rmf_ros2

Release 1.0.0

Open Source Robotics Corporation

Mar 20, 2022

CONTENTS:

Packages that implement OpenRMF's distributed architecture using ROS 2.

CHAPTER

ONE

RMF_ROS2 API

1.1 Class Hierarchy

1.2 File Hierarchy

1.3 Full API

1.3.1 Namespaces

Namespace rmf_fleet_adapter

Contents

Namespaces

Variables

Namespace rmf_fleet_adapter::agv

Contents

Namespaces

- *Namespace rmf_fleet_adapter::agv::test*

Classes

- *Class Adapter*
- *Class EasyTrafficLight*
- *Class EasyTrafficLight::Blocker*
- *Class FleetUpdateHandle*
- *Class FleetUpdateHandle::Confirmation*
- *Class RobotCommandHandle*
- *Class RobotUpdateHandle*
- *Class RobotUpdateHandle::ActionExecution*
- *Class RobotUpdateHandle::Interruption*
- *Class RobotUpdateHandle::IssueTicket*
- *Class RobotUpdateHandle::Unstable*
- *Class Waypoint*

Functions

- *Function rmf_fleet_adapter::agv::parse_graph*

Typedefs

- *Typedef rmf_fleet_adapter::agv::AdapterPtr*
- *Typedef rmf_fleet_adapter::agv::ConstAdapterPtr*
- *Typedef rmf_fleet_adapter::agv::ConstFleetUpdateHandlePtr*
- *Typedef rmf_fleet_adapter::agv::ConstRobotUpdateHandlePtr*
- *Typedef rmf_fleet_adapter::agv::EasyTrafficLightPtr*
- *Typedef rmf_fleet_adapter::agv::FleetUpdateHandlePtr*
- *Typedef rmf_fleet_adapter::agv::RobotUpdateHandlePtr*

Namespace rmf_fleet_adapter::agv::test

Contents

Classes

Namespace rmf_task_ros2

Contents

Namespaces

Classes

Functions

Typedefs

- *Typedef rmf_task_ros2::DispatchStatePtr*
- *Typedef rmf_task_ros2::TaskID*

Variables

- *Variable rmf_task_ros2::BidNoticeTopicName*
- *Variable rmf_task_ros2::BidResponseTopicName*
- *Variable rmf_task_ros2::CancelTaskSrvName*
- *Variable rmf_task_ros2::DispatchAckTopicName*
- *Variable rmf_task_ros2::DispatchCommandTopicName*
- *Variable rmf_task_ros2::DispatchStatesTopicName*
- *Variable rmf_task_ros2::GetDispatchStatesSrvName*
- *Variable rmf_task_ros2::Prefix*
- *Variable rmf_task_ros2::SubmitTaskSrvName*
- *Variable rmf_task_ros2::TaskStatusTopicName*

Namespace rmf_task_ros2::bidding

Contents

Classes

- *Struct Response*
- *Struct Response::Proposal*
- *Class AsyncBidder*
- *Class Auctioneer*
- *Class Auctioneer::Evaluator*
- *Class LeastFleetCostEvaluator*
- *Class LeastFleetDiffCostEvaluator*
- *Class QuickestFinishEvaluator*

Functions

- *Function rmf_task_ros2::bidding::convert(const BidResponseMsg&)*
- *Function rmf_task_ros2::bidding::convert(const Response::Proposal&)*
- *Function rmf_task_ros2::bidding::convert(const Response&, const std::string&)*

Typedefs

- *Typedef rmf_task_ros2::bidding::BidNoticeMsg*
- *Typedef rmf_task_ros2::bidding::BidProposalMsg*
- *Typedef rmf_task_ros2::bidding::BidResponseMsg*
- *Typedef rmf_task_ros2::bidding::Responses*

Namespace rmf_traffic_ros2

Contents

Namespaces

- *Namespace rmf_traffic_ros2::blockade*
- *Namespace rmf_traffic_ros2::geometry*
- *Namespace rmf_traffic_ros2::schedule*

Functions

- *Function rmf_traffic_ros2::convert(rclcpp::Duration)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::Trajectory&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic::Trajectory&)*
- *Function rmf_traffic_ros2::convert(const rmf_site_map_msgs::msg::SiteMap&, int, double)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic::geometry::Circle&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::Circle&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ConvexShapeContext&)*
- *Function rmf_traffic_ros2::convert(const geometry::ConvexShapeContext&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ShapeContext&)*
- *Function rmf_traffic_ros2::convert(const geometry::ShapeContext&)*

- *Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::Profile&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic::Profile&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::Route&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic::Route&)*
- *Function rmf_traffic_ros2::convert(const std::vector<rmf_traffic_msgs::msg::Route>&)*
- *Function rmf_traffic_ros2::convert(const std::vector<rmf_traffic::Route>&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ScheduleChangeAddItem&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Change::Add::Item&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ScheduleChangeAdd&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Change::Add&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ScheduleChangeDelay&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Change::Delay&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Change::UnregisterParticipant&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ScheduleChangeCull&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Change::Cull&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Inconsistencies::Element&, const rmf_traffic::schedule::ProgressVersion)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Itinerary&)*
- *Function rmf_traffic_ros2::convert(const std::vector<rmf_traffic_msgs::msg::Itinerary>&)*
- *Function rmf_traffic_ros2::convert(const std::vector<rmf_traffic::schedule::Itinerary>&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ParticipantDescription&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::ParticipantDescription&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::Participants&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::ParticipantDescriptionsMap&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Patch::Participant&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ScheduleParticipantPatch&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Patch&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::SchedulePatch&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ScheduleQuerySpacetime&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Query::Spacetime&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ScheduleQueryParticipants&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Query::Participants&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ScheduleQuery&)*
- *Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Query&)*
- *Function rmf_traffic_ros2::convert(rmf_traffic::Time)*
- *Function rmf_traffic_ros2::convert(builtin_interfaces::msg::Time)*

- *Function rmf_traffic_ros2::convert(rclcpp::Time)*
- *Function rmf_traffic_ros2::convert(rmf_traffic::Duration)*
- *Function rmf_traffic_ros2::to_ros2*

Variables

- *Variable rmf_traffic_ros2::BlockadeCancelTopicName*
- *Variable rmf_traffic_ros2::BlockadeHeartbeatTopicName*
- *Variable rmf_traffic_ros2::BlockadeReachedTopicName*
- *Variable rmf_traffic_ros2::BlockadeReadyTopicName*
- *Variable rmf_traffic_ros2::BlockadeReleaseTopicName*
- *Variable rmf_traffic_ros2::BlockadeSetTopicName*
- *Variable rmf_traffic_ros2::EmergencyTopicName*
- *Variable rmf_traffic_ros2::FailOverEventTopicName*
- *Variable rmf_traffic_ros2::HeartbeatTopicName*
- *Variable rmf_traffic_ros2::ItineraryClearTopicName*
- *Variable rmf_traffic_ros2::ItineraryDelayTopicName*
- *Variable rmf_traffic_ros2::ItineraryExtendTopicName*
- *Variable rmf_traffic_ros2::ItineraryReachedTopicName*
- *Variable rmf_traffic_ros2::ItinerarySetTopicName*
- *Variable rmf_traffic_ros2::NegotiationAckTopicName*
- *Variable rmf_traffic_ros2::NegotiationConclusionTopicName*
- *Variable rmf_traffic_ros2::NegotiationForfeitTopicName*
- *Variable rmf_traffic_ros2::NegotiationNoticeTopicName*
- *Variable rmf_traffic_ros2::NegotiationProposalTopicName*
- *Variable rmf_traffic_ros2::NegotiationRefusalTopicName*
- *Variable rmf_traffic_ros2::NegotiationRejectionTopicName*
- *Variable rmf_traffic_ros2::NegotiationRepeatTopicName*
- *Variable rmf_traffic_ros2::ParticipantsInfoTopicName*
- *Variable rmf_traffic_ros2::Prefix*
- *Variable rmf_traffic_ros2::QueriesInfoTopicName*
- *Variable rmf_traffic_ros2::QueryUpdateTopicNameBase*
- *Variable rmf_traffic_ros2::RegisterParticipantSrvName*
- *Variable rmf_traffic_ros2::RegisterQueryServiceName*
- *Variable rmf_traffic_ros2::RequestChangesServiceName*
- *Variable rmf_traffic_ros2::ScheduleInconsistencyTopicName*
- *Variable rmf_traffic_ros2::UnregisterParticipantSrvName*

Namespace rmf_traffic_ros2::blockade

Contents

Classes

Functions

Typedefs

Namespace rmf_traffic_ros2::geometry

Contents

Classes

Namespace rmf_traffic_ros2::schedule

Contents

Classes

- *Struct AtomicOperation*
- *Class AbstractParticipantLogger*
- *Class MirrorManager*
- *Class MirrorManager::Options*
- *Class MirrorManagerFuture*
- *Class Negotiation*
- *Class Negotiation::Worker*
- *Class ParticipantRegistry*
- *Class Writer*
- *Class YamlLogger*

Functions

- *Function rmf_traffic_ros2::schedule::make_mirror*
- *Function rmf_traffic_ros2::schedule::make_monitor_node*
- *Function rmf_traffic_ros2::schedule::make_node*

Typedefs

- *Typedef rmf_traffic_ros2::schedule::Database*
- *Typedef rmf_traffic_ros2::schedule::ParticipantDescription*
- *Typedef rmf_traffic_ros2::schedule::ParticipantId*
- *Typedef rmf_traffic_ros2::schedule::WriterPtr*

Namespace std::chrono_literals

1.3.2 Classes and Structs

Struct Response

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_bidding_Response.hpp

Nested Relationships

Nested Types

- *Struct Response::Proposal*

Struct Documentation

struct rmf_task_ros2::bidding::**Response**

Public Members

std::optional<*Proposal*> **proposal**

std::vector<std::string> **errors**

struct Proposal

Public Members

std::string **fleet_name**

std::string **expected_robot_name**

double **prev_cost**

double **new_cost**

rmf_traffic::Time **finish_time**

Struct Response::Proposal

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_bidding_Response.hpp

Nested Relationships

This struct is a nested type of *Struct Response*.

Struct Documentation

struct rmf_task_ros2::bidding::*Response*::**Proposal**

Public Members

std::string **fleet_name**

std::string **expected_robot_name**

double **prev_cost**

double **new_cost**

rmf_traffic::Time **finish_time**

Struct DispatchState

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_DispatchState.hpp

Nested Relationships

Nested Types

- *Struct DispatchState::Assignment*

Struct Documentation

struct rmf_task_ros2**::DispatchState**

Note TaskStatus struct is based on TaskSummary.msg

Public Types

enum Status

Values:

enumerator Queued
This task has not been assigned yet.

enumerator Selected
An assignment has been selected, but we have not received acknowledgment from the assignee

enumerator Dispatched
The task is dispatched and no longer being managed by the dispatcher.

enumerator FailedToAssign
There was a failure to assign the task to anyone.

enumerator CanceledInFlight
The task was canceled before it managed to get dispatched.

using Msg = rmf_task_msgs::msg::DispatchState

Public Functions

DispatchState(std::string *task_id*, rmf_traffic::Time *submission_time*)

Public Members

std::string **task_id**
The Task ID for that this dispatching refers to.

rmf_traffic::Time **submission_time**
Submission arrival time.

Status **status** = *Status*::*Queued*
The status of this dispatching.

std::optional<*Assignment*> **assignment**
The assignment that was made for this dispatching.

std::vector<nlohmann::json> **errors**
Any errors that have occurred for this dispatching.

struct Assignment

Public Members

std::string **fleet_name**

std::string **expected_robot_name**

Struct DispatchState::Assignment

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_DispatchState.hpp

Nested Relationships

This struct is a nested type of *Struct DispatchState*.

