Program Listing for File yaml_parser.hpp

Return to documentation for file (include/holoscan/utils/yaml_parser.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_UTILS_YAML_PARSER_HPP #define HOLOSCAN_UTILS_YAML_PARSER_HPP #include <yaml-cpp/yaml.h> #include <iostream> #include <memory> #include <regex> #include <sstream> #include <string> #include <utility> #include <vector> #include "common/yaml_parser.hpp" *// YAML parser for std::complex types* #include "../core/common.hpp" *// Note: GXF provides a custom YAML parser for std::complex types. // // Examples of valid strings are: // "1.0 + 2.5j" // "-1.0 - 3i" // "1+3.3j" // // There may be 0 or 1 space between a + or - sign and the digits. // Either "i" or "j" must appear immediately after the second number.* namespace holoscan { template <typename typeT, typename valueT = void> struct YAMLNodeParser; template <typename typeT> struct YAMLNodeParser<typeT> { static typeT parse(const YAML::Node& node) { try { return node.as<typeT>(); } catch (...) { std::stringstream ss; ss << node; HOLOSCAN_LOG_ERROR("Unable to parse YAML node: '{}'", ss.str()); return typeT(); } } }; template <> struct YAMLNodeParser<int8_t> { static uint8_t parse(const YAML::Node& node) { try { return static_cast<int8_t>(node.as<int32_t>()); } catch (...) { std::stringstream ss; ss << node; HOLOSCAN_LOG_ERROR("Unable to parse YAML node: '{}'", ss.str()); return 0; } } }; template <> struct YAMLNodeParser<uint8_t> { static uint8_t parse(const YAML::Node& node) { try { return static_cast<uint8_t>(node.as<uint32_t>()); } catch (...) { std::stringstream ss; ss << node; HOLOSCAN_LOG_ERROR("Unable to parse YAML node: '{}'", ss.str()); return 0; } } }; template <typename typeT> struct YAMLNodeParser<std::complex<typeT>> { static std::complex<typeT> parse(const YAML::Node& node) { try { return static_cast<std::complex<typeT>> (node.as<std::complex<typeT>>()); } catch (...) { std::stringstream ss; ss << node; HOLOSCAN_LOG_ERROR("Unable to parse YAML node: '{}'", ss.str()); return 0; } } };

```
template <typename typeT> struct YAMLNodeParser<std::vector<typeT>> { static std::vector<typeT> parse(const YAML::Node& node) { if (!node.IsSequence()) { std::stringstream ss; ss << node; HOLOSCAN_LOG_ERROR("Unable to parse YAML node: '{}'. It is not a sequence.", ss.str()); return std::vector<typeT>(); } std::vector<typeT> result(node.size()); for (size_t i = 0; i < node.size(); i++) { const auto value = YAMLNodeParser<typeT>::parse(node[i]); // TODO: check if value is valid result[i] = std::move(value); } return result; } }; template <typename typeT, std::size_t N> struct YAMLNodeParser<std::array<typeT, N>> { static std::array<typeT, N> parse(const YAML::Node& node) { if (!node.IsSequence()) { std::stringstream ss; ss << node; HOLOSCAN_LOG_ERROR("Unable to parse YAML node: '{}'. It is not a sequence.", ss.str()); return std::array<typeT, N>(); } if (node.size() != N) { std::stringstream ss; ss << node; HOLOSCAN_LOG_ERROR( "Unable to parse YAML node: '{}'. It is not a sequence of size {}.", ss.str(), N); return std::array<typeT, N>(); } std::array<typeT, N> result; for (size_t i = 0; i < node.size(); i++) { const auto value = YAMLNodeParser<typeT>::parse(node[i]); // TODO: check if value is valid result[i] = std::move(value); } return result; } }; // Skip std::shared_ptr<Resource> and std::shared_ptr<Condition> template <typename typeT> struct YAMLNodeParser<std::shared_ptr<typeT>> { static std::shared_ptr<typeT> parse(const YAML::Node&) { return {}; } }; // Skip IOSpec* template <> struct YAMLNodeParser<IOSpec*> { static IOSpec* parse(const YAML::Node&) { return nullptr; } }; // Skip std::vector<IOSpec*> template <> struct YAMLNodeParser<std::vector<IOSpec*>> { static std::vector<IOSpec*> parse(const YAML::Node&) { return {}; } }; } // namespace holoscan #endif/* HOLOSCAN_UTILS_YAML_PARSER_HPP */
```

© Copyright 2022-2024, NVIDIA.. PDF Generated on 06/06/2024