Program Listing for File endpoint.hpp

Return to documentation for file (include/holoscan/core/endpoint.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_ENDPOINT_HPP #define HOLOSCAN_CORE_ENDPOINT_HPP #include <gxf/core/gxf.h> #include <iostream> #include <memory> #include <string> #include <utility> #include "./errors.hpp" #include "./expected.hpp" #include "./resource.hpp" #include "./gxf/gxf_component.hpp" #include "./gxf/gxf_utils.hpp" #include "gxf/core/expected.hpp" #include "gxf/serialization/endpoint.hpp" #include "gxf/std/allocator.hpp" *// for nvidia::gxf::MemoryStorageType* namespace holoscan { class Endpoint : public Resource { public: Endpoint() = default; Endpoint(Endpoint&&) = default; ~Endpoint() override = default; explicit Endpoint(nvidia::gxf::Endpoint* gxf_endpoint) : gxf_endpoint_(gxf_endpoint) {} using MemoryStorageType = nvidia::gxf::MemoryStorageType; *// C++ API wrappers* virtual bool is_write_available() { if (!gxf_endpoint_) { throw std::runtime_error("GXF endpoint has not been set"); } return gxf_endpoint_->isWriteAvailable(); } virtual bool is_read_available() { if (!gxf_endpoint_) { throw std::runtime_error("GXF endpoint has not been set"); } return gxf_endpoint_->isReadAvailable(); } virtual expected<size_t, RuntimeError> write(const void* data, size_t size) { if (!gxf_endpoint_) { return make_unexpected<RuntimeError>(RuntimeError(ErrorCode::kCodecError, "GXF endpoint has not been set")); } auto maybe_size = gxf_endpoint_->write(data, size); if (!maybe_size) { *// converted nvidia::gxf::Unexpected to holoscan::unexpected* auto err_msg = fmt::format("GXF endpoint read failure: {}", GxfResultStr(maybe_size.error())); return make_unexpected<RuntimeError> (RuntimeError(ErrorCode::kCodecError, err_msg)); } return maybe_size.value(); } virtual expected<size_t, RuntimeError> read(void* data, size_t size) { if (!gxf_endpoint_) { return make_unexpected<RuntimeError>(

```
RuntimeError(ErrorCode::kCodecError, "GXF endpoint has not been set")); } auto maybe_size = gxf_endpoint_->read(data, size); if (!maybe_size) { // converted nvidia::gxf::Unexpected to holoscan::unexpected auto err_msg = fmt::format("GXF endpoint read failure: {}", GxfResultStr(maybe_size.error())); return make_unexpected<RuntimeError>(RuntimeError(ErrorCode::kCodecError, err_msg)); } return maybe_size.value(); } virtual expected<void, RuntimeError> write_ptr(const void* pointer, size_t size, MemoryStorageType type) { if (!gxf_endpoint_) { return make_unexpected<RuntimeError>( RuntimeError(ErrorCode::kCodecError, "GXF endpoint has not been set")); } auto maybe_void = gxf_endpoint_->write_ptr(pointer, size, type); if (!maybe_void) { // converted nvidia::gxf::Unexpected to holoscan::unexpected auto err_msg = fmt::format("GXF endpoint read failure: {}", GxfResultStr(maybe_void.error())); return make_unexpected<RuntimeError>(RuntimeError(ErrorCode::kCodecError, err_msg)); } return expected<void, RuntimeError>(); } // Note: in GXF, writeTrivialType and readTrivialType below are not on Endpoint itself, but on // SerializationBuffer and UcxSerializationBuffer // Writes an object of type T to the endpoint template <typename T> expected<size_t, RuntimeError> write_trivial_type(const T* object) { return write(object, sizeof(T)); } // Reads an object of type T from the endpoint template <typename T> expected<size_t, RuntimeError> read_trivial_type(T* object) { return read(object, sizeof(T)); } private: nvidia::gxf::Endpoint* gxf_endpoint_; }; } // namespace holoscan #endif/* HOLOSCAN_CORE_ENDPOINT_HPP */
```