_eid(gxf_uid_t eid)

Set the GXF entity ID of the operator initialized by this executor.

If this is 0, a new entity is created for the operator. Otherwise, the operator as a codelet will be added to the existing entity specified by this ID. This is useful when initializing operators inside the existing entity. (e.g., when initializing an operator from `holoscan::gxf::OperatorWrapper` class)

Parameters

eid – The GXF entity ID.

inline void op_cid(gxf_uid_t cid)

Set the GXF component ID of the operator initialized by this executor.

If this is 0, a new component is created for the operator. This is useful when initializing operators using the existing component inside the existing entity. (e.g., when initializing an operator from `holoscan::gxf::OperatorWrapper` class)

Parameters

cid – The GXF component ID.

inline bool own_gxf_context()

Returns whether the GXF context is created by this executor.

Returns

true if the GXF context is created by this executor. Otherwise, false.

inline const std::string &entity_prefix()

Get the entity prefix string.

Returns

The entity prefix string.

inline virtual void context(void *context)

Set the context.

Parameters

context – The context.

inline void *context()

Get the context.

Returns

The context.

Public Static Functions

static void create_input_port(Fragment *fragment, gxf_context_t gxf_context, gxf_uid_t eid, IOSpec *io_spec, bool bind_port = false, Operator *op = nullptr)

Create and setup GXF components for input port.

For a given input port specification, create a GXF Receiver component for the port and create a GXF SchedulingTerm component that is corresponding to the Condition of the port.

If there is no condition specified for the port, a default condition (MessageAvailableCondition) is created. It currently supports ConditionType::kMessageAvailable and ConditionType::kNone condition types.

This function is a static function so that it can be called from other classes without dependency on this class.

Parameters

- **fragment** – The fragment that this operator belongs to.
- **gxf_context** – The GXF context.
- **eid** – The GXF entity ID. (Deprecated: now ignored. The eid is obtained from op instead)
- **io_spec** – The input port specification.
- **bind_port** – If true, bind the port to the existing GXF Receiver component. Otherwise,

- **op** – The operator to which this port is being added. create a new GXF Receiver component.

static void create_output_port(Fragment *fragment, gxf_context_t gxf_context, gxf_uid_t eid, IOSpec *io_spec, bool bind_port = false, Operator *op = nullptr)

Create and setup GXF components for output port.

For a given output port specification, create a GXF Receiver component for the port and create a GXF SchedulingTerm component that is corresponding to the Condition of the port.

If there is no condition specified for the port, a default condition (DownstreamMessageAffordableCondition) is created. It currently supports ConditionType::kDownstreamMessageAffordable and ConditionType::kNone condition types.

This function is a static function so that it can be called from other classes without dependency on on this class.

Parameters

- **fragment** – The fragment that this operator belongs to.
- **gxf_context** – The GXF context.
- **eid** – The GXF entity ID. (Deprecated: now ignored. The eid is obtained from op instead)
- **io_spec** – The output port specification.
- **bind_port** – If true, bind the port to the existing GXF Transmitter component. Otherwise,
- **op** – The operator to which this port is being added. create a new GXF Transmitter component.

Protected Functions

virtual bool initialize_fragment() override

Initialize the fragment_ in this Executor.

This method is called by run() to initialize the fragment and the graph of operators in the fragment before execution.

Returns

true if fragment initialization is successful. Otherwise, false.

virtual bool initialize_operator(Operator *op) override

Initialize the given operator.

This method is called by Operator::initialize() to initialize the operator.

Depending on the type of the operator, this method may be overridden to initialize the operator. For example, the default executor (GXFExecutor) initializes the operator using the GXF API and sets the operator's ID to the ID of the GXF codelet.

Parameters

op – The pointer to the operator.

Returns

true if the operator is initialized successfully. Otherwise, false.

virtual bool initialize_scheduler(Scheduler *sch) override

Initialize the given scheduler.

This method is called by Scheduler::initialize() to initialize the operator.

