Program Listing for File virtual_operator.hpp

Return to documentation for file (include/holoscan/core/services/common/virtual_operator.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_SERVICES_COMMON_VIRTUAL_OPERATOR_HPP #define HOLOSCAN_CORE_SERVICES_COMMON_VIRTUAL_OPERATOR_HPP #include <string> #include <utility> #include "../../operator.hpp" namespace holoscan::ops { class VirtualOperator : public holoscan::Operator { public: // *HOLOSCAN_OPERATOR_FORWARD_ARGS(VirtualOperator)* template <typename StringT, typename = std::enable_if_t<std::is_constructible_v<std::string, StringT>>> VirtualOperator(StringT port_name, IOSpec::ConnectorType connector_type, ArgList arg_list) : port_name_(port_name), connector_type_(connector_type), arg_list_(arg_list) { operator_type_ = OperatorType::kVirtual; } VirtualOperator() : Operator() { operator_type_ = OperatorType::kVirtual; } void initialize() override; const std::string& port_name() const { return port_name_; } void port_name(const std::string& port_name) { port_name_ = port_name; } IOSpec::ConnectorType connector_type() const { return connector_type_; } const ArgList& arg_list() const { return arg_list_; } IOSpec* input_spec(); IOSpec* output_spec(); IOSpec::IOType io_type() const { return io_type_; } protected: std::string port_name_; IOSpec::ConnectorType connector_type_; ArgList arg_list_; IOSpec* input_spec_ = nullptr; IOSpec* output_spec_ = nullptr; IOSpec::IOType io_type_ = IOSpec::IOType::kInput; }; class VirtualTransmitterOp : public VirtualOperator { public: template <typename StringT, typename ArgListT, typename = std::enable_if_t<std::is_constructible_v<std::string, StringT> && std::is_same_v<ArgList, std::decay_t<ArgListT>>>> explicit VirtualTransmitterOp(StringT&& output_port_name, IOSpec::ConnectorType connector_type, ArgListT&& arg_list) : VirtualOperator(std::forward<StringT>

```
(output_port_name), connector_type, std::forward<ArgListT>(arg_list)) { io_type_ = IOSpec::IOType::kOutput; } void setup(OperatorSpec& spec) override; }; class VirtualReceiverOp : public VirtualOperator { public: template <typename StringT, typename ArgListT, typename = std::enable_if_t<std::is_constructible_v<std::string, StringT> && std::is_same_v<ArgList, std::decay_t<ArgListT>>>> explicit VirtualReceiverOp(StringT&& input_port, IOSpec::ConnectorType connector_type, ArgListT&& arg_list) : VirtualOperator(std::forward<StringT>(input_port), connector_type, std::forward<ArgListT>(arg_list)) { io_type_ = IOSpec::IOType::kInput; } void setup(OperatorSpec& spec) override; }; } // namespace holoscan::ops #endif/* HOLOSCAN_CORE_SERVICES_COMMON_VIRTUAL_OPERATOR_HPP */
```