Program Listing for File operator_spec.hpp

Return to documentation for file (`include/holoscan/core/operator_spec.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_OPERATOR_SPEC_HPP #define
HOLOSCAN_CORE_OPERATOR_SPEC_HPP #include <iostream> #include <memory>
#include <string> #include <typeinfo> #include <unordered_map> #include <utility>
#include <vector> #include "./common.hpp" #include "./component_spec.hpp"
#include "./io_spec.hpp" namespace holoscan { class OperatorSpec : public
ComponentSpec { public: explicit OperatorSpec(Fragment* fragment = nullptr) :
ComponentSpec(fragment) {} std::unordered_map<std::string,
std::shared_ptr<IOSpec>>& inputs() { return inputs_; } template <typename DataT>
IOSpec& input() { return input<DataT>("__iospec_input"); } template <typename
DataT> IOSpec& input(std::string name) { auto spec = std::make_shared<IOSpec>
(this, name, IOSpec::IOType::kInput, &typeid(DataT)); auto [iter, is_exist] =
inputs_.insert_or_assign(name, std::move(spec)); if (!is_exist) {
HOLOSCAN_LOG_ERROR("Input port '{}' already exists", name); } return *(iter-
>second.get()); } std::unordered_map<std::string, std::shared_ptr<IOSpec>>&
outputs() { return outputs_; } template <typename DataT> IOSpec& output() { return
output<DataT>("__iospec_output"); } template <typename DataT> IOSpec&
output(std::string name) { auto spec = std::make_shared<IOSpec>(this, name,
IOSpec::IOType::kOutput, &typeid(DataT)); auto [iter, is_exist] =
outputs_.insert_or_assign(name, std::move(spec)); if (!is_exist) {
HOLOSCAN_LOG_ERROR("Output port '{}' already exists", name); } return *(iter-
>second.get()); } using ComponentSpec::param; void
param(Parameter<holoscan::IOSpec*>& parameter, const char* key, const char*
headline, const char* description, ParameterFlag flag = ParameterFlag::kNone) {
parameter.key_ = key; parameter.headline_ = headline; parameter.description_ =
description; parameter.flag_ = flag; auto [_, is_exist] = params_.try_emplace(key,
parameter); if (!is_exist) { HOLOSCAN_LOG_ERROR("Parameter '{}' already exists",
key); } } void param(Parameter<holoscan::IOSpec*>& parameter, const char* key,
const char* headline, const char* description, std::initializer_list<void*> init_list) {
(void)init_list; parameter.key_ = key; parameter.headline_ = headline;
parameter.description_ = description; // Set default value to nullptr
parameter.default_value_ = static_cast<holoscan::IOSpec*>(nullptr); auto [_, is_exist]
= params_.try_emplace(key, parameter); if (!is_exist) {
HOLOSCAN_LOG_ERROR("Parameter '{}' already exists", key); } } void
param(Parameter<holoscan::IOSpec*>& parameter, const char* key, const char*
headline, const char* description, holoscan::IOSpec* default_value, ParameterFlag
flag = ParameterFlag::kNone) { parameter.default_value_ = default_value;
param(parameter, key, headline, description, flag); } void
param(Parameter<std::vector<holoscan::IOSpec*>>& parameter, const char* key,
const char* headline, const char* description, ParameterFlag flag =
ParameterFlag::kNone) { parameter.key_ = key; parameter.headline_ = headline;
parameter.description_ = description; parameter.flag_ = flag; auto [_, is_exist] =
params_.try_emplace(key, parameter); if (!is_exist) {
HOLOSCAN_LOG_ERROR("Parameter '{}' already exists", key); } } void
param(Parameter<std::vector<holoscan::IOSpec*>>& parameter, const char* key,
const char* headline, const char* description,
std::initializer_list<holoscan::IOSpec*> init_list) { parameter.key_ = key;
parameter.headline_ = headline; parameter.description_ = description;
parameter.default_value_ = init_list; // create a vector from initializer list auto [_,
is_exist] = params_.try_emplace(key, parameter); if (!is_exist) {
HOLOSCAN_LOG_ERROR("Parameter '{}' already exists", key); } } void
param(Parameter<std::vector<holoscan::IOSpec*>>& parameter, const char* key,
const char* headline, const char* description, std::vector<holoscan::IOSpec*>
default_value, ParameterFlag flag = ParameterFlag::kNone) {
parameter.default_value_ = default_value; param(parameter, key, headline,
description, flag); } YAML::Node to_yaml_node() const override; protected:
std::unordered_map<std::string, std::shared_ptr<IOSpec>> inputs_;
std::unordered_map<std::string, std::shared_ptr<IOSpec>> outputs_; }; } //
namespace holoscan #endif/* HOLOSCAN_CORE_OPERATOR_SPEC_HPP */
```