Struct Documentation

struct rmf_task_ros2::*DispatchState*::**Assignment**

Public Members

std::string **fleet_name**

std::string **expected_robot_name**

Struct AtomicOperation

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_ParticipantRegistry.hpp

Struct Documentation

struct rmf_traffic_ros2::schedule::**AtomicOperation**
This records a single operation on the database class.

Public Types

enum OpType
Values:

enumerator Add

enumerator Update

Public Members

OpType **operation**

ParticipantDescription **description**

Class PyConvexShape

- Defined in file_latest_rmf_fleet_adapter_python_include_rmf_fleet_adapter_python_geometry_PyConvexShape.hpp

Inheritance Relationships

Base Type

- public ConvexShape

Class Documentation

class PyConvexShape : **public** ConvexShape

Public Functions

inline geometry::FinalConvexShape **finalize_convex**() **const override**

Class PyEvent

- Defined in file_latest_rmf_fleet_adapter_python_include_rmf_fleet_adapter_python_graph_PyEvent.hpp

Inheritance Relationships

Base Type

- `public Event`

Class Documentation

class PyEvent : **public** Event

Public Functions

inline Duration **duration**() **const override**

inline *Lane*::Executor &**execute**(*Lane*::Executor &*executor*) **const override**

inline *EventPtr* **clone**() **const override**

Class PyExecutor

- Defined in file_latest_rmf_fleet_adapter_python_include_rmf_fleet_adapter_python_graph_PyExecutor.hpp

Inheritance Relationships

Base Type

- `public Executor`

Class Documentation

class PyExecutor : **public** Executor

Public Functions

inline void **execute**(**const** *Lane*::DoorOpen &*open*) **override**

inline void **execute**(**const** *Lane*::DoorClose &*close*) **override**

inline void **execute**(**const** *Lane*::LiftSessionBegin &*begin*) **override**

inline void **execute**(**const** *Lane*::LiftDoorOpen &*open*) **override**

inline void **execute**(**const** *Lane*::LiftSessionEnd &*end*) **override**

inline void **execute**(**const** *Lane*::LiftMove &*move*) **override**

inline void **execute**(**const** *Lane*::Dock &*dock*) **override**

inline void **execute**(**const** *Lane*::Wait &*wait*) **override**

Class PyFinalConvexShape

- Defined in file_latest_rmf_fleet_adapter_python_include_rmf_fleet_adapter_python_geometry_PyConvexShape.hpp

Inheritance Relationships

Base Type

- `public FinalConvexShape`

Class Documentation

class PyFinalConvexShape : **public** FinalConvexShape

Class PyFinalShape

- Defined in file_latest_rmf_fleet_adapter_python_include_rmf_fleet_adapter_python_geometry_PyShape.hpp

Inheritance Relationships

Base Type

- `public FinalShape`

Class Documentation

class PyFinalShape : **public** FinalShape

Class PyOrientationConstraint

- Defined in file_latest_rmf_fleet_adapter_python_include_rmf_fleet_adapter_python_graph_PyOrientationConstraint.hpp

Inheritance Relationships

Base Type

- `public OrientationConstraint`

Class Documentation

class PyOrientationConstraint : **public** OrientationConstraint

Public Functions

inline bool **apply**(Eigen::Vector3d *&position*, **const** Eigen::Vector2d *&course_vector*) **const override**

inline rmf_utils::clone_ptr<OrientationConstraint> **clone**() **const override**

Class PyRobotCommandHandle

- Defined in file_latest_rmf_fleet_adapter_python_include_rmf_fleet_adapter_python_PyRobotCommandHandle.hpp

Inheritance Relationships

Base Type

- `public rmf_fleet_adapter::agv::RobotCommandHandle` (*Class RobotCommandHandle*)

Class Documentation

class PyRobotCommandHandle : **public** rmf_fleet_adapter::agv::*RobotCommandHandle*

Public Types

using ArrivalEstimator = std::function<void(std::size_t path_index, rmf_traffic::Duration remaining_time)>

Public Functions

inline void **follow_new_path**(**const** std::vector<rmf_traffic::agv::Plan::Waypoint> *&waypoints*, *ArrivalEstimator next_arrival_estimator*, std::function<void()> *path_finished_callback*) **override**

inline virtual void **stop**() **override**
Have the robot come to an immediate stop.

inline virtual void **dock**(**const** std::string *&dock_name*, std::function<void()> *docking_finished_callback*) **override**Have the robot begin a pre-defined docking procedure. Implement this function as a no-op if your robots do not perform docking procedures.

Parameters

- `[in] dock_name`: The predefined name of the docking procedure to use.
- `[in] docking_finished_callback`: Trigger this callback when the docking is finished.

Class PyShape

- Defined in file_latest_rmf_fleet_adapter_python_include_rmf_fleet_adapter_python_geometry_PyShape.hpp

Inheritance Relationships

Base Type

- `public Shape`

Class Documentation

class PyShape : **public** Shape

Public Functions

inline geometry::FinalShape **finalize**() **const override**

Class Adapter

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_Adapter.hpp

Inheritance Relationships

Base Type

- `public std::enable_shared_from_this< Adapter >`

Class Documentation

class rmf_fleet_adapter::agv::**Adapter** : **public** std::enable_shared_from_this<*Adapter*>

Public Types

using Blockers = std::vector<*EasyTrafficLight*::*Blocker*>

Public Functions

std::shared_ptr<*FleetUpdateHandle*> **add_fleet**(**const** std::string &*fleet_name*, rmf_traffic::agv::VehicleTraits *traits*, rmf_traffic::agv::Graph *navigation_graph*, std::optional<std::string> *server_uri* = std::nullopt)

Add a fleet to be adapted.

If a single real-life fleet needs to integrate robots with varying traits or with different navigation graphs, it is okay to call this function multiple times with the same fleet_name and add a robot using whichever handle has the traits and navigation graph that match the robot.

Parameters

- `[in] fleet_name`: The name of the fleet that is being added.
- `[in] traits`: Specify the approximate traits of the vehicles in this fleet.
- `[in] navigation_graph`: Specify the navigation graph used by the vehicles in this fleet.
- `[in] server_uri`: Specify the URI for the websocket server that receives updates on tasks and states. If nullopt, data will not be published.

void **add_easy_traffic_light**(std::function<void(*EasyTrafficLightPtr* handle>) *handle_callback*, **const** std::string &*fleet_name*, **const** std::string &*robot_name*, rmf_traffic::agv::VehicleTraits *traits*, std::function<void()> *pause_callback*std::function<void()> *resume_callback*std::function<void(*Blockers*)> *deadlock_callback* = nullptrCreate an easy-to-use version of a traffic light to help manage robots that can only support pause and resume commands.

This API is much simpler to use than the standard traffic light API, but it provides less information about the exact timing needed for the starts and stops.

This API should only be used for demo purposes, or if system integrators can ensure very low-latency and reliable connections to the robots to ensure that the commands arrive on time.

Parameters

- `[in] handle_callback`: The callback that will be triggered when the *EasyTrafficLight* handle is ready to be used. This callback will only be triggered once.
- `[in] fleet_name`: The name of the fleet
- `[in] robot_name`: The name of the robot
- `[in] traits`: The traits of the robot
- `[in] pause_callback`: The callback that should be triggered by the traffic light when an immediate pause is needed.
- `[in] resume_callback`: The callback that will be triggered by the traffic light when the robot may resume moving forward.
- `[in] deadlock_callback`: The callback that will be triggered by the traffic light if there is a permanent blocker disrupting the ability of this vehicle to proceed. Manual intervention may be required in this circumstance. A callback does not need to be provided for this. Either way, an error message will be printed to the log.

std::shared_ptr<rclcpp::Node> **node**()
Get the rclcpp::Node that this adapter will be using for communication.

std::shared_ptr<**const** rclcpp::Node> **node**() **const**
const-qualified *node()*

Adapter &**start**()
Begin running the event loop for this adapter. The event loop will operate in another thread, so this function is non-blocking.

Adapter &**stop**()
Stop the event loop if it is running.

Adapter &**wait**()
Wait until the adapter is done spinning.

See *wait_for()*

Adapter &**wait_for**(std::chrono::nanoseconds *max_wait*)
Wait until the adapter is done spinning, or until the maximum wait time duration is reached.

See *wait()*

Public Static Functions

static std::shared_ptr<*Adapter*> **init_and_make**(**const** std::string &*node_name*, std::optional<rmf_traffic::Duration> *discovery_timeout* = std::nullopt)
Initialize an rclcpp context and make an adapter instance. This will instantiate an rclcpp::Node and allow you to add fleets to be adapted.

This is an easier-to-use but less customizable alternative to *make()*.

See *make()*

Parameters

- `[in] node_name`: The name for the rclcpp::Node that will be produced for this *Adapter*.
- `[in] discovery_timeout`: How long we will wait to discover the Schedule Node before giving up. If rmf_utils::nullopt is given, then this will try to use the discovery_timeout node paramter, or it will wait 1 minute if the discovery_timeout node parameter was not defined.

static std::shared_ptr<*Adapter*> **make**(**const** std::string &*node_name*, **const** rclcpp::NodeOptions &*node_options* = rclcpp::NodeOptions(), std::optional<rmf_traffic::Duration> *discovery_timeout* = std::nullopt)
Make an adapter instance. This will instantiate an rclcpp::Node and allow you to add fleets to be adapted.

Note You must initialize rclcpp before calling this, either by using rclcpp::init(~) or rclcpp::Context::init(~). This requirement can be avoided by using *init_and_make()* instead of this function.

See *init_and_make()*

Parameters

- `[in] node_name`: The name for the rclcpp::Node that will be produced for this *Adapter*.
- `[in] node_options`: The options that the rclcpp::Node will be constructed with.
- `[in] discovery_timeout`: How long we will wait to discover the Schedule Node before giving up. If rmf_utils::nullopt is given, then this will try to use the discovery_timeout node paramter, or it will wait 1 minute if the discovery_timeout node parameter was not defined.

Class EasyTrafficLight

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_EasyTrafficLight.hpp

Nested Relationships

Nested Types

- *Class EasyTrafficLight::Blocker*

Inheritance Relationships

Base Type

- `public std::enable_shared_from_this< EasyTrafficLight >`

Class Documentation

class `rmf_fleet_adapter::agv::`**EasyTrafficLight** : **public** std::enable_shared_from_this<*EasyTrafficLight*>

Public Types

enum MovingInstruction
This instruction is given for moving updates. It.

Values:

enumerator MovingError
This indicates that your robot has not obeyed its instructions to stop. When this happens, it could mean serious problems for the overall traffic light system, including permanent deadlocks with other robots. This error may be seen if moving_from(~) is called during the time gap between the robot being instructed to pause and the feedback from the robot that it has paused.

enumerator ContinueAtNextCheckpoint
When the robot reaches the next checkpoint, it should continue.

enumerator WaitAtNextCheckpoint
When the robot reaches the next checkpoint, it must wait.

enumerator PauseImmediately
The robot should pause immediately. This typically means there has been a change of plans and now the robot is scheduled to give way to another.

enum WaitingInstruction
Values:

enumerator WaitingError
This indicates that your robot has not obeyed its instructions to stop. When this happens, it could mean serious problems for the overall traffic light system, including permanent deadlocks with other robots.

enumerator Resume
The robot can continue along its path. It no longer needs to wait here.

enumerator Wait
The robot must continue waiting here.

Public Functions

void **follow_new_path**(**const** std::vector<*Waypoint*> &*new_path*)
Update the traffic light with a new path for your robot.

Warning This function will throw an exception if there are less than 2 waypoints in the path.

MovingInstruction **moving_from**(std::size_t *checkpoint*, Eigen::Vector3d *location*)
Tell the traffic light that the robot is moving.

