Program Listing for File component_spec-inl.hpp

Return to documentation for file (include/holoscan/core/component_spec-inl.hpp)

/* * SPDX-FileCopyrightText: Copyright (c) 2022-2023 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_COMPONENT_SPEC_INL_HPP #define HOLOSCAN_CORE_COMPONENT_SPEC_INL_HPP #include <memory> #include <utility> #include "./component_spec.hpp" #include "./argument_setter.hpp" #include "./executors/gxf/gxf_parameter_adaptor.hpp" namespace holoscan { template <typename typeT> void ComponentSpec::param(Parameter<typeT>& parameter, const char* key, const char* headline, const char* description, ParameterFlag flag) { parameter.key_ = key; parameter.headline_ = headline; parameter.description_ = description; parameter.flag_ = flag; if constexpr (is_one_of_v<typename holoscan::type_info<typeT>::element_type, std::shared_ptr<Resource>, std::shared_ptr<Condition>>) { ArgumentSetter::ensure_type<typeT>(); ::holoscan::gxf::GXFParameterAdaptor::ensure_type<typeT>(); } params_.try_emplace(key, ParameterWrapper{parameter}); } template <typename typeT> void ComponentSpec::param(Parameter<typeT>& parameter, const char* key, const char* headline, const char* description, std::initializer_list<void*>) { parameter.key_ = key; parameter.headline_ = headline; parameter.description_ = description; parameter.default_value_ = typeT{}; if constexpr (is_one_of_v<typename holoscan::type_info<typeT>::element_type, std::shared_ptr<Resource>, std::shared_ptr<Condition>>) { ArgumentSetter::ensure_type<typeT>(); ::holoscan::gxf::GXFParameterAdaptor::ensure_type<typeT>(); } params_.try_emplace(key, ParameterWrapper{parameter}); } template <typename typeT> void ComponentSpec::param(Parameter<typeT>& parameter, const char* key, const char* headline, const char* description, const typeT& default_value, ParameterFlag flag) { parameter.key_ = key; parameter.headline_ = headline;

```
parameter.description_ = description; parameter.flag_ = flag;
parameter.default_value_ = default_value; if constexpr (is_one_of_v<typename
holoscan::type_info<typeT>::element_type, std::shared_ptr<Resource>,
std::shared_ptr<Condition>>) { ArgumentSetter::ensure_type<typeT>();
::holoscan::gxf::GXFParameterAdaptor::ensure_type<typeT>(); }
params_.try_emplace(key, ParameterWrapper{parameter}); } template <typename
typeT> void ComponentSpec::param(Parameter<typeT>& parameter, const char*
key, const char* headline, const char* description, typeT&& default_value,
ParameterFlag flag) { parameter.key_ = key; parameter.headline_ = headline;
parameter.description_ = description; parameter.flag_ = flag;
parameter.default_value_ = std::move(default_value); if constexpr
(is_one_of_v<typename holoscan::type_info<typeT>::element_type,
std::shared_ptr<Resource>, std::shared_ptr<Condition>>) {
ArgumentSetter::ensure_type<typeT>();
::holoscan::gxf::GXFParameterAdaptor::ensure_type<typeT>(); }
params_.try_emplace(key, ParameterWrapper{parameter}); } } // namespace
holoscan #endif/* HOLOSCAN_CORE_COMPONENT_SPEC_INL_HPP */
```