# Program Listing for File graph.hpp

Return to documentation for file ( include/holoscan/core/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_GRAPH_HPP #define HOLOSCAN_CORE_GRAPH_HPP #include <functional> #include <iostream> #include <memory> #include <optional> #include <set> #include <string> #include <unordered_map> #include <vector> #include "./common.hpp" namespace holoscan { *// Forward declarations* class Operator; *// Graph type aliases // for operator graph* using OperatorNodeType = std::shared_ptr<Operator>; using OperatorEdgeDataElementType = std::unordered_map<std::string, std::set<std::string, std::less<>>>; using OperatorGraph = Graph<OperatorNodeType, OperatorEdgeDataElementType>; *// for fragment graph* using FragmentNodeType = std::shared_ptr<Fragment>; using FragmentEdgeDataElementType = std::unordered_map<std::string, std::set<std::string, std::less<>>>; using FragmentGraph = Graph<FragmentNodeType, FragmentEdgeDataElementType>; template <typename NodeT = OperatorNodeType, typename EdgeDataElementT = OperatorEdgeDataElementType> class Graph { public: using NodeType = NodeT; using NodePredicate = std::function<bool(const NodeT&)>; using EdgeDataElementType = EdgeDataElementT; using EdgeDataType = std::shared_ptr<EdgeDataElementT>; Graph() = default; virtual ~Graph() = default; virtual void add_node(const NodeT& node) = 0; virtual void add_flow(const NodeType& node_u, const NodeType& node_v, const EdgeDataType& port_map) = 0; virtual std::optional<EdgeDataType> get_port_map(const NodeType& node_u, const NodeType& node_v) = 0; virtual bool is_empty() { return !find_node([](const NodeType&) { return true; }); } virtual bool is_root(const NodeType& node) = 0; virtual bool is_user_defined_root(const NodeType& node) = 0; virtual bool is_leaf(const NodeType& node) = 0; virtual std::vector<NodeType> has_cycle() = 0;

```
virtual std::vector<NodeType> get_root_nodes() = 0; virtual std::vector<NodeType> get_nodes() = 0; virtual std::vector<NodeType> get_next_nodes(const NodeType& node) = 0; virtual NodeType find_node(const NodePredicate& pred) = 0; virtual NodeType find_node(const NodeType& node) = 0; virtual NodeType find_node(std::string name) = 0; virtual std::vector<NodeType> get_previous_nodes(const NodeType& node) = 0; virtual void context(void* context) { context_ = context; } virtual void* context() { return context_; } protected: void* context_ = nullptr; }; } // namespace holoscan #endif/* HOLOSCAN_CORE_GRAPH_HPP */
```