Return what the robot should do when it reaches its next checkpoint. This may change in between calls to this function. The results may even change from ContinueAtNextCheckpoint to WaitAtNextCheckpoint if a negotiation decided to have this robot give way to another robot. You must always use the latest value received from this function.

Warning This function should only be called if the system has enough time for the robot to stop at the next checkpoint (i.e. accounting for the network latency of sending out the stop command and the maximum deceleration of the robot). The expectation is that if this function returns a WaitAtNextCheckpoint instruction, then the robot will definitely wait at the next checkpoint (until instructed otherwise). If that expectation is violated, you may get MovingError and/or WaitingError results, and the overall traffic flow may get interrupted or deadlocked.

Note If your robot might not be able to stop in time to wait at the next checkpoint, then call moving_from(checkpoint+1, location) instead, even if your robot has not physically reached checkpoint+1 yet.

Parameters

- `[in] checkpoint`: The last checkpoint which the robot passed over.
- `[in] location`: The current location of the robot.

WaitingInstruction **waiting_at**(std::size_t *checkpoint*)
Tell the traffic light that the robot is waiting at a checkpoint.

Return whether the robot should resume its travel or keep waiting.

Parameters

- `[in] checkpoint`: The checkpoint where the robot is waiting.

WaitingInstruction **waiting_after**(std::size_t *checkpoint*, Eigen::Vector3d *location*)
Tell the traffic light that the robot is waiting at a location in-between waypoints.

Parameters

- `[in] checkpoint`: The last checkpoint that the robot passed.
- `[in] location`: The location where the robot is waiting.

std::size_t **last_reached**() **const**
Get the last checkpoint that the traffic light knows has been reached.

EasyTrafficLight &**update_idle_location**(std::string *map_name*, Eigen::Vector3d *position*)
Update the location of the robot while it is idle. This means the robot is sitting somewhere without the intention of going anywhere.

Parameters

- `[in] map_name`: The name of the reference map where the robot is located.
- `[in] position`: The (x, y, yaw) coordinates of the robot.

EasyTrafficLight &**update_battery_soc**(double *battery_soc*)
Update the current battery level of the robot by specifying its state of charge as a fraction of its total charge capacity.

EasyTrafficLight &**replan**()
Tell the fleet adapter to replan. This can help to break out of deadlocks.

EasyTrafficLight &**fleet_state_publish_period**(std::optional<rmf_traffic::Duration> *value*)
Specify a period for how often the fleet state message is published for this fleet. Passing in std::nullopt will disable the fleet state message publishing. The default value is 1s.

class Blocker
This class will be provided to the deadlock_callback when a deadlock has occurred due to an unresolvable conflict. Human intervention may be required at this point, because the RMF traffic negotiation system does not have a high enough level of control over the conflicting participants to resolve it.

Public Functions

rmf_traffic::schedule::ParticipantId **participant_id**() **const**
Get the schedule participant ID of the blocker.

const rmf_traffic::schedule::ParticipantDescription &**description**() **const**
Get the description of the blocker.

Class EasyTrafficLight::Blocker

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_EasyTrafficLight.hpp

Nested Relationships

This class is a nested type of *Class EasyTrafficLight*.

Class Documentation

class `rmf_fleet_adapter::agv::`*`EasyTrafficLight`*`::Blocker`
This class will be provided to the deadlock_callback when a deadlock has occurred due to an unresolvable conflict. Human intervention may be required at this point, because the RMF traffic negotiation system does not have a high enough level of control over the conflicting participants to resolve it.

Public Functions

rmf_traffic::schedule::ParticipantId **participant_id**() **const**
Get the schedule participant ID of the blocker.

const rmf_traffic::schedule::ParticipantDescription &**description**() **const**
Get the description of the blocker.

Class FleetUpdateHandle

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_FleetUpdateHandle.hpp

Nested Relationships

Nested Types

- *Class FleetUpdateHandle::Confirmation*

Inheritance Relationships

Base Type

- `public std::enable_shared_from_this< FleetUpdateHandle >`

Class Documentation

class `rmf_fleet_adapter::agv::`**FleetUpdateHandle** : **public** std::enable_shared_from_this<*FleetUpdateHandle*>

Public Types

using ConsiderRequest = std::function<void (**const** nlohmann::json &description, *Confirmation* &confirm)>
Signature for a callback that decides whether to accept a specific category of task request.

Parameters

- `[in] description`: A description of the task that is being considered
- `[in] confirm`: Use this object to decide if you want to accept the task

using AcceptTaskRequest = std::function<bool (**const** rmf_task_msgs::msg::TaskProfile &profile)>
A callback function that evaluates whether a fleet will accept a task request

Return true to indicate that this fleet should accept the request, false to reject the request.

Parameters

- `[in] request`: Information about the task request that is being considered.

using AcceptDeliveryRequest = std::function<bool (**const** rmf_task_msgs::msg::Delivery &request) >

A callback function that evaluates whether a fleet will accept a delivery request.

Return true to indicate that this fleet should accept the request, false to reject the request.

Parameters

- `[in] request`: Information about the delivery request that is being considered.

Public Functions

void **add_robot** (std::shared_ptr<*RobotCommandHandle*> *command*, **const** std::string &*name*, **const** rmf_traffic::Profile &*profile*, rmf_traffic::agv::Plan::StartSet *start*, std::function<void) std::shared_ptr<*RobotUpdateHandle*> handle > *handle_cb*Add a robot to this fleet adapter.

Return a handle to give the adapter updates about the robot.

Parameters

- `[in] command`: A reference to a command handle for this robot.
- `[in] name`: The name of this robot.
- `[in] profile`: The profile of this robot. This profile should account for the largest possible payload that the robot might carry.
- `[in] start`: The initial location of the robot, expressed as a Plan::StartSet. Multiple Start objects might be needed if the robot is not starting precisely on a waypoint. The function rmf_traffic::agv::compute_plan_starts() may help with this.
- `[in] handle_cb`: This callback function will get triggered when the *RobotUpdateHandle* is ready to be used by the Fleet API side of the *Adapter*. Setting up a new robot requires communication with the Schedule Node, so there may be a delay before the robot is ready to be used.

FleetUpdateHandle &**consider_delivery_requests** (*ConsiderRequest* *consider_pickup*, *ConsiderRequest* *consider_dropoff*)

Allow this fleet adapter to consider delivery requests.

Pass in nullptrs to disable delivery requests.

By default, delivery requests are not accepted until you provide these callbacks.

The *FleetUpdateHandle* will ensure that the requests are feasible for the robots before triggering these callbacks.

Parameters

- `[in] consider_pickup`: Decide whether to accept a pickup request. The description will satisfy the event_description_PickUp.json schema of rmf_fleet_adapter.
- `[in] consider_dropoff`: Decide whether to accept a dropoff request. The description will satisfy the event_description_DropOff.json schema of rmf_fleet_adapter.

FleetUpdateHandle &**consider_cleaning_requests** (*ConsiderRequest* *consider*)

Allow this fleet adapter to consider cleaning requests.

Pass in a nullptr to disable cleaning requests.

By default, cleaning requests are not accepted until you provide this callback.

Parameters

- `[in] consider`: Decide whether to accept a cleaning request. The description will satisfy the event_description_Clean.json schema of rmf_fleet_adapter. The *FleetUpdateHandle* will ensure that the request is feasible for the robots before triggering this callback.

FleetUpdateHandle &**consider_patrol_requests**(*ConsiderRequest consider*)

Allow this fleet adapter to consider patrol requests.

Pass in a nullptr to disable patrol requests.

By default, patrol requests are always accepted.

Parameters

- `[in] consider`: Decide whether to accept a patrol request. The description will satisfy the task_description_Patrol.json schema of rmf_fleet_adapter. The *FleetUpdateHandle* will ensure that the request is feasible for the robots before triggering this callback.

FleetUpdateHandle &**consider_composed_requests**(*ConsiderRequest consider*)

Allow this fleet adapter to consider composed requests.

Pass in a nullptr to disable composed requests.

By default, composed requests are always accepted, as long as the events that they are composed of are accepted.

Parameters

- `[in] consider`: Decide whether to accept a composed request. The description will satisfy the task_description_Compose.json schema of rmf_fleet_adapter. The *FleetUpdateHandle* will ensure that the request is feasible for the robots before triggering this callback.

FleetUpdateHandle &**add_performable_action**(**const** std::string &*category*, *ConsiderRequest consider*)

Allow this fleet adapter to execute a PerformAction activity of specified category which may be present in sequence event.

Parameters

- `[in] category`: A string that categorizes the action. This value should be used when filling out the category field in event_description_PerformAction.json schema.
- `[in] consider`: Decide whether to accept the action based on the description field in event_description_PerformAction.json schema.

void **close_lanes**(std::vector<std::size_t> *lane_indices*)

Specify a set of lanes that should be closed.

void **open_lanes**(std::vector<std::size_t> *lane_indices*)

Specify a set of lanes that should be open.

bool **set_task_planner_params**(std::shared_ptr<rmf_battery::agv::BatterySystem> *battery_system*, std::shared_ptr<rmf_battery::MotionPowerSink> *motion_sink*, std::shared_ptr<rmf_battery::DevicePowerSink> *ambient_sink*, std::shared_ptr<rmf_battery::DevicePowerSink> *tool_sink*, double *recharge_threshold*, double *recharge_soc*, bool *account_for_battery_drain*, rmf_task::ConstRequestFactoryPtr *finishing_requst* = nullptr)

Set the parameters required for task planning. Without calling this function, this fleet will not bid for and accept tasks.

Return true if task planner parameters were successfully updated.

Parameters

- `[in] battery_system`: Specify the battery system used by the vehicles in this fleet.
- `[in] motion_sink`: Specify the motion sink that describes the vehicles in this fleet.
- `[in] ambient_sink`: Specify the device sink for ambient sensors used by the vehicles in this fleet.
- `[in] tool_sink`: Specify the device sink for special tools used by the vehicles in this fleet.
- `[in] recharge_threshold`: The threshold for state of charge below which robots in this fleet will cease to operate and require recharging. A value between 0.0 and 1.0 should be specified.
- `[in] recharge_soc`: The state of charge to which robots in this fleet should be charged up to by automatic recharging tasks. A value between 0.0 and 1.0 should be specified.
- `[in] account_for_battery_drain`: Specify whether battery drain is to be considered while allocating tasks. If false, battery drain will not be considered when planning for tasks. As a consequence, charging tasks will not be automatically assigned to vehicles in this fleet when battery levels fall below the recharge_threshold.
- `[in] finishing_request`: A factory for a request that should be performed by each robot in this fleet at the end of its assignments.

FleetUpdateHandle &**accept_task_requests**(*AcceptTaskRequest check*)

Provide a callback that indicates whether this fleet will accept a BidNotice request. By default all requests will be rejected.

Note The callback function that you give should ideally be non-blocking and return quickly. It's meant to check whether this fleet's vehicles are compatible with the requested payload, pickup, and dropoff behavior settings. The path planning feasibility will be taken care of by the adapter internally.

FleetUpdateHandle &**accept_delivery_requests**(*AcceptDeliveryRequest check*)

Provide a callback that indicates whether this fleet will accept a delivery request. By default all delivery requests will be rejected.

Note The callback function that you give should ideally be non-blocking and return quickly. It's meant to check whether this fleet's vehicles are compatible with the requested payload, pickup, and dropoff behavior settings. The path planning feasibility will be taken care of by the adapter internally.

FleetUpdateHandle &**default_maximum_delay**(std::optional<rmf_traffic::Duration> *value*)

Specify the default value for how high the delay of the current itinerary can become before it gets interrupted and replanned. A nullopt value will allow for an arbitrarily long delay to build up without being interrupted.

std::optional<rmf_traffic::Duration> **default_maximum_delay**() **const**
Get the default value for the maximum acceptable delay.

