Program Listing for File fragment_scheduler.hpp

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

```
/* * SPDX-FileCopyrightText: Copyright (c) 2023 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_FRAGMENT_SCHEDULER_HPP #define HOLOSCAN_CORE_FRAGMENT_SCHEDULER_HPP #include <memory> #include <string> #include <unordered_map> #include <unordered_set> #include <utility> #include <vector> #include "holoscan/core/expected.hpp" namespace holoscan { struct SystemResourceRequirement { std::string fragment_name; float cpu = -1.0f; float cpu_limit = -1.0f; float gpu = -1.0f; float gpu_limit = -1.0f; uint64_t memory = 0; uint64_t memory_limit = 0; uint64_t shared_memory = 0; uint64_t shared_memory_limit = 0; uint64_t gpu_memory = 0; uint64_t gpu_memory_limit = 0; }; struct AvailableSystemResource { std::string app_worker_id; std::unordered_set<std::string> target_fragments; int32_t cpu = 0; int32_t gpu = 0; uint64_t memory = 0; uint64_t shared_memory = 0; uint64_t gpu_memory = 0; bool has_enough_resources(const SystemResourceRequirement& resource_requirement) const; }; class FragmentAllocationStrategy { public: virtual ~FragmentAllocationStrategy() = default; void add_resource_requirement(const SystemResourceRequirement& resource_requirement); void add_resource_requirement(SystemResourceRequirement&& resource_requirement); void add_available_resource(const AvailableSystemResource& available_resource); void add_available_resource(AvailableSystemResource&& available_resource); virtual void on_add_resource_requirement( const SystemResourceRequirement& resource_requirement) = 0; virtual void on_add_available_resource(const AvailableSystemResource& available_resource) = 0; virtual holoscan::expected<std::unordered_map<std::string, std::string>, std::string> schedule() = 0; protected: std::unordered_map<std::string,
```

```
SystemResourceRequirement> resource_requirements_;
std::unordered_map<std::string, AvailableSystemResource> available_resources_; };
class FragmentScheduler { public: explicit FragmentScheduler(
std::unique_ptr<FragmentAllocationStrategy>&& allocation_strategy = {}); virtual
~FragmentScheduler(); void add_resource_requirement(const
SystemResourceRequirement& resource_requirement); void
add_resource_requirement(SystemResourceRequirement&&
resource_requirement); void add_available_resource(const
AvailableSystemResource& available_resource); void
add_available_resource(AvailableSystemResource&& available_resource);
holoscan::expected<std::unordered_map<std::string, std::string>, std::string>
schedule(); private: std::unique_ptr<FragmentAllocationStrategy> strategy_; }; } //
namespace holoscan #endif/* HOLOSCAN_CORE_FRAGMENT_SCHEDULER_HPP */
```