Program Listing for File component.hpp

Return to documentation for file (include/holoscan/core/component.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_COMPONENT_HPP #define HOLOSCAN_CORE_COMPONENT_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" #define HOLOSCAN_COMPONENT_FORWARD_TEMPLATE() \ template <typename ArgT, \ typename... ArgsT, \ typename = std::enable_if_t< \ !std::is_base_of_v<::holoscan::ComponentBase, 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_COMPONENT_FORWARD_ARGS(class_name) \ HOLOSCAN_COMPONENT_FORWARD_TEMPLATE() \ class_name(ArgT&& arg, ArgsT&&... args) \ : Component(std::forward<ArgT>(arg), std::forward<ArgsT>(args)...) #define HOLOSCAN_COMPONENT_FORWARD_ARGS_SUPER(class_name, super_class_name) \ HOLOSCAN_COMPONENT_FORWARD_TEMPLATE() \ class_name(ArgT&& arg, ArgsT&&... args) \ : super_class_name(std::forward<ArgT>(arg), std::forward<ArgsT>(args)...) namespace holoscan { namespace gxf { class GXFExecutor; } *// namespace gxf* class ComponentBase { public: ComponentBase() = default; HOLOSCAN_COMPONENT_FORWARD_TEMPLATE() explicit ComponentBase(ArgT&& arg, ArgsT&&... args) { add_arg(std::forward<ArgT>(arg)); (add_arg(std::forward<ArgsT>(args)), ...); } virtual ~ComponentBase() = default; int64_t id() const { return id_; } const std::string& name() const { return name_; } Fragment* fragment() { return fragment_; } void add_arg(const Arg& arg) { args_.emplace_back(arg); } void add_arg(Arg&& arg) {

```
args_.emplace_back(std::move(arg)); } void add_arg(const ArgList& arg) {
args_.reserve(args_.size() + arg.size()); args_.insert(args_.end(), arg.begin(), arg.end());
} void add_arg(ArgList&& arg) { args_.reserve(args_.size() + arg.size()); args_.insert(
args_.end(), std::make_move_iterator(arg.begin()),
std::make_move_iterator(arg.end())); arg.clear(); } std::vector<Arg>& args() { return
args_; } virtual void initialize() {} virtual YAML::Node to_yaml_node() const; std::string
description() const; protected: friend class holoscan::Executor; // Make GXFExecutor a
friend class so it can call protected initialization methods friend class
holoscan::gxf::GXFExecutor; // Make Fragment a friend class so it can call
reset_graph_entities friend class holoscan::Fragment; void
update_params_from_args(std::unordered_map<std::string, ParameterWrapper>&
params); virtual void reset_graph_entities(); int64_t id_ = -1; std::string name_ = "";
Fragment* fragment_ = nullptr; std::vector<Arg> args_; }; class Component : public
ComponentBase { protected: // Make GXFExecutor a friend class so it can call
protected initialization methods friend class holoscan::gxf::GXFExecutor; using
ComponentBase::update_params_from_args; void update_params_from_args();
virtual void set_parameters() {} std::shared_ptr<ComponentSpec> spec_; }; } //
namespace holoscan #endif/* HOLOSCAN_CORE_COMPONENT_HPP */
```