Program Listing for File executor.hpp

Return to documentation for file (`include/holoscan/core/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_EXECUTOR_HPP #define
HOLOSCAN_CORE_EXECUTORS_EXECUTOR_HPP #include <cstdint> #include
<functional> #include <future> #include <memory> #include <set> #include
<string> #include <unordered_map> #include <vector> #include "./common.hpp"
#include "./extension_manager.hpp" #include "./graph.hpp" #include
"./operator.hpp" namespace holoscan { class Executor { public: Executor() = delete;
explicit Executor(Fragment* fragment) : fragment_(fragment) {} virtual ~Executor() =
default; virtual void run(OperatorGraph& graph) { (void)graph; } virtual
std::future<void> run_async(OperatorGraph& graph) { (void)graph; return {}; }
virtual void interrupt() {} void fragment(Fragment* fragment) { fragment_ =
fragment; } Fragment* fragment() { return fragment_; } virtual void context(void*
context) { context_ = context; } void* context() { return context_; } // add uint64_t
context getters/setters for Python API void context_uint64(uint64_t context) { context_
= reinterpret_cast<void*>(context); } uint64_t context_uint64() { return
reinterpret_cast<uint64_t>(context_); } virtual std::shared_ptr<ExtensionManager>
extension_manager() { return extension_manager_; } void exception(const
std::exception_ptr& e) { exception_ = e; } const std::exception_ptr& exception() {
return exception_; } protected: friend class Fragment; // make Fragment a friend class
to access protected members of // Executor (add_receivers()). friend class Operator; //
make Operator a friend class to access protected members of // Executor
(initialize_operator()). friend class Scheduler; // make Scheduler a friend class to access
protected members of // Executor (initialize_scheduler()). friend class NetworkContext;
// make NetworkContext a friend class to access protected members // of Executor
(initialize_network_context()). virtual bool initialize_fragment() { return false; } virtual
bool initialize_operator(Operator* op) { (void)op; return false; } virtual bool
initialize_scheduler(Scheduler* sch) { (void)sch; return false; } virtual bool
initialize_network_context(NetworkContext* network_context) {
(void)network_context; return 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) { (void)op; (void)receivers_name; (void)new_input_labels;
(void)iospec_vector; return false; } Fragment* fragment_ = nullptr; void* context_ =
nullptr; std::shared_ptr<ExtensionManager> extension_manager_;
std::exception_ptr exception_; }; } // namespace holoscan #endif/*
HOLOSCAN_CORE_EXECUTORS_EXECUTOR_HPP */
```