Program Listing for File component_spec.hpp

Return to documentation for file (include/holoscan/core/component_spec.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_HPP #define
HOLOSCAN_CORE_COMPONENT_SPEC_HPP #include <yaml-cpp/yaml.h> #include
<any> #include <functional> #include <iostream> #include <memory> #include
<string> #include <typeinfo> #include <unordered_map> #include <utility> #include
"./common.hpp" #include "./parameter.hpp" namespace holoscan { class
ComponentSpec { public: explicit ComponentSpec(Fragment* fragment = nullptr) :
fragment_(fragment) {} void fragment(Fragment* fragment) { fragment_ = fragment;
} Fragment* fragment() { return fragment_; } template <typename typeT> void
param(Parameter<typeT>& parameter, const char* key, ParameterFlag flag =
ParameterFlag::kNone) { param(parameter, key, "N/A", "N/A", flag); } template
<typename typeT> void param(Parameter<typeT>& parameter, const char* key,
const char* headline, ParameterFlag flag = ParameterFlag::kNone) {
param(parameter, key, headline, "N/A", flag); } template <typename typeT> void
param(Parameter<typeT>& parameter, const char* key, const char* headline, const
char* description, ParameterFlag flag = ParameterFlag::kNone); template
<typename typeT> void param(Parameter<typeT>& parameter, const char* key,
const char* headline, const char* description, std::initializer_list<void*> init_list);
template <typename typeT> void param(Parameter<typeT>& parameter, const
char* key, const char* headline, const char* description, const typeT&
default_value, ParameterFlag flag = ParameterFlag::kNone); template <typename
typeT> void param(Parameter<typeT>& parameter, const char* key, const char*
headline, const char* description, typeT&& default_value, ParameterFlag flag =
ParameterFlag::kNone); std::unordered_map<std::string, ParameterWrapper>&
params() { return params_; } virtual YAML::Node to_yaml_node() const; std::string
```

description() const; protected: Fragment* fragment_ = nullptr; std::unordered_map<std::string, ParameterWrapper> params_; }; } *// namespace holoscan // -- // Template definitions // // Since the template definitions depends on template methods in other headers, we declare the // template methods above, and define them below with the proper header files, so that we don't // have circular dependencies. // --* #include "./component_spec-inl.hpp" #endif/* HOLOSCAN_CORE_COMPONENT_SPEC_HPP */