# Program Listing for File application.hpp

Return to documentation for file ( include/holoscan/core/application.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_APPLICATION_HPP #define HOLOSCAN_CORE_APPLICATION_HPP #include <iostream> *// for std::cout* #include <memory> *// for std::shared_ptr* #include <set> *// for std::set* #include <string> *// for std::string* #include <type_traits> *// for std::enable_if_t, std::is_constructible* #include <utility> *// for std::pair* #include <vector> *// for std::vector* #include "./fragment.hpp" #include "./app_driver.hpp" #include "./app_worker.hpp" #include "./cli_parser.hpp" namespace holoscan { template <typename AppT, typename... ArgsT> std::shared_ptr<AppT> make_application(ArgsT&&... args) { return std::make_shared<AppT> (std::forward<ArgsT>(args)...); } class Application : public Fragment { public: explicit Application(const std::vector<std::string>& argv = {}); ~Application() override = default; template <typename FragmentT = Fragment, typename StringT, typename... ArgsT, typename = std::enable_if_t<std::is_constructible_v<std::string, StringT>>> std::shared_ptr<Fragment> make_fragment(StringT name, ArgsT&&... args) { auto fragment = std::make_shared<FragmentT>(std::forward<ArgsT>(args)...); fragment->name(name); fragment->application(this); *// Set the fragment config to the application config.* fragment->config(config_); return fragment; } template <typename FragmentT, typename... ArgsT> std::shared_ptr<FragmentT> make_fragment(ArgsT&&... args) { auto fragment = std::make_shared<FragmentT> (std::forward<ArgsT>(args)...); fragment->application(this); *// Set the fragment config to the application config.* fragment->config(config_); return fragment; } std::string& description(); Application& description(const std::string& desc) &; Application&& description(const std::string& desc) &&; std::string& version(); Application& version(const std::string& version) &; Application&& version(const std::string& version) &&; std::vector<std::string>& argv(); CLIOptions& options();

```
FragmentGraph& fragment_graph(); virtual void add_fragment(const std::shared_ptr<Fragment>& frag); // Inherit Fragment's add_flow methods (for Operator) in addition to the overloads below using Fragment::add_flow; virtual void add_flow(const std::shared_ptr<Fragment>& upstream_frag, const std::shared_ptr<Fragment>& downstream_frag, std::set<std::pair<std::string, std::string>> port_pairs); void compose_graph() override; void run() override; std::future<void> run_async() override; protected: friend class AppDriver; friend class AppWorker; AppDriver& driver(); AppWorker& worker(); void process_arguments(); static expected<SchedulerType, ErrorCode> get_distributed_app_scheduler_env(); static expected<bool, ErrorCode> get_stop_on_deadlock_env(); static expected<int64_t, ErrorCode> get_stop_on_deadlock_timeout_env(); static expected<int64_t, ErrorCode> get_max_duration_ms_env(); static expected<double, ErrorCode> get_check_recession_period_ms_env(); static void set_scheduler_for_fragments(std::vector<FragmentNodeType>& target_fragments); std::string app_description_{}; std::string app_version_{"0.0.0"}; CLIParser cli_parser_; std::vector<std::string> argv_; std::unique_ptr<FragmentGraph> fragment_graph_; std::shared_ptr<AppDriver> app_driver_; std::shared_ptr<AppWorker> app_worker_; private: void set_ucx_env(); }; } // namespace holoscan #endif/* HOLOSCAN_CORE_APPLICATION_HPP */
```