Program Listing for File flow_graph.hpp

Return to documentation for file (include/holoscan/core/graphs/flow_graph.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_GRAPHS_FLOW_GRAPH_HPP #define HOLOSCAN_CORE_GRAPHS_FLOW_GRAPH_HPP #include <functional> #include <list> #include <memory> #include <set> #include <string> #include <unordered_map> #include <vector> #include "../graph.hpp" namespace holoscan { *// Forward declarations* template <typename NodeT, typename EdgeDataElementT> class FlowGraph; *// Graph type aliases // for operator graph* using OperatorFlowGraph = FlowGraph<OperatorNodeType, OperatorEdgeDataElementType>; *// for fragment graph* using FragmentFlowGraph = FlowGraph<FragmentNodeType, FragmentEdgeDataElementType>; template <typename NodeT = OperatorNodeType, typename EdgeDataElementT = OperatorEdgeDataElementType> class FlowGraph : public Graph<NodeT, EdgeDataElementT> { public: using NodeType = NodeT; using NodePredicate = std::function<bool(const NodeType&)>; using EdgeDataElementType = EdgeDataElementT; using EdgeDataType = std::shared_ptr<EdgeDataElementType>; using Graph<NodeT, EdgeDataElementT>::Graph; ~FlowGraph() override = default; void add_node(const NodeType& node) override; void add_flow(const NodeType& node_u, const NodeType& node_v, const EdgeDataType& port_map) override; std::optional<EdgeDataType> get_port_map(const NodeType& node_u, const NodeType& node_v) override; bool is_root(const NodeType& node) override; bool is_user_defined_root(const NodeType& node) { return get_nodes().empty() ? false : get_nodes()[0] == node; } bool is_leaf(const NodeType& node) override; std::vector<NodeType> has_cycle() override; std::vector<NodeType> get_root_nodes() override; std::vector<NodeType> get_nodes() override; std::vector<NodeType> get_next_nodes(const NodeType& node) override;

```
std::vector<NodeType> get_previous_nodes(const NodeType& node) override; NodeType find_node(const NodePredicate& pred) override; NodeType find_node(const NodeType& node) override; NodeType find_node(std::string name) override; private: std::unordered_map<NodeType, std::unordered_map<NodeType, EdgeDataType>> succ_; std::unordered_map<NodeType, std::unordered_map<NodeType, EdgeDataType>> pred_; std::list<NodeType> ordered_nodes_; std::unordered_map<std::string, NodeType> name_map_; }; } // namespace holoscan #endif/* HOLOSCAN_CORE_GRAPHS_FLOW_GRAPH_HPP */
```