# Program Listing for File gxf_executor.hpp

Return to documentation for file ( `include/holoscan/core/executors/gxf/gxf_executor.hpp` )

```
/* * SPDX-FileCopyrightText: Copyright (c) 2022-2024 NVIDIA CORPORATION &
AFFILIATES. All rights reserved. * SPDX-License-Identifier: Apache-2.0 * * Licensed
under the Apache License, Version 2.0 (the "License"); * you may not use this file
except in compliance with the License. * You may obtain a copy of the License at * *
http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law
or agreed to in writing, software * distributed under the License is distributed on an
"AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
express or implied. * See the License for the specific language governing
permissions and * limitations under the License. */ #ifndef
HOLOSCAN_CORE_EXECUTORS_GXF_GXF_EXECUTOR_HPP #define
HOLOSCAN_CORE_EXECUTORS_GXF_GXF_EXECUTOR_HPP #include <gxf/core/gxf.h>
#include <cstdint> #include <functional> #include <future> #include <list> #include
<memory> #include <set> #include <string> #include <unordered_map> #include
<tuple> #include <utility> #include <vector> #include "../../app_driver.hpp" #include
"../../executor.hpp" #include "../../graph.hpp" #include
"../../gxf/gxf_extension_manager.hpp" #include "gxf/app/graph_entity.hpp"
namespace holoscan { // Forward declarations class Arg; class Condition; class
Resource; } // namespace holoscan namespace holoscan::gxf { class GXFExecutor :
public holoscan::Executor { public: GXFExecutor() = delete; explicit
GXFExecutor(holoscan::Fragment* app, bool create_gxf_context = true);
~GXFExecutor() override; void run(OperatorGraph& graph) override;
std::future<void> run_async(OperatorGraph& graph) override; void interrupt()
override; void context(void* context) override; // Inherit Executor::context(). using
Executor::context; std::shared_ptr<ExtensionManager> extension_manager()
override; 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); 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); void op_eid(gxf_uid_t eid) { op_eid_ = eid; } void op_cid(gxf_uid_t cid) {
op_cid_ = cid; } bool own_gxf_context() { return own_gxf_context_; } const
std::string& entity_prefix() { return entity_prefix_; } protected: bool
initialize_fragment() override; bool initialize_operator(Operator* op) override; bool
```

```
initialize_scheduler(Scheduler* sch) override; bool
initialize_network_context(NetworkContext* network_context) override; 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; friend class
holoscan::AppDriver; friend class holoscan::AppWorker; 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(); bool own_gxf_context_ = false; gxf_uid_t op_eid_ = 0; gxf_uid_t
op_cid_ = 0; std::shared_ptr<GXFExtensionManager> gxf_extension_manager_;
nvidia::gxf::Extension* gxf_holoscan_extension_ = nullptr; bool
is_extensions_loaded_ = false; bool is_gxf_graph_initialized_ = false; bool
is_gxf_graph_activated_ = false; std::string entity_prefix_;
std::vector<std::shared_ptr<holoscan::ConnectionItem>> connection_items_;
std::list<std::shared_ptr<nvidia::gxf::GraphEntity>> implicit_broadcast_entities_;
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_; private: // Map of
connections indexed by source port uid and stores a pair of the target operator name //
and target port name using TargetPort =
std::pair<holoscan::OperatorGraph::NodeType, std::string>; using TargetsInfo =
std::tuple<std::string, IOSpec::ConnectorType, std::set<TargetPort>>; using
TargetConnectionsMapType = std::unordered_map<gxf_uid_t, TargetsInfo>; using
BroadcastEntityMapType = std::unordered_map<
holoscan::OperatorGraph::NodeType, std::unordered_map<std::string,
std::shared_ptr<nvidia::gxf::GraphEntity>>>; void initialize_gxf_resources(
std::unordered_map<std::string, std::shared_ptr<Resource>>& resources, gxf_uid_t
eid, std::shared_ptr<nvidia::gxf::GraphEntity> graph_entity); gxf_result_t
add_connection(gxf_uid_t source_cid, gxf_uid_t target_cid); void
create_broadcast_components(holoscan::OperatorGraph::NodeType op,
BroadcastEntityMapType& broadcast_entities, const TargetConnectionsMapType&
connections); void connect_broadcast_to_previous_op(const
BroadcastEntityMapType& broadcast_entities, holoscan::OperatorGraph::NodeType
op, holoscan::OperatorGraph::NodeType prev_op,
holoscan::OperatorGraph::EdgeDataType port_map_val); bool is_holoscan() const;
bool add_condition_to_graph_entity(std::shared_ptr<Condition> condition,
std::shared_ptr<nvidia::gxf::GraphEntity> graph_entity); bool
add_resource_to_graph_entity(std::shared_ptr<Resource> resource,
std::shared_ptr<nvidia::gxf::GraphEntity> graph_entity); /* @brief Add an IOspec
connector resource and any conditions to the graph entity. * * Helper function for
add_component_arg_to_graph_entity. * * @param io_spec Pointer to the IOSpec
object to update. * @param graph_entity The graph entity this IOSpec will be
associated with. * @return true if the IOSpec's components were all successfully
added to the graph entity. */ bool add_iospec_to_graph_entity(IOSpec* io_spec,
std::shared_ptr<nvidia::gxf::GraphEntity> graph_entity); /* @brief Add any GXF
resources and conditions present in the arguments to the provided graph * entity. *
* Handles Component, Resource and IOSpec arguments and vectors of each of
these. * * @param io_spec Pointer to the IOSpec object to update. * @param
graph_entity The graph entity this IOSpec will be associated with. * @return true if
the IOSpec's components were all successfully added to the graph entity. */ void
add_component_args_to_graph_entity(std::vector<Arg>& args,
std::shared_ptr<nvidia::gxf::GraphEntity> graph_entity); }; } // namespace
holoscan::gxf #endif/* HOLOSCAN_CORE_EXECUTORS_GXF_GXF_EXECUTOR_HPP */
```