FleetUpdateHandle &**fleet_state_publish_period**(std::optional<rmf_traffic::Duration> *value*)
The behavior is identical to fleet_state_topic_publish_period.

FleetUpdateHandle &**fleet_state_topic_publish_period**(std::optional<rmf_traffic::Duration> *value*)
Specify a period for how often the fleet state message is published for this fleet. Passing in std::nullopt will disable the fleet state message publishing. The default value is 1s.

FleetUpdateHandle &**fleet_state_update_period**(std::optional<rmf_traffic::Duration> *value*)
Specify a period for how often the fleet state is updated in the database and to the API server. This is separate from publishing the fleet state over the ROS2 fleet state topic. Passing in std::nullopt will disable the updating, but this is not recommended unless you intend to provide the API server with the fleet states through some other means.

The default value is 1s.

FleetUpdateHandle(*FleetUpdateHandle*&&) = delete

FleetUpdateHandle &**operator=**(*FleetUpdateHandle*&&) = delete

class Confirmation
Confirmation is a class used by the task acceptance callbacks to decide if a task description should be accepted.

Public Functions

Confirmation()
Constructor.

Confirmation &**accept**()
Call this function to decide that you want to accept the task request. If this function is never called, it will be assumed that the task is rejected.

bool **is_accepted**() **const**
Check whether.

Confirmation &**errors**(std::vector<std::string> *error_messages*)
Call this function to bring attention to errors related to the task request. Each call to this function will overwrite any past calls, so it is recommended to only call it once.

Confirmation &**add_errors**(std::vector<std::string> *error_messages*)
Call this function to add errors instead of overwriting the ones that were already there.

const std::vector<std::string> &**errors**() **const**
Check the errors that have been given to this confirmation.

Class FleetUpdateHandle::Confirmation

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_FleetUpdateHandle.hpp

Nested Relationships

This class is a nested type of *Class FleetUpdateHandle*.

Class Documentation

class `rmf_fleet_adapter::agv::`*`FleetUpdateHandle`*`::`**`Confirmation`**

Confirmation is a class used by the task acceptance callbacks to decide if a task description should be accepted.

Public Functions

`Confirmation``()`

Constructor.

Confirmation &**`accept`**`()`

Call this function to decide that you want to accept the task request. If this function is never called, it will be assumed that the task is rejected.

bool **`is_accepted`**`()` **`const`**

Check whether.

Confirmation &**`errors`**`(`std::vector<std::string> *error_messages*`)`

Call this function to bring attention to errors related to the task request. Each call to this function will overwrite any past calls, so it is recommended to only call it once.

Confirmation &**`add_errors`**`(`std::vector<std::string> *error_messages*`)`

Call this function to add errors instead of overwriting the ones that were already there.

`const` std::vector<std::string> &**`errors`**`()` **`const`**

Check the errors that have been given to this confirmation.

Class RobotCommandHandle

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_RobotCommandHandle.hpp

Inheritance Relationships

Derived Type

- `public PyRobotCommandHandle` (*Class PyRobotCommandHandle*)

Class Documentation

class rmf_fleet_adapter::agv::**RobotCommandHandle**

Implement this class to receive robot commands from RMF.

Subclassed by *PyRobotCommandHandle*

Public Types

using Duration = rmf_traffic::Duration

using ArrivalEstimator = std::function<void(std::size_t path_index, *Duration* remaining_time)>

Use this callback function to update the fleet adapter on how long the robot will take to reach its next destination.

Parameters

- `[in] path_index`: The index of the path element that the robot is currently heading towards.
- `[in] remaining_time`: An estimate of how much longer the robot will take to arrive at `path_index`.

using RequestCompleted = std::function<void()>

Trigger this callback function when the follow_new_path request has been completed. It should only be triggered that one time and then discarded.

Public Functions

virtual void **follow_new_path**(**const** std::vector<rmf_traffic::agv::Plan::Waypoint> &*waypoints*, *ArrivalEstimator* *next_arrival_estimator*, *RequestCompleted* *path_finished_callback*) = 0

Have the robot follow a new path. If it was already following a path, then it should immediately switch over to this one.

Parameters

- `[in] waypoints`: The sequence of waypoints to follow. When the robot arrives at a waypoint in this sequence, it should wait at that waypoint until the clock reaches the time() field of the waypoint. This is important because the waypoint timing is used to avoid traffic conflicts with other vehicles.
- `[in] next_arrival_estimator`: Use this callback to give estimates for how long the robot will take to reach the path element of the specified index. You should still be calling *RobotUpdateHandle::update_position()* even as you call this function.
- `[in] path_finished_callback`: Trigger this callback when the robot is done following the new path. You do not need to trigger waypoint_arrival_callback when triggering this one.

virtual void **stop**() = 0

Have the robot come to an immediate stop.

virtual void **dock**(**const** std::string &*dock_name*, *RequestCompleted* *docking_finished_callback*) = 0

Have the robot begin a pre-defined docking procedure. Implement this function as a no-op if your robots do not perform docking procedures.

Parameters

- `[in] dock_name`: The predefined name of the docking procedure to use.
- `[in] docking_finished_callback`: Trigger this callback when the docking is finished.

virtual ~RobotCommandHandle() = default

Class RobotUpdateHandle

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_RobotUpdateHandle.hpp

Nested Relationships

Nested Types

- *Class RobotUpdateHandle::ActionExecution*
- *Class RobotUpdateHandle::Interruption*
- *Class RobotUpdateHandle::IssueTicket*
- *Class RobotUpdateHandle::Unstable*

Class Documentation

class `rmf_fleet_adapter::agv::`**RobotUpdateHandle**

You will be given an instance of this class every time you add a new robot to your fleet. Use that instance to send updates to RoMi-H about your robot's state.

Public Types

enum Tier

Values:

enumerator Info

General status information, does not require special attention.

enumerator Warning

Something unusual that might require attention.

enumerator Error

A critical failure that requires immediate operator attention.

using ActionExecutor = std::function<void(**const** std::string &category, **const** nlohmann::json &description, *ActionExecution* execution)>

Signature for a callback to request the robot to perform an action

Parameters

- `[in] category`: A category of the action to be performed
- `[in] description`: A description of the action to be performed

- `[in] execution`: An *ActionExecution* object that will be provided to the user for updating the state of the action.

Public Functions

void **interrupted**()

void **replan**()
Tell the RMF schedule that the robot needs a new plan. A new plan will be generated, starting from the last position that was given by *update_position()*. It is best to call *update_position()* with the latest position of the robot before calling this function.

void **update_position**(std::size_t *waypoint*, double *orientation*)
Update the current position of the robot by specifying the waypoint that the robot is on and its orientation.

void **update_position**(**const** Eigen::Vector3d &*position*, **const** std::vector<std::size_t> &*lanes*)
Update the current position of the robot by specifying the x, y, yaw position of the robot and one or more lanes that the robot is occupying.

Warning At least one lane must be specified. If no lane information is available, then use the update_position(std::string, Eigen::Vector3d) signature of this function.

void **update_position**(**const** Eigen::Vector3d &*position*, std::size_t *target_waypoint*)
Update the current position of the robot by specifying the x, y, yaw position of the robot and the waypoint that it is moving towards.

This should be used if the robot has diverged significantly from its course but it is merging back onto a waypoint.

void **update_position**(**const** std::string &*map_name*, **const** Eigen::Vector3d &*position*, **const** double *max_merge_waypoint_distance* = 0.1, **const** double *max_merge_lane_distance* = 1.0, **const** double *min_lane_length* = 1e-8)
Update the current position of the robot by specifying the x, y, yaw position of the robot and what map the robot is on.

We will attempt to merge the robot back onto the navigation graph. The parameters for this function are passed along to rmf_traffic::agv::compute_plan_starts().

Warning This function should only be used if the robot has diverged from the navigation graph somehow.

RobotUpdateHandle &**set_charger_waypoint**(**const** std::size_t *charger_wp*)
Set the waypoint where the charger for this robot is located. If not specified, the nearest waypoint in the graph with the is_charger() property will be assumed as the charger for this robot.

void **update_battery_soc**(**const** double *battery_soc*)
Update the current battery level of the robot by specifying its state of charge as a fraction of its total charge capacity

RobotUpdateHandle &**maximum_delay**(rmf_utils::optional<rmf_traffic::Duration> *value*)
Specify how high the delay of the current itinerary can become before it gets interrupted and replanned. A nullopt value will allow for an arbitrarily long delay to build up without being interrupted.

rmf_utils::optional<rmf_traffic::Duration> **maximum_delay**() **const**
Get the value for the maximum delay.

Note The setter for the maximum_delay field is run asynchronously, so it may take some time for the return value of this getter to match the value that was given to the setter.

void **set_action_executor**(*ActionExecutor action_executor*)
Set the ActionExecutor for this robot.

void **submit_direct_request**(nlohmann::json *task_request*, std::string *request_id*, std::function<void) nlohmann::json response
> *receive_response*Submit a direct task request to this manager

Parameters

- `[in] task_request`: A JSON description of the task request. It should match the task_request.json schema of rmf_api_msgs, in particular it must contain `category` and `description` properties.
- `[in] request_id`: The unique ID for this task request.
- `[in] receive_response`: Provide a callback to receive the response. The response will be a robot_task_response.json message from rmf_api_msgs (note: this message is not validated before being returned).

Interruption **interrupt**(std::vector<std::string> *labels*, std::function<void)
> *robot_is_interrupted*Interrupt (pause) the current task, yielding control of the robot away from the fleet adapter's task manager.

Return a handle for this interruption.

Parameters

- `[in] labels`: Labels that will be assigned to this interruption. It is recommended to include information about why the interruption is happening.

IssueTicket **create_issue**(*Tier tier*, std::string *category*, nlohmann::json *detail*)
Create a new issue for the robot.

Return A ticket for this issue

Parameters

- `[in] tier`: The severity of the issue
- `[in] category`: A brief category to describe the issue
- `[in] detail`: Full details of the issue that might be relevant to an operator or logging system.

void **log_info**(std::string *text*)
Add a log entry with Info severity.

void **log_warning**(std::string *text*)
Add a log entry with Warning severity.

void **log_error**(std::string *text*)
Add a log entry with Error severity.

Unstable &**unstable**()
Get a mutable reference to the unstable API extension.

const *Unstable* &**unstable**() **const**
Get a const reference to the unstable API extension.

class ActionExecution

The *ActionExecution* class should be used to manage the execution of and provide updates on ongoing actions.

Public Functions

void **update_remaining_time**(rmf_traffic::Duration *remaining_time_estimate*)

void **finished**()

bool **okay**() **const**

class Interruption

An object to maintain an interruption of the current task. When this object is destroyed, the task will resume.

Public Functions

void **resume**(std::vector<std::string> *labels*)

Call this function to resume the task while providing labels for resuming.

class IssueTicket

An object to maintain an issue that is happening with the robot. When this object is destroyed without calling *resolve()*, the issue will be "dropped", which issues a warning to the log.

Public Functions

void **resolve**(nlohmann::json *msg*)

Indicate that the issue has been resolved. The provided message will be logged for this robot and the issue will be removed from the robot state.

class Unstable

This API is experimental and will not be supported in the future. Users are to avoid relying on these feature for any integration.