Depending on the type of the scheduler, this method may be overridden to initialize the scheduler. For example, the default executor (GXFExecutor) initializes the scheduler using the GXF API and sets the operator's ID to the ID of the GXF scheduler.

Parameters

sch – The pointer to the scheduler.

Returns

true if the scheduler is initialized successfully. Otherwise, false.

virtual bool initialize_network_context(NetworkContext *network_context) override

Initialize the given network context.

This method is called by NetworkContext::initialize() to initialize the operator.

Depending on the type of the network context, this method may be overridden to initialize the network context. For example, the default executor (GXFExecutor) initializes the network context using the GXF API and sets the operator's ID to the ID of the GXF network context.

Parameters

network_context – The pointer to the network context.

Returns

true if the network context is initialized successfully. Otherwise, false.

virtual bool add_receivers(const std::shared_ptr<Operator> &op, const std::string &receivers_name, std::vector<std::string> &new_input_labels, std::vector<holoscan::IOSpec*> &iospec_vector) override

Add the receivers as input ports of the given operator.

This method is to be called by the Fragment::add_flow() method to support for the case where the destination input port label points to the parameter name of the downstream operator, and the parameter type is 'std::vector<holoscan::IOSpec*>'. This finds a parameter with with 'std::vector<holoscan::IOSpec*>' type and create a new input port with a specific label ('<parameter name>:<index>'. e.g, 'receivers:0').

Parameters

- **op** – The reference to the shared pointer of the operator.
- **receivers_name** – The name of the receivers whose parameter type is 'std::vector<holoscan::IOSpec*>'.
- **new_input_labels** – The reference to the vector of input port labels to which the input port labels are added. In the case of multiple receivers, the input

port label is updated to '<parameter name>:<index>' (e.g. 'receivers' => 'receivers:<index>').

- **iospec_vector** – The reference to the vector of IOSpec pointers.

Returns

true if the receivers are added successfully. Otherwise, false.

bool initialize_gxf_graph(OperatorGraph &graph)

void activate_gxf_graph()

void run_gxf_graph()

bool connection_items(std::vector<std::shared_ptr<holoscan::ConnectionItem>> &connection_items)

void add_operator_to_entity_group(gxf_context_t context, gxf_uid_t entity_group_gid, std::shared_ptr<Operator> op)

void register_extensions()

Protected Attributes

bool own_gxf_context_ = false

Whether this executor owns the GXF context.

gxf_uid_t op_eid_ = 0

The GXF entity ID of the operator. Create new entity for initializing a new operator if this is 0.

gxf_uid_t op_cid_ = 0

The GXF component ID of the operator. Create new component for initializing a new operator if this is 0.

std::shared_ptr<GXFExtensionManager> gxf_extension_manager_

The GXF extension manager.

nvidia::gxf::Extension *gxf_holoscan_extension_ = nullptr

The GXF holoscan extension.

bool is_extensions_loaded_ = false

The flag to indicate whether the extensions are loaded.

bool is_gxf_graph_initialized_ = false

The flag to indicate whether the GXF graph is initialized.

bool is_gxf_graph_activated_ = false

The flag to indicate whether the GXF graph is activated.

std::string entity_prefix_

The entity prefix for the fragment.

std::vector<std::shared_ptr<holoscan::ConnectionItem>> connection_items_

The connection items for virtual operators.

std::list<std::shared_ptr<nvidia::gxf::GraphEntity>> implicit_broadcast_entities_

The list of implicit broadcast entities to be added to the network entity group.

std::shared_ptr<nvidia::gxf::GraphEntity> util_entity_

std::shared_ptr<nvidia::gxf::GraphEntity> gpu_device_entity_

std::shared_ptr<nvidia::gxf::GraphEntity> scheduler_entity_

std::shared_ptr<nvidia::gxf::GraphEntity> network_context_entity_

std::shared_ptr<nvidia::gxf::GraphEntity> connections_entity_

Friends

friend class holoscan::AppDriver

friend class holoscan::AppWorker