Program Listing for File condition.hpp

Return to documentation for file (include/holoscan/core/condition.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_CONDITION_HPP #define HOLOSCAN_CORE_CONDITION_HPP #include <gxf/core/gxf.h> #include <any> #include <iostream> #include <memory> #include <optional> #include <string> #include <typeindex> #include <typeinfo> #include <unordered_map> #include <utility> #include "./common.hpp" #include "./component.hpp" #include "./gxf/gxf_component.hpp" #include "./gxf/gxf_utils.hpp" #define HOLOSCAN_CONDITION_FORWARD_TEMPLATE() \ template <typename ArgT, \ typename... ArgsT, \ typename = \ std::enable_if_t<!std::is_base_of_v<::holoscan::Condition, 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_CONDITION_FORWARD_ARGS(class_name) \ HOLOSCAN_CONDITION_FORWARD_TEMPLATE() \ class_name(ArgT&& arg, ArgsT&&... args) \ : Condition(std::forward<ArgT>(arg), std::forward<ArgsT>(args)...) {} #define HOLOSCAN_CONDITION_FORWARD_ARGS_SUPER(class_name, super_class_name) \ HOLOSCAN_CONDITION_FORWARD_TEMPLATE() \ class_name(ArgT&& arg, ArgsT&&... args) \ : super_class_name(std::forward<ArgT>(arg), std::forward<ArgsT>(args)...) {} namespace holoscan { *// Forward declarations* class Operator; *// Note: Update `IOSpec::to_yaml_node()` if you add new condition types* enum class ConditionType { kNone, kMessageAvailable, kDownstreamMessageAffordable, kCount, kBoolean, kPeriodic, kAsynchronous, }; class Condition : public Component { public: Condition() = default; Condition(Condition&&) = default; HOLOSCAN_CONDITION_FORWARD_TEMPLATE() explicit Condition(ArgT&& arg, ArgsT&&... args) { add_arg(std::forward<ArgT>(arg)); (add_arg(std::forward<ArgsT>(args)), ...); } ~Condition() override = default; using

```
Component::name; Condition& name(const std::string& name) & { name_ = name; return *this; } Condition&& name(const std::string& name) && { name_ = name; return std::move(*this); } using Component::fragment; Condition& fragment(Fragment* fragment) { fragment_ = fragment; return *this; } Condition& 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; virtual void setup(ComponentSpec& spec) { (void)spec; } YAML::Node to_yaml_node() const override; protected: // Add friend classes that can call reset_graph_entites friend class holoscan::Operator; using Component::reset_graph_entities; bool is_initialized_ = false; }; } // namespace holoscan #endif/* HOLOSCAN_CORE_CONDITION_HPP */
```