Public Types

enum Decision

Values:

enumerator Undefined

enumerator Clear

enumerator Crowded

using Decide = std::function<void(*Decision*)>

A callback with this signature will be given to the watchdog when the robot is ready to enter a lift. If the watchdog passes in a true, then the robot will proceed to enter the lift. If the watchdog passes in a false, then the fleet adapter will release its session with the lift and resume later.

using Watchdog = std::function<void(**const** std::string&, *Decide*)>

Public Functions

rmf_traffic::schedule::Participant ***get_participant**()
Get the schedule participant of this robot.

void **set_lift_entry_watchdog**(*Watchdog* *watchdog*, rmf_traffic::Duration *wait_duration* = std::chrono::seconds(10))
Set a callback that can be used to check whether the robot is clear to enter the lift.

Class RobotUpdateHandle::ActionExecution

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_RobotUpdateHandle.hpp

Nested Relationships

This class is a nested type of *Class RobotUpdateHandle*.

Class Documentation

class rmf_fleet_adapter::agv::*RobotUpdateHandle*::**ActionExecution**
The *ActionExecution* class should be used to manage the execution of and provide updates on ongoing actions.

Public Functions

void **update_remaining_time**(rmf_traffic::Duration *remaining_time_estimate*)

void **finished**()

bool **okay**() **const**

Class RobotUpdateHandle::Interruption

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_RobotUpdateHandle.hpp

Nested Relationships

This class is a nested type of *Class RobotUpdateHandle*.

Class Documentation

class rmf_fleet_adapter::agv::*RobotUpdateHandle*::**Interruption**
An object to maintain an interruption of the current task. When this object is destroyed, the task will resume.

Public Functions

void **resume**(std::vector<std::string> *labels*)
Call this function to resume the task while providing labels for resuming.

Class RobotUpdateHandle::IssueTicket

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_RobotUpdateHandle.hpp

Nested Relationships

This class is a nested type of *Class RobotUpdateHandle*.

Class Documentation

class rmf_fleet_adapter::agv::*RobotUpdateHandle*::**IssueTicket**
An object to maintain an issue that is happening with the robot. When this object is destroyed without calling *resolve()*, the issue will be "dropped", which issues a warning to the log.

Public Functions

void **resolve**(nlohmann::json *msg*)
Indicate that the issue has been resolved. The provided message will be logged for this robot and the issue will be removed from the robot state.

Class RobotUpdateHandle::Unstable

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_RobotUpdateHandle.hpp

Nested Relationships

This class is a nested type of *Class RobotUpdateHandle*.

Class Documentation

class rmf_fleet_adapter::agv::*RobotUpdateHandle*::**Unstable**
This API is experimental and will not be supported in the future. Users are to avoid relying on these feature for any integration.

Public Types

enum Decision
Values:

enumerator Undefined

enumerator Clear

enumerator Crowded

using Decide = std::function<void (*Decision*) >
A callback with this signature will be given to the watchdog when the robot is ready to enter a lift. If the watchdog passes in a true, then the robot will proceed to enter the lift. If the watchdog passes in a false, then the fleet adapter will release its session with the lift and resume later.

using Watchdog = std::function<void (**const** std::string&, *Decide*) >

Public Functions

rmf_traffic::schedule::Participant ***get_participant** ()
Get the schedule participant of this robot.

void **set_lift_entry_watchdog** (*Watchdog watchdog*, rmf_traffic::Duration *wait_duration* = std::chrono::seconds(10))
Set a callback that can be used to check whether the robot is clear to enter the lift.

Class MockAdapter

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_test_MockAdapter.hpp

Inheritance Relationships

Base Type

- `public std::enable_shared_from_this< MockAdapter >`

Class Documentation

class `rmf_fleet_adapter::agv::test::`**MockAdapter** : **public** std::enable_shared_from_this<*MockAdapter*>
This class is an alternative to the *Adapter* class, but made specifically for testing. It does not try to connect to a Schedule Node or to any Negotiation topics. It keeps its database internal.

Public Functions

MockAdapter(**const** std::string &*node_name*, **const** rclcpp::NodeOptions &*node_options* = rclcpp::NodeOptions())
Create a mock adapter.

std::shared_ptr<*FleetUpdateHandle*> **add_fleet**(**const** std::string &*fleet_name*, rmf_traffic::agv::VehicleTraits *traits*, rmf_traffic::agv::Graph *navigation_graph*, std::optional<std::string> *server_uri* = std::nullopt)
Add a fleet to test.

std::shared_ptr<rclcpp::Node> **node**()
Get the rclcpp Node for this adapter.

std::shared_ptr<**const** rclcpp::Node> **node**() **const**
const-qualified *node()*

void **add_secondary_node**(std::shared_ptr<rclcpp::Node> *secondary_node*)

void **start**()
Start spinning this adapter.

void **stop**()
Stop this adapter from spinning.

void **dispatch_task**(std::string *task_id*, **const** nlohmann::json &*request*)
Submit a task request.

~MockAdapter()

Class Waypoint

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_Waypoint.hpp

Class Documentation

class rmf_fleet_adapter::agv::**Waypoint**

Public Functions

Waypoint(std::string *map_name*, Eigen::Vector3d *position*, rmf_traffic::Duration *mandatory_delay* = std::chrono::nanoseconds(0), bool *yield* = true)
Constructor

Parameters

- `[in] map_name`: The name of the reference map that the position is on.
- `[in] position`: A position along the robot's path where it will (or can) have zero instantaneous velocity. This will usually be a point where the robot needs to turn, or a point that comes before an intersection, so the robot can come to a stop and allow other vehicles to pass. This is a 2D heterogeneous vector. The first two elements refer to translational (x,y) position while the third element is a yaw value in radians.

- `[in] mandatory_delay`: A delay that the robot is expected to experience once it arrives at this waypoint. Usually this will be caused by the robots needing to wait for doors to open or lifts to arrive.
- `[in] yield`: Whether or not the robot can wait at this point. If yield is false, then the planner will assume that it cannot ask the robot to wait here. If yield is true, then the planner may request that the robot wait here to avoid conflicts with other traffic participants.

const std::string &**map_name**() **const**
Get the map of this *Waypoint*.

Waypoint &**map_name**(std::string *new_map_name*)
Set the map of this *Waypoint*.

const Eigen::Vector3d &**position**() **const**
Get the position of this *Waypoint*.

Waypoint &**position**(Eigen::Vector3d *new_position*)
Set the position of this *Waypoint*.

rmf_traffic::Duration **mandatory_delay**() **const**
Get the mandatory delay associated with this *Waypoint*.

Waypoint &**mandatory_delay**(rmf_traffic::Duration *duration*)
Set the mandatory delay associated with this *Waypoint*.

bool **yield**() **const**
Get whether the robot can yield here.

Waypoint &**yield**(bool *on*)
Set whether the robot can yield here.

Class AsyncBidder

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_bidding_AsyncBidder.hpp

Class Documentation

class `rmf_task_ros2::bidding::`**AsyncBidder**

Public Types

using Respond = std::function<void(**const** *Response*&)>

using ReceiveNotice = std::function<void(**const** *BidNoticeMsg* &notice, *Respond* respond)>
Callback function when a bid notice is received from the autioneer

Return submission Estimates of a task. This submission is used by dispatcher for eval

Parameters

- `[in] notice`: bid notice msg

Public Static Functions

static std::shared_ptr<*AsyncBidder*> **make**(**const** std::shared_ptr<rclcpp::Node> &*node*, *ReceiveNotice* *notice_cb*)

Create a bidder to bid for incoming task requests from Task *Dispatcher*

Parameters

- `[in] node`: ROS 2 node instance
- `[in] fleet_name`: Name of the bidder
- `[in] valid_task_types`: A list of valid tasks types which are supported by the bidder
- `[in] submission_cb`: fn which is used to provide a bid submission during a call for bid

Class Auctioneer

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_bidding_Auctioneer.hpp

Nested Relationships

Nested Types

- *Class Auctioneer::Evaluator*

Inheritance Relationships

Base Type

- `public std::enable_shared_from_this< Auctioneer >`

Class Documentation

class `rmf_task_ros2::bidding::`**Auctioneer** : **public** std::enable_shared_from_this<*Auctioneer*>

The *Auctioneer* class is responsible for initiating and mediating the bidding process during task dispatching. Hence, this class instance is solely used within the *Dispatcher* class

Public Types

using **BiddingResultCallback** = std::function<void(**const** std::string &task_id, **const** std::optional<*Response*::*Proposal*> winner, **const** std::vector<std::string> &errors)>

Callback which will provide the winner when a bid is concluded

Parameters

- `[in] task_id`: bidding task id

- `[in] winner`: single winner from all submissions. nullopt if non

using ConstEvaluatorPtr = std::shared_ptr<**const** *Evaluator*>

Public Functions

void **request_bid**(**const** *BidNoticeMsg* &*bid_notice*)

Start a bidding process by provide a bidding task. Note the each bidding process is conducted sequentially

Parameters

- `[in] bid_notice`: bidding task, task which will call for bid

void **ready_for_next_bid**()

Call this to tell the auctioneer that it may begin to perform the next bid.

void **set_evaluator**(*ConstEvaluatorPtr evaluator*)

Provide a custom evaluator which will be used to choose the best bid If no selection is given, Default is: *LeastFleetDiffCostEvaluator*

Parameters

- `[in] evaluator`:

Public Static Functions

static std::shared_ptr<*Auctioneer*> **make**(**const** std::shared_ptr<rclcpp::Node> &*node*, *BiddingResultCallback result_callback*, *ConstEvaluatorPtr evaluator*)

Create an instance of the *Auctioneer*. This instance will handle all the task dispatching bidding mechanism. A default evaluator is used.

See *make()*

Parameters

- `[in] node`: ros2 node which will manage the bidding
- `[in] result_callback`: This callback fn will be called when a bidding result is concluded

class Evaluator

A pure abstract interface class for the auctioneer to choose the best choosing the best submissions.

Subclassed by *rmf_task_ros2::bidding::LeastFleetCostEvaluator*, *rmf_task_ros2::bidding::LeastFleetDiffCostEvaluator*, *rmf_task_ros2::bidding::QuickestFinishEvaluator*

Public Functions

virtual std::optional<std::size_t> **choose**(**const** *Responses* &*responses*) **const** = 0
Given a list of submissions, choose the one that is the "best". It is up to the implementation of the *Evaluator* to decide how to rank.

virtual **~Evaluator**() = default

Class Auctioneer::Evaluator

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_bidding_Auctioneer.hpp

Nested Relationships

This class is a nested type of *Class Auctioneer*.

Inheritance Relationships

Derived Types

- `public rmf_task_ros2::bidding::LeastFleetCostEvaluator` (*Class LeastFleetCostEvaluator*)
- `public rmf_task_ros2::bidding::LeastFleetDiffCostEvaluator` (*Class LeastFleetDiffCostEvaluator*)
- `public rmf_task_ros2::bidding::QuickestFinishEvaluator` (*Class QuickestFinishEvaluator*)

Class Documentation

class `rmf_task_ros2::bidding::Auctioneer::`**Evaluator**
A pure abstract interface class for the auctioneer to choose the best choosing the best submissions.

Subclassed by *rmf_task_ros2::bidding::LeastFleetCostEvaluator*, *rmf_task_ros2::bidding::LeastFleetDiffCostEvaluator*, *rmf_task_ros2::bidding::QuickestFinishEvaluator*

Public Functions

virtual std::optional<std::size_t> **choose**(**const** *Responses* &*responses*) **const** = 0
Given a list of submissions, choose the one that is the "best". It is up to the implementation of the *Evaluator* to decide how to rank.

virtual **~Evaluator**() = default

Class LeastFleetCostEvaluator

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_bidding_Auctioneer.hpp

Inheritance Relationships

Base Type

- `public rmf_task_ros2::bidding::Auctioneer::Evaluator` (*Class Auctioneer::Evaluator*)

Class Documentation

class `rmf_task_ros2::bidding::`**LeastFleetCostEvaluator** : **public** rmf_task_ros2::bidding::*Auctioneer*::*Evalua*

Public Functions

virtual std::optional<std::size_t> **choose**(**const** *Responses* &*submissions*) **const final**
Given a list of submissions, choose the one that is the "best". It is up to the implementation of the Evaluator to decide how to rank.

