Program Listing for File dataflow_tracker.hpp

Return to documentation for file (include/holoscan/core/dataflow_tracker.hpp)

```
/* * SPDX-FileCopyrightText: Copyright (c) 2023-2024 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 CORE_DATAFLOW_TRACKER_HPP #define CORE_DATAFLOW_TRACKER_HPP #include <limits.h> #include <algorithm> #include <fstream> #include <iostream> #include <map> #include <memory> #include <mutex> #include <queue> #include <string> #include <unordered_map> #include <vector> #include "./forward_def.hpp" namespace holoscan { constexpr uint64_t kDefaultNumStartMessagesToSkip = 10; constexpr uint64_t kDefaultNumLastMessagesToDiscard = 10; constexpr int kDefaultLatencyThreshold = 0; constexpr uint64_t kDefaultNumBufferedMessages = 100; constexpr const char* kDefaultLogfileName = "logger.log"; enum class DataFlowMetric { kMaxMessageID, kMinMessageID, kMaxE2ELatency, kAvgE2ELatency, kMinE2ELatency, kNumSrcMessages, kNumDstMessages, }; static const std::unordered_map<DataFlowMetric, std::string> metricToString = { {DataFlowMetric::kMaxE2ELatency, "Max end-to-end Latency (ms)"}, {DataFlowMetric::kMaxMessageID, "Max Latency Message No"}, {DataFlowMetric::kAvgE2ELatency, "Avg end-to-end Latency (ms)"}, {DataFlowMetric::kMinE2ELatency, "Min end-to-end Latency (ms)"}, {DataFlowMetric::kMinMessageID, "Min Latency Message No"}, {DataFlowMetric::kNumDstMessages, "Number of messages"}}; class PathMetrics { public: PathMetrics() { path = ""; metrics[DataFlowMetric::kMaxE2ELatency] = INT_MIN; metrics[DataFlowMetric::kMaxMessageID] = -1; metrics[DataFlowMetric::kAvgE2ELatency] = 0; metrics[DataFlowMetric::kMinE2ELatency] = INT_MAX; metrics[DataFlowMetric::kMinMessageID] = -1; metrics[DataFlowMetric::kNumDstMessages] = 0; num_skipped_messages = 0; } uint64_t get_buffer_size(); std::string path; std::unordered_map<DataFlowMetric,
```

```
double> metrics; std::queue<double> latency_buffer; uint64_t
num_skipped_messages; }; class DataFlowTracker { public: DataFlowTracker() {}
~DataFlowTracker(); void set_skip_starting_messages(uint64_t num) {
num_start_messages_to_skip_ = num; } void set_skip_latencies(int threshold); void
set_discard_last_messages(uint64_t num) { num_last_messages_to_discard_ = num; }
void enable_logging(std::string filename = kDefaultLogfileName, uint64_t
num_buffered_messages = kDefaultNumBufferedMessages); void print() const; int
get_num_paths(); std::vector<std::string> get_path_strings(); double
get_metric(std::string pathstring, holoscan::DataFlowMetric metric);
std::map<std::string, uint64_t> get_metric( holoscan::DataFlowMetric metric =
DataFlowMetric::kNumSrcMessages); void end_logging(); protected: // Making
DFFTCollector friend class to access update_latency, // update_source_messages_number,
and write_to_logfile. friend class DFFTCollector; // Making
AnnotatedDoubleBufferReceiver friend class to access update_latency and
write_to_logfile // because the cyclic paths are updated from there, instead of
DFFTCollector friend class AnnotatedDoubleBufferReceiver; void
update_latency(std::string pathstring, double current_latency); void
update_source_messages_number(std::string source, uint64_t num); void
write_to_logfile(std::string text); private: std::map<std::string, uint64_t>
source_messages_; std::mutex source_messages_mutex_; std::map<std::string,
std::shared_ptr<holoscan::PathMetrics>> all_path_metrics_; std::mutex
all_path_metrics_mutex_; uint64_t num_start_messages_to_skip_ =
kDefaultNumStartMessagesToSkip; int latency_threshold_ = 0; uint64_t
num_last_messages_to_discard_ = kDefaultNumLastMessagesToDiscard; bool
is_file_logging_enabled_ = false; std::string logger_filename_; uint64_t
num_buffered_messages_ = 100; std::ofstream logger_ofstream_;
std::vector<std::string> buffered_messages_; std::mutex
buffered_messages_mutex_; uint64_t logfile_messages_ = 0; }; } // namespace
holoscan #endif/* CORE_DATAFLOW_TRACKER_HPP */
```