Program Listing for File scheduler.hpp

Return to documentation for file (include/holoscan/core/scheduler.hpp)

/* * SPDX-FileCopyrightText: Copyright (c) 2023-2024 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_SCHEDULER_HPP #define HOLOSCAN_CORE_SCHEDULER_HPP #include <stdio.h> #include <iostream> #include <memory> #include <string> #include <type_traits> #include <unordered_map> #include <utility> #include <vector> #include "./parameter.hpp" #include "./type_traits.hpp" #include "./arg.hpp" #include "./forward_def.hpp" #include "./component.hpp" #include "./resource.hpp" #define HOLOSCAN_SCHEDULER_FORWARD_TEMPLATE() \ template <typename ArgT, \ typename... ArgsT, \ typename = \ std::enable_if_t<!std::is_base_of_v<::holoscan::Scheduler, std::decay_t<ArgT>> && \ (std::is_same_v<::holoscan::Arg, std::decay_t<ArgT>> || \ std::is_same_v<::holoscan::ArgList, std::decay_t<ArgT>>)>> #define HOLOSCAN_SCHEDULER_FORWARD_ARGS(class_name) \ HOLOSCAN_SCHEDULER_FORWARD_TEMPLATE() \ class_name(ArgT&& arg, ArgsT&&... args) \ : Scheduler(std::forward<ArgT>(arg), std::forward<ArgsT>(args)...) {} #define HOLOSCAN_SCHEDULER_FORWARD_ARGS_SUPER(class_name, super_class_name) \ HOLOSCAN_SCHEDULER_FORWARD_TEMPLATE() \ class_name(ArgT&& arg, ArgsT&&... args) \ : super_class_name(std::forward<ArgT>(arg), std::forward<ArgsT>(args)...) {} namespace holoscan { enum class SchedulerType { kDefault, kGreedy, kMultiThread, kEventBased }; class Scheduler : public Component { public: Scheduler() = default; Scheduler(Scheduler&&) = default; HOLOSCAN_SCHEDULER_FORWARD_TEMPLATE() explicit Scheduler(ArgT&& arg, ArgsT&&... args) { add_arg(std::forward<ArgT>(arg)); (add_arg(std::forward<ArgsT>(args)), ...); } ~Scheduler() override = default; using Component::id; Scheduler& id(int64_t id) { id_ = id; return *this; } using holoscan::Component::name; Scheduler& name(const std::string& name) & { name_ = name; return *this; } Scheduler&&

```
name(const std::string& name) && { name_ = name; return std::move(*this); } using holoscan::Component::fragment; Scheduler& fragment(Fragment* fragment) { fragment_ = fragment; return *this; } Scheduler& spec(const std::shared_ptr<ComponentSpec>& spec) { spec_ = spec; return *this; } ComponentSpec* spec() { return spec_.get(); } std::shared_ptr<ComponentSpec> spec_shared() { return spec_; } using Component::add_arg; void add_arg(const std::shared_ptr<Resource>& arg) { if (resources_.find(arg->name()) != resources_.end()) { HOLOSCAN_LOG_ERROR( "Resource '{}' already exists in the scheduler. Please specify a unique " "name when creating a Resource instance.", arg->name()); } else { resources_[arg->name()] = arg; } } void add_arg(std::shared_ptr<Resource>&& arg) { if (resources_.find(arg->name()) != resources_.end()) { HOLOSCAN_LOG_ERROR( "Resource '{}' already exists in the scheduler. Please specify a unique " "name when creating a Resource instance.", arg->name()); } else { resources_[arg->name()] = std::move(arg); } } std::unordered_map<std::string, std::shared_ptr<Resource>>& resources() { return resources_; } virtual void setup(ComponentSpec& spec) { (void)spec; } void initialize() override; YAML::Node to_yaml_node() const override; protected: void reset_graph_entities() override; std::unordered_map<std::string, std::shared_ptr<Resource>> resources_; }; } // namespace holoscan #endif/* HOLOSCAN_CORE_SCHEDULER_HPP */
```