Class LeastFleetDiffCostEvaluator

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_bidding_Auctioneer.hpp

Inheritance Relationships

Base Type

- `public rmf_task_ros2::bidding::Auctioneer::Evaluator` (*Class Auctioneer::Evaluator*)

Class Documentation

class `rmf_task_ros2::bidding::`**LeastFleetDiffCostEvaluator** : **public** rmf_task_ros2::bidding::*Auctioneer*::*E*

Public Functions

virtual std::optional<std::size_t> **choose**(**const** *Responses* &*responses*) **const final**
Given a list of submissions, choose the one that is the "best". It is up to the implementation of the Evaluator to decide how to rank.

Class QuickestFinishEvaluator

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_bidding_Auctioneer.hpp

Inheritance Relationships

Base Type

- `public rmf_task_ros2::bidding::Auctioneer::Evaluator` (*Class Auctioneer::Evaluator*)

Class Documentation

class `rmf_task_ros2::bidding::`**QuickestFinishEvaluator** : **public** rmf_task_ros2::bidding::*Auctioneer*::*Evalua*

Public Functions

virtual std::optional<std::size_t> **choose**(**const** *Responses* &*submissions*) **const final**
Given a list of submissions, choose the one that is the "best". It is up to the implementation of the Evaluator to decide how to rank.

Class Dispatcher

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_Dispatcher.hpp

Inheritance Relationships

Base Type

- `public std::enable_shared_from_this< Dispatcher >`

Class Documentation

class `rmf_task_ros2::`**Dispatcher** : **public** std::enable_shared_from_this<*Dispatcher*>
This dispatcher class holds an instance which handles the dispatching of tasks to all downstream RMF fleet adapters.

Public Types

using DispatchStates = std::unordered_map<*TaskID*, *DispatchStatePtr*>

using DispatchStateCallback = std::function<void(**const** *DispatchState* &status)>

Public Functions

std::optional<*TaskID*> **submit_task**(**const** rmf_task_msgs::msg::TaskDescription *&task_description*)

Submit task to dispatcher node. Calling this function will immediately trigger the bidding process, then the task "action". Once submmitted, Task State will be in 'Pending' State, till the task is awarded to a fleet then the state will turn to 'Queued'

Return task_id self-generated task_id, nullopt is submit task failed

Parameters

- `[in] task_description`: Submit a task to dispatch

bool **cancel_task**(**const** *TaskID* *&task_id*)

Cancel an active task which was previously submitted to *Dispatcher*. This will terminate the task with a State of: `Canceled`. If a task is `Queued` or `Executing`, this function will send a cancel req to the respective fleet adapter. It is the responsibility of the fleet adapter to make sure it cancels the task internally.

Return true if success

Parameters

- `[in] task_id`: Task to cancel

std::optional<*DispatchState*> **get_dispatch_state**(**const** *TaskID* *&task_id*) **const**

Check the state of a submited task. It can be either active or terminated

Return State of the task, nullopt if task is not available

Parameters

- `[in] task_id`: task_id obtained from *`submit_task()`*

const *DispatchStates* &**active_dispatches**() **const**

Get a mutable ref of active tasks map list handled by dispatcher.

const *DispatchStates* &**finished_dispatches**() **const**

Get a mutable ref of terminated tasks map list.

void **on_change**(*DispatchStateCallback on_change_fn*)

Trigger this callback when a task status is changed. This will return the Changed task status.

Parameters

- `[in] callback`: function

void **evaluator**(bidding::*Auctioneer*::*ConstEvaluatorPtr evaluator*)

Change the default evaluator to a custom evaluator, which is used by bidding auctioneer. Default evaluator is: `LeastFleetDiffCostEvaluator`

Parameters

- `[in] evaluator`: evaluator used to select the best bid from fleets

std::shared_ptr<rclcpp::Node> **node**()

Get the rclcpp::Node that this dispatcher will be using for communication.

void **spin**()
spin dispatcher node

Public Static Functions

static std::shared_ptr<*Dispatcher*> **init_and_make_node**(**const** std::string *dispatcher_node_name*)
Initialize an rclcpp context and make an dispatcher instance. This will instantiate an rclcpp::Node, a task dispatcher node. *Dispatcher* node will allow you to dispatch submitted task to the best fleet/robot within RMF.

See *init_and_make_node()*

Parameters

- `[in] dispatcher_node_name`: The ROS 2 node to manage the Dispatching of Task

static std::shared_ptr<*Dispatcher*> **make_node**(**const** std::string *dispatcher_node_name*)
Similarly this will init the dispatcher, but you will also need to init rclcpp via rclcpp::init(~).

See *make_node()*

Parameters

- `[in] dispatcher_node_name`: The ROS 2 node to manage the Dispatching of Task

static std::shared_ptr<*Dispatcher*> **make**(**const** std::shared_ptr<rclcpp::Node> &*node*)
Create a dispatcher by providing the ros2 node

See *make()*

Parameters

- `[in] node`: ROS 2 node instance

Class Writer

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_blockade_Writer.hpp

Inheritance Relationships

Base Type

- `public std::enable_shared_from_this< Writer >`

Class Documentation

class rmf_traffic_ros2::blockade::**Writer** : **public** std::enable_shared_from_this<*Writer*>

The *Writer* class provides an API that allows a Node to create blockade Participants.

Public Types

using ReservedRange = rmf_traffic::blockade::ReservedRange

using ReservationId = rmf_traffic::blockade::ReservationId

using NewRangeCallback = std::function<void (**const** *ReservationId* reservation, **const** *ReservedRange* &range) >

Public Functions

rmf_traffic::blockade::Participant **make_participant** (rmf_traffic::blockade::ParticipantId *id*, double *radius*, *NewRangeCallback* *new_range_cb*)

Make a blockade participant.

Return the API for updating the blockade Participant.

Parameters

- [in] id: The ID of the participant that is being created. This must match the schedule ParticipantId. All blockade participants must also be schedule participants to comply with the RMF traffic protocol.
- [in] radius: The radius around the path that the participant will occupy.
- [in] new_range_cb: This callback will get triggered when a new range arrives.

Public Static Functions

static std::shared_ptr<*Writer*> **make** (rclcpp::Node &*node*)

Create an instance of a writer. The writer and all Participants it creates depend on the life of the rclcpp::Node. It's best to keep all of these as members of the Node.

Parameters

- [in] node: The node that will manage the subscriptions of this writer.

Class ConvexShapeContext

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_geometry_ConvexShape.hpp

Class Documentation

class rmf_traffic_ros2::geometry::**ConvexShapeContext**

Public Functions

ConvexShapeContext()

rmf_traffic_msgs::msg::ConvexShape **insert**(rmf_traffic::geometry::ConstFinalConvexShapePtr *shape*)

rmf_traffic::geometry::ConstFinalConvexShapePtr **at**(**const** rmf_traffic_msgs::msg::ConvexShape &*shape*) **const**

Class ShapeContext

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_geometry_Shape.hpp

Class Documentation

class rmf_traffic_ros2::geometry::**ShapeContext**

Public Functions

ShapeContext()

rmf_traffic_msgs::msg::Shape **insert**(rmf_traffic::geometry::ConstFinalShapePtr *shape*)

rmf_traffic::geometry::ConstFinalShapePtr **at**(**const** rmf_traffic_msgs::msg::Shape &*shape*) **const**

Class AbstractParticipantLogger

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_ParticipantRegistry.hpp

Inheritance Relationships

Derived Type

- public rmf_traffic_ros2::schedule::YamlLogger (*Class YamlLogger*)

Class Documentation

class rmf_traffic_ros2::schedule::**AbstractParticipantLogger**

This is the base class for the persistence logger.

Subclassed by *rmf_traffic_ros2::schedule::YamlLogger*

Public Functions

virtual void **write_operation**(*AtomicOperation operation*) = 0
Called when we wish to commit an operation to disk

Parameters

- `[in] operation`:

virtual std::optional<*AtomicOperation*> **read_next_record**() = 0
Called when we wish to read the next record up during initiallization.

Return a std::nullopt when we have exhausted all records.

virtual **~AbstractParticipantLogger**() = default

Class MirrorManager

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_MirrorManager.hpp

Nested Relationships

Nested Types

- *Class MirrorManager::Options*

Class Documentation

class `rmf_traffic_ros2::schedule::`**MirrorManager**

Public Functions

std::shared_ptr<**const** rmf_traffic::schedule::Mirror> **view**() **const**
Get an immutable view of the mirror.

void **update**()
Attempt to update this mirror immediately.

const *Options* &**get_options**() **const**
Get the options for this mirror manager.

MirrorManager &**set_options**(*Options options*)
Set the options for this mirror manager.

rmf_traffic::schedule::Database **fork**() **const**
Fork a new database off of the managed Mirror. The state of the new database will match the last state of the upstream database that this Mirror knows about.

class Options
Options for the *MirrorManager*.

Public Functions

Options(std::mutex *_update_mutex_ = nullptr, bool _update_on_wakeup_ = true)
update_mutex A reference to a mutex that should be locked when performing an update. When set to a nullptr, no mutex will be locked.

Constructor

update_on_wakeup Specify if the mirror should perform an update whenever it gets woken up by the schedule.

std::mutex ***update_mutex**() **const**
Get a reference to the mutex that will be used when performing an update.

Options &**update_mutex**(std::mutex *_mutex_)
Set the mutex that should be used when performing an update.

bool **update_on_wakeup**() **const**
True if the mirror should be updated each time a MirrorWakeup message is received. The MirrorWakeup messages are sent out each time a change is introduced to the schedule database.

Options &**update_on_wakeup**(bool _choice_)
Toggle the choice to wakeup on an update.

Class MirrorManager::Options

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_MirrorManager.hpp

Nested Relationships

This class is a nested type of *Class MirrorManager*.

Class Documentation

class rmf_traffic_ros2::schedule::*MirrorManager*::**Options**
Options for the *MirrorManager*.

Public Functions

Options(std::mutex *_update_mutex_ = nullptr, bool _update_on_wakeup_ = true)
update_mutex A reference to a mutex that should be locked when performing an update. When set to a nullptr, no mutex will be locked.

Constructor

update_on_wakeup Specify if the mirror should perform an update whenever it gets woken up by the schedule.

std::mutex ***update_mutex**() **const**
Get a reference to the mutex that will be used when performing an update.

Options &**update_mutex**(std::mutex *_mutex_)
Set the mutex that should be used when performing an update.

bool **update_on_wakeup**() **const**
: True if the mirror should be updated each time a MirrorWakeup message is received. The MirrorWakeup messages are sent out each time a change is introduced to the schedule database.

Options &**update_on_wakeup**(bool *choice*)
: Toggle the choice to wakeup on an update.

Class MirrorManagerFuture

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_MirrorManager.hpp

Class Documentation

class rmf_traffic_ros2::schedule::**MirrorManagerFuture**

Public Functions

void **wait**() **const**
: Wait for the *MirrorManager* to finish initializing. This will block until initialization is finished.

Note The initialization requires some ROS2 service calls, so there needs to be a separate thread running rclcpp::spin(~) while this function blocks. Otherwise it will block forever.

std::future_status **wait_for**(**const** rmf_traffic::Duration &*timeout*) **const**
: Wait for the *MirrorManager* to finish initializing. This will block until initialization is finished or until the timeout duration has passed.

See *wait()*

std::future_status **wait_until**(**const** rmf_traffic::Time &*time*) **const**
: Wait for the *MirrorManager* to finish initializing. This will block until initialization is finished or until the time has been reached.

