Program Listing for File entity.hpp

Return to documentation for file (include/holoscan/core/gxf/entity.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_GXF_ENTITY_HPP #define HOLOSCAN_CORE_GXF_ENTITY_HPP #include <memory> *// Entity definition // Since it has code that causes a warning as an error, we disable it here.* #pragma GCC diagnostic push #pragma GCC diagnostic ignored "-Wmissing-field-initializers" #include "gxf/core/entity.hpp" #pragma GCC diagnostic pop #include "gxf/multimedia/video.hpp" #include "gxf/std/tensor.hpp" #include "holoscan/core/gxf/gxf_utils.hpp" #include "holoscan/core/type_traits.hpp" *// Forward declaration* namespace holoscan { class ExecutionContext; } namespace holoscan::gxf { class Entity : public nvidia::gxf::Entity { public: Entity() = default; explicit Entity(const nvidia::gxf::Entity& other) : nvidia::gxf::Entity(other) {} explicit Entity(nvidia::gxf::Entity&& other) : nvidia::gxf::Entity(other) {} *// Creates a new entity* static Entity New(ExecutionContext* context); operator bool() const { return !is_null(); } *// Gets a component by type. Asserts if no such component.* template <typename DataT, typename = std::enable_if_t<!holoscan::is_vector_v<DataT> && holoscan::is_one_of_v<DataT, holoscan::Tensor>>> std::shared_ptr<DataT> get(const char* name = nullptr, bool log_errors = true) const { *// We should use nullptr as a default name because In GXF, 'nullptr' should be used with // GxfComponentFind() if we want to get the first component of the given type. // Try to get nvidia::gxf::Tensor from GXF Entity.* gxf_tid_t tid; auto tid_result = GxfComponentTypeId(context(), nvidia::TypenameAsString<nvidia::gxf::Tensor>(), &tid); if (tid_result != GXF_SUCCESS) { if (log_errors) { HOLOSCAN_LOG_ERROR("Unable to get component type id from 'nvidia::gxf::Tensor' (error code: {})", tid_result); } return nullptr; } gxf_uid_t cid; auto cid_result = GxfComponentFind(context(), eid(), tid, name, nullptr, &cid); if (cid_result != GXF_SUCCESS) { if (log_errors) { HOLOSCAN_LOG_ERROR("Unable to find component

from the name '{}' (error code: {})", name == nullptr ? "" : name, cid_result); } return nullptr; } *// Create a holoscan::Tensor object from the newly constructed GXF Tensor object. (~680 ns)* auto handle = nvidia::gxf::Handle<nvidia::gxf::Tensor>::Create(context(), cid); auto maybe_dl_ctx = (*handle->get()).toDLManagedTensorContext(); if (!maybe_dl_ctx) { HOLOSCAN_LOG_ERROR("Failed to get std::shared_ptr<DLManagedTensorContext> from nvidia::gxf::Tensor"); return nullptr; } std::shared_ptr<Tensor> tensor = std::make_shared<Tensor>(maybe_dl_ctx.value()); return tensor; } *// Adds a component with given type* template <typename DataT, typename = std::enable_if_t<!holoscan::is_vector_v<DataT> && holoscan::is_one_of_v<DataT, holoscan::Tensor>>> void add(std::shared_ptr<DataT>& data, const char* name = nullptr) { gxf_tid_t tid; HOLOSCAN_GXF_CALL_FATAL(GxfComponentTypeId(context(), nvidia::TypenameAsString<nvidia::gxf::Tensor>(), &tid)); gxf_uid_t cid; HOLOSCAN_GXF_CALL_FATAL(GxfComponentAdd(context(), eid(), tid, name, &cid)); auto handle = nvidia::gxf::Handle<nvidia::gxf::Tensor>::Create(context(), cid); nvidia::gxf::Tensor* tensor_ptr = handle->get(); *// Copy the member data (std::shared_ptr<DLManagedTensorCtx>) from the Tensor to the // nvidia::gxf::Tensor* *tensor_ptr = nvidia::gxf::Tensor(data->dl_ctx()); } }; *// Modified version of the Tensor version of gxf::Entity::get // Retrieves a VideoBuffer instead // TODO: Support gxf::VideoBuffer natively in Holoscan* nvidia::gxf::Handle<nvidia::gxf::VideoBuffer> get_videobuffer(Entity entity, const char* name = nullptr); } *// namespace holoscan::gxf* #endif/* HOLOSCAN_CORE_GXF_ENTITY_HPP */