See *wait()*

bool **valid**() **const**
: Check if this *MirrorManagerFuture* is still valid. This means that *get()* has never been called.

MirrorManager **get**()
: Get the *MirrorManager* of this future. This will wait until the *MirrorManager* is initialized before returning it.

Class Negotiation

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Negotiation.hpp

Nested Relationships

Nested Types

- *Class Negotiation::Worker*

Class Documentation

class rmf_traffic_ros2::schedule::**Negotiation**

A ROS2 interface for negotiating solutions to schedule conflicts.

Public Types

using TableViewPtr = rmf_traffic::schedule::Negotiation::Table::ViewerPtr

using ResponderPtr = rmf_traffic::schedule::Negotiator::ResponderPtr

using StatusUpdateCallback = std::function<void(uint64_t conflict_version, *TableViewPtr* table_view)>

using StatusConclusionCallback = std::function<void(uint64_t conflict_version, bool success)>

Public Functions

Negotiation(rclcpp::Node &*node*, std::shared_ptr<**const** rmf_traffic::schedule::Snappable> *viewer*, std::shared_ptr<*Worker*> *worker* = nullptr)

Constructor

Parameters

- `[in] worker`: If a worker is provided, the *Negotiation* will be performed asynchronously. If it is not provided, then the Negotiators must be single-threaded, and their respond() functions must block until finished.

Negotiation &**timeout_duration**(rmf_traffic::Duration *duration*)

Set the timeout duration for negotiators. If a negotiator does not respond within this time limit, then the negotiation will automatically be forfeited. This is important to prevent negotiations from getting hung forever.

rmf_traffic::Duration **timeout_duration**() **const**

Get the current timeout duration setting.

void **on_status_update**(*StatusUpdateCallback* *cb*)

Register a callback with this *Negotiation* manager that triggers on negotiation status updates.

Parameters

- `[in] cb`: The callback function to be called upon status updates.

void **on_conclusion**(*StatusConclusionCallback cb*)
Register a callback with this *Negotiation* manager that triggers on negotiation status conclusions.

Parameters

- `[in] cb`: The callback function to be called upon status conclusions.

TableViewPtr **table_view**(uint64_t *conflict_version*, **const** std::vector<rmf_traffic::schedule::ParticipantId> &*sequence*) **const**
Get a Negotiation::TableView that provides a view into what participants are proposing.

This function does not care about table versioning.

Return A TableView into what participants are proposing.

Parameters

- `[in] conflict_version`: The conflict version of the negotiation
- `[in] sequence`: The sequence of participant ids. Follows convention of other sequences (ie. The last ParticipantId is the owner of the table)

void **set_retained_history_count**(uint *count*)
Set the number of negotiations to retain.

Parameters

- `[in] count`: The number of negotiations to retain

std::shared_ptr<void> **register_negotiator**(rmf_traffic::schedule::ParticipantId *for_participant*, std::unique_ptr<rmf_traffic::schedule::Negotiator> *negotiator*)
Register a negotiator with this *Negotiation* manager.

Return a handle that should be kept by the caller. When this handle expires, this negotiator will be automatically unregistered.

Parameters

- `[in] for_participant`: The ID of the participant that this negotiator will work for
- `[in] negotiator`: The negotiator interface to use for this participant

std::shared_ptr<void> **register_negotiator**(rmf_traffic::schedule::ParticipantId *for_participant*, std::unique_ptr<rmf_traffic::schedule::Negotiator> *negotiator*, std::function<void) > *on_negotiation_failure*Register a negotiator with this *Negotiation* manager.

Return a handle that should be kept by the caller. When this handle expires, this negotiator will be automatically unregistered.

Parameters

- `[in] for_participant`: The ID of the participant that this negotiator will work for
- `[in] negotiator`: The negotiator interface to use for this participant
- `[in] on_negotiation_failure`: A callback that will be triggered if a negotiation for this participant fails

std::shared_ptr<void> **register_negotiator**(rmf_traffic::schedule::ParticipantId *for_participant*, std::function<void(*TableViewPtr* view, *ResponderPtr* responder)> *respond*, std::function<void> *on_negotiation_failure* = nullptrRegister a negotiator with this *Negotiation* manager using a lambda.

Return a handle that should be kept by the caller. When this handle expires, this negotiator will be automatically unregistered.

Parameters

- `[in] for_participant`: The ID of the participant that this negotiator will work for
- `[in] respond`: The callback that will be used as the negotiator's response
- `[in] on_negotiation_failure`: A callback that will be triggered if a negotiation for this participant fails

class Worker

The *Worker* class can be used to make the *Negotiation* asynchronous.

Public Functions

virtual void **schedule**(std::function<void()> *job* = 0Tell the worker to add a callback to its schedule. It is imperative that this function is thread-safe.

virtual ~Worker() = default

Class Negotiation::Worker

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Negotiation.hpp

Nested Relationships

This class is a nested type of *Class Negotiation*.

Class Documentation

class rmf_traffic_ros2::schedule::*Negotiation*::**Worker**

The *Worker* class can be used to make the *Negotiation* asynchronous.

Public Functions

virtual void **schedule**(std::function<void()> *job* = 0Tell the worker to add a callback to its schedule. It is imperative that this function is thread-safe.

virtual ~Worker() = default

Class ParticipantRegistry

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_ParticipantRegistry.hpp

Class Documentation

class `rmf_traffic_ros2::schedule::ParticipantRegistry`

Adds a persistance layer to the participant ids. This allows the scheduler to restart without the need to restart fleet adapters. Internally, this class implements a an append only journal. This makes it independent of any id generation inside the database, as long as the said database id generation algorithm is deterministic.

Public Types

using Registration = rmf_traffic::schedule::Writer::Registration

Public Functions

ParticipantRegistry(std::unique_ptr<*AbstractParticipantLogger*> *logger*, std::shared_ptr<*Database*> *database*)

Constructor

Parameters

- `[in] logger`: The logging implementation to use for recording registration.
- `[in] database`: The database that will register the participants.

Registration **add_or_retrieve_participant**(*ParticipantDescription* *description*)

Adds a participant or retrieves its ID if it was already added in the past

Return ParticipantId of the participant

Parameters

- `[in] description`: The description of the participant that one wishes to register.

Class Writer

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Writer.hpp

Inheritance Relationships

Base Type

- `public std::enable_shared_from_this< Writer >`

Class Documentation

class rmf_traffic_ros2::schedule::**Writer** : **public** std::enable_shared_from_this<*Writer*>

The *Writer* class provides an API that allows a Node to create schedule Participants.

Public Functions

bool **ready**() **const**

Returns true if all the services needed by this writer are ready.

void **wait_for_service**() **const**

Wait for the necessary services to be available.

bool **wait_for_service**(rmf_traffic::Time *stop*) **const**

Wait for the necessary services to be available, or for the time point to be reached, whichever happens first.

Return true if the necessary services are now available, false otherwise.

Parameters

- [in] stop: The maximum time point that this will wait until

std::future<rmf_traffic::schedule::Participant> **make_participant**(rmf_traffic::schedule::ParticipantDescription *description*)

Begin creation of a schedule participant.

The node of this *Writer* needs to be spun in order for the Participant to finish being created.

Parameters

- [in] description.: The description of the participant.

void **async_make_participant**(rmf_traffic::schedule::ParticipantDescription *description*, std::function<void(rmf_traffic::schedule::Participant)> *ready_callback*)

Asynchronously create a schedule participant.

When the Participant is ready to be used, the ready_callback will be triggered with the newly created Participant instance.

Parameters

- [in] description: The description of the participant.
- [in] ready_callback: The callback that will be triggered when the participant is ready.

Public Static Functions

static std::shared_ptr<*Writer*> **make**(**const** std::shared_ptr<rclcpp::Node> &*node*)

Create an instance of a writer. The writer and all Participants it creates depend on the life of the rclcpp::Node. It's best to keep all of these as members of the Node.

Parameters

- [in] node: The node that will manage the subscriptions of this writer. This will be held as a std::weak_ptr<rclcpp::Node> so it is okay to store the writer inside the node itself.

Class YamlLogger

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_ParticipantRegistry.hpp

Inheritance Relationships

Base Type

- `public rmf_traffic_ros2::schedule::AbstractParticipantLogger` (*Class AbstractParticipantLogger*)

Class Documentation

class `rmf_traffic_ros2::schedule::YamlLogger` : **public** rmf_traffic_ros2::schedule::*AbstractParticipantLogger*

YAML logger class. Logs everything to YAML buffers on disk.

Public Functions

YamlLogger(std::string *filename*)

Constructor Loads and logs to the specified file.

Exceptions

- `YAML::ParserException`: if there is an error in the syntax of log file.
- `YAML::BadFile`: if there are problems with reading the file.
- `std::filesystem_error`: if there is no permission to create the directory.

virtual void **write_operation**(*AtomicOperation operation*) **override**

See *AbstractParticipantLogger*.

virtual std::optional<*AtomicOperation*> **read_next_record**() **override**

See *AbstractParticipantLogger*

Exceptions

- `std::runtime_error`: if there was an error in the logfile.

1.3.3 Functions

Function rmf_fleet_adapter::agv::parse_graph

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_parse_graph.hpp

Function Documentation

rmf_traffic::agv::Graph rmf_fleet_adapter::agv::**parse_graph**(**const** std::string &*filename*, **const** rmf_traffic::agv::VehicleTraits &*vehicle_traits*)

Parse the graph described by a yaml file.

Warning This will throw a std::runtime_error if the file has a syntax error.

Function rmf_task_ros2::bidding::convert(const Response::Proposal&)

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_bidding_Response.hpp

Function Documentation

BidProposalMsg rmf_task_ros2::bidding::**convert**(**const** *Response*::*Proposal* &*proposal*)

Function rmf_task_ros2::bidding::convert(const Response&, const std::string&)

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_bidding_Response.hpp

Function Documentation

BidResponseMsg rmf_task_ros2::bidding::**convert**(**const** *Response* &*response*, **const** std::string &*task_id*)

Function rmf_task_ros2::bidding::convert(const BidResponseMsg&)

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_bidding_Response.hpp

Function Documentation

Response rmf_task_ros2::bidding::**convert**(**const** *BidResponseMsg* &*msg*)

Function rmf_task_ros2::convert(const std::optional<DispatchState::Assignment>&)

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_DispatchState.hpp

Function Documentation

rmf_task_msgs::msg::Assignment rmf_task_ros2::**convert**(**const** std::optional<*DispatchState*::*Assignment*> &*assignment*)

Function rmf_task_ros2::convert(const DispatchState&)

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_DispatchState.hpp

Function Documentation

rmf_task_msgs::msg::DispatchState rmf_task_ros2::**convert**(**const** *DispatchState* &*state*)

Function rmf_traffic_ros2::blockade::make_node(const rclcpp::NodeOptions&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_blockade_Node.hpp

Function Documentation

std::shared_ptr<rclcpp::Node> rmf_traffic_ros2::blockade::**make_node**(**const** rclcpp::NodeOptions &*options* = rclcpp::NodeOptions())

Make a blockade node instance.

Function rmf_traffic_ros2::blockade::make_node(const std::string&, const rclcpp::NodeOptions&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_blockade_Node.hpp

Function Documentation

std::shared_ptr<rclcpp::Node> rmf_traffic_ros2::blockade::**make_node**(**const** std::string &*node_name*, **const** rclcpp::NodeOptions &*options* = rclcpp::NodeOptions())

Make a blockade node instance, specifying a node name.

Function rmf_traffic_ros2::convert(const rmf_site_map_msgs::msg::SiteMap&, int, double)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_agv_Graph.hpp

Function Documentation

rmf_traffic::agv::Graph rmf_traffic_ros2::**convert**(**const** rmf_site_map_msgs::msg::SiteMap &*from*, int *graph_idx* = 0, double *wp_tolerance* = 1e-3)

Function rmf_traffic_ros2::convert(const rmf_traffic::geometry::Circle&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_geometry_Circle.hpp

Function Documentation

rmf_traffic_msgs::msg::Circle rmf_traffic_ros2::**convert**(**const** rmf_traffic::geometry::Circle &*circle*)

Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::Circle&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_geometry_Circle.hpp

Function Documentation

rmf_traffic::geometry::Circle rmf_traffic_ros2::**convert**(**const** rmf_traffic_msgs::msg::Circle &*circle*)

Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ConvexShapeContext&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_geometry_ConvexShape.hpp

Function Documentation

geometry::*ConvexShapeContext* rmf_traffic_ros2::**convert**(**const** rmf_traffic_msgs::msg::ConvexShapeContext &*from*)

Function rmf_traffic_ros2::convert(const geometry::ConvexShapeContext&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_geometry_ConvexShape.hpp

Function Documentation

rmf_traffic_msgs::msg::ConvexShapeContext rmf_traffic_ros2::**convert**(**const** geometry::*ConvexShapeContext* &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ShapeContext&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_geometry_Shape.hpp

Function Documentation

geometry::*ShapeContext* `rmf_traffic_ros2::convert`(**const** rmf_traffic_msgs::msg::ShapeContext &*from*)

Function rmf_traffic_ros2::convert(const geometry::ShapeContext&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_geometry_Shape.hpp

Function Documentation

rmf_traffic_msgs::msg::ShapeContext `rmf_traffic_ros2::convert`(**const** geometry::*ShapeContext* &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::Profile&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_Profile.hpp

Function Documentation

rmf_traffic::Profile `rmf_traffic_ros2::convert`(**const** rmf_traffic_msgs::msg::Profile &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic::Profile&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_Profile.hpp

Function Documentation

rmf_traffic_msgs::msg::Profile `rmf_traffic_ros2::convert`(**const** rmf_traffic::Profile &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::Route&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_Route.hpp

Function Documentation

rmf_traffic::Route `rmf_traffic_ros2::convert`(**const** rmf_traffic_msgs::msg::Route &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic::Route&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_Route.hpp

Function Documentation

rmf_traffic_msgs::msg::Route `rmf_traffic_ros2::convert`(**const** rmf_traffic::Route &*from*)

Function rmf_traffic_ros2::convert(const std::vector<rmf_traffic_msgs::msg::Route>&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_Route.hpp

Function Documentation

std::vector<rmf_traffic::Route> `rmf_traffic_ros2::convert`(**const** std::vector<rmf_traffic_msgs::msg::Route> &*from*)

Function rmf_traffic_ros2::convert(const std::vector<rmf_traffic::Route>&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_Route.hpp

Function Documentation

std::vector<rmf_traffic_msgs::msg::Route> `rmf_traffic_ros2::convert`(**const** std::vector<rmf_traffic::Route> &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ScheduleChangeAddItem&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Change.hpp

Function Documentation

rmf_traffic::schedule::Change::Add::Item `rmf_traffic_ros2::convert`(**const** rmf_traffic_msgs::msg::ScheduleChangeAddItem &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Change::Add::Item&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Change.hpp

Function Documentation

rmf_traffic_msgs::msg::ScheduleChangeAddItem `rmf_traffic_ros2::convert`(**const** rmf_traffic::schedule::Change::Add::Item &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ScheduleChangeAdd&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Change.hpp

Function Documentation

rmf_traffic::schedule::Change::Add `rmf_traffic_ros2::convert`(**const** rmf_traffic_msgs::msg::ScheduleChangeAdd &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Change::Add&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Change.hpp

Function Documentation

rmf_traffic_msgs::msg::ScheduleChangeAdd `rmf_traffic_ros2::convert`(**const** rmf_traffic::schedule::Change::Add &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ScheduleChangeDelay&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Change.hpp

Function Documentation

rmf_traffic::schedule::Change::Delay `rmf_traffic_ros2::convert`(**const** rmf_traffic_msgs::msg::ScheduleChangeDelay &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Change::Delay&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Change.hpp

Function Documentation

rmf_traffic_msgs::msg::ScheduleChangeDelay `rmf_traffic_ros2::convert`(**const** rmf_traffic::schedule::Change::Delay &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Change::UnregisterParticipant&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Change.hpp

Function Documentation

rmf_traffic::schedule::ParticipantId `rmf_traffic_ros2::convert`(**const** rmf_traffic::schedule::Change::UnregisterParticipant &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ScheduleChangeCull&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Change.hpp

Function Documentation

rmf_traffic::schedule::Change::Cull `rmf_traffic_ros2::convert`(**const** rmf_traffic_msgs::msg::ScheduleChangeCull &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Change::Cull&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Change.hpp

Function Documentation

rmf_traffic_msgs::msg::ScheduleChangeCull `rmf_traffic_ros2::convert`(**const** rmf_traffic::schedule::Change::Cull &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Inconsistencies::Element&, const rmf_traffic::schedule::ProgressVersion)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Inconsistencies.hpp

Function Documentation

rmf_traffic_msgs::msg::ScheduleInconsistency `rmf_traffic_ros2::convert`(**const** rmf_traffic::schedule::Inconsistencies::Element *&from*, **const** rmf_traffic::schedule::ProgressVersion *progress_version*)

Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Itinerary&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Itinerary.hpp

Function Documentation

std::vector<rmf_traffic_msgs::msg::Route> `rmf_traffic_ros2::convert`(**const** rmf_traffic::schedule::Itinerary *&from*)

Function rmf_traffic_ros2::convert(const std::vector<rmf_traffic_msgs::msg::Itinerary>&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Itinerary.hpp

Function Documentation

std::vector<rmf_traffic::schedule::Itinerary> `rmf_traffic_ros2::convert`(**const** std::vector<rmf_traffic_msgs::msg::Itinerary> *&from*)

Function rmf_traffic_ros2::convert(const std::vector<rmf_traffic::schedule::Itinerary>&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Itinerary.hpp

Function Documentation

std::vector<rmf_traffic_msgs::msg::Itinerary> `rmf_traffic_ros2::convert`(**const** std::vector<rmf_traffic::schedule::Itinerary> *&from*)

Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ParticipantDescription&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_ParticipantDescription.hpp

Function Documentation

rmf_traffic::schedule::ParticipantDescription `rmf_traffic_ros2::convert`(**const** rmf_traffic_msgs::msg::ParticipantDescription &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::ParticipantDescription&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_ParticipantDescription.hpp

Function Documentation

rmf_traffic_msgs::msg::ParticipantDescription `rmf_traffic_ros2::convert`(**const** rmf_traffic::schedule::ParticipantDescription &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::Participants&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_ParticipantDescription.hpp

Function Documentation

rmf_traffic::schedule::ParticipantDescriptionsMap `rmf_traffic_ros2::convert`(**const** rmf_traffic_msgs::msg::Participants &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::ParticipantDescriptionsMap&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_ParticipantDescription.hpp

Function Documentation

rmf_traffic_msgs::msg::Participants `rmf_traffic_ros2::convert`(**const** rmf_traffic::schedule::ParticipantDescriptionsMap &*from*)

Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Patch::Participant&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Patch.hpp

Function Documentation

rmf_traffic_msgs::msg::ScheduleParticipantPatch `rmf_traffic_ros2::`**`convert`**`(`**`const`** rmf_traffic::schedule::Patch::Participant *&from*)

Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ScheduleParticipantPatch&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Patch.hpp

Function Documentation

rmf_traffic::schedule::Patch::Participant `rmf_traffic_ros2::`**`convert`**`(`**`const`** rmf_traffic_msgs::msg::ScheduleParticipantPatch *&from*)

Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Patch&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Patch.hpp

Function Documentation

rmf_traffic_msgs::msg::SchedulePatch `rmf_traffic_ros2::`**`convert`**`(`**`const`** rmf_traffic::schedule::Patch *&from*)

Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::SchedulePatch&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Patch.hpp

Function Documentation

rmf_traffic::schedule::Patch `rmf_traffic_ros2::`**`convert`**`(`**`const`** rmf_traffic_msgs::msg::SchedulePatch *&from*)

Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ScheduleQuerySpacetime&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Query.hpp

Function Documentation

rmf_traffic::schedule::Query::Spacetime `rmf_traffic_ros2::convert`(**const** rmf_traffic_msgs::msg::ScheduleQuerySpacetime *&from*)

Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Query::Spacetime&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Query.hpp

Function Documentation

rmf_traffic_msgs::msg::ScheduleQuerySpacetime `rmf_traffic_ros2::convert`(**const** rmf_traffic::schedule::Query::Spacetime *&from*)

Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ScheduleQueryParticipants&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Query.hpp

Function Documentation

rmf_traffic::schedule::Query::Participants `rmf_traffic_ros2::convert`(**const** rmf_traffic_msgs::msg::ScheduleQueryParticipants *&from*)

Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Query::Participants&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Query.hpp

Function Documentation

rmf_traffic_msgs::msg::ScheduleQueryParticipants `rmf_traffic_ros2::convert`(**const** rmf_traffic::schedule::Query::Participants *&from*)

Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::ScheduleQuery&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Query.hpp

Function Documentation

rmf_traffic::schedule::Query `rmf_traffic_ros2::`**`convert`**(**`const`** rmf_traffic_msgs::msg::ScheduleQuery *&from*)

Function rmf_traffic_ros2::convert(const rmf_traffic::schedule::Query&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Query.hpp

Function Documentation

rmf_traffic_msgs::msg::ScheduleQuery `rmf_traffic_ros2::`**`convert`**(**`const`** rmf_traffic::schedule::Query *&from*)

Function rmf_traffic_ros2::convert(rmf_traffic::Time)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_Time.hpp

Function Documentation

builtin_interfaces::msg::Time `rmf_traffic_ros2::`**`convert`**(rmf_traffic::Time *time*)

Function rmf_traffic_ros2::convert(builtin_interfaces::msg::Time)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_Time.hpp

Function Documentation

rmf_traffic::Time `rmf_traffic_ros2::`**`convert`**(builtin_interfaces::msg::Time *time*)

Function rmf_traffic_ros2::convert(rclcpp::Time)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_Time.hpp

Function Documentation

rmf_traffic::Time rmf_traffic_ros2::**convert**(rclcpp::Time *time*)

Function rmf_traffic_ros2::convert(rmf_traffic::Duration)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_Time.hpp

Function Documentation

rclcpp::Duration rmf_traffic_ros2::**convert**(rmf_traffic::Duration *duration*)

Function rmf_traffic_ros2::convert(rclcpp::Duration)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_Time.hpp

Function Documentation

rmf_traffic::Duration rmf_traffic_ros2::**convert**(rclcpp::Duration *duration*)

Function rmf_traffic_ros2::convert(const rmf_traffic_msgs::msg::Trajectory&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_Trajectory.hpp

Function Documentation

rmf_traffic::Trajectory rmf_traffic_ros2::**convert**(**const** rmf_traffic_msgs::msg::Trajectory &*from*)

Convert from a Trajectory message to a Trajectory instance.

If the Trajectory is malformed, this will throw a std::runtime_error describing the issue.

Function rmf_traffic_ros2::convert(const rmf_traffic::Trajectory&)

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_Trajectory.hpp

Function Documentation

rmf_traffic_msgs::msg::Trajectory rmf_traffic_ros2::**convert**(**const** rmf_traffic::Trajectory &*from*)

Convert from a Trajectory instance to a Trajectory message.

Function rmf_traffic_ros2::schedule::make_mirror

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_MirrorManager.hpp

Function Documentation

MirrorManagerFuture rmf_traffic_ros2::schedule::**make_mirror**(**const** std::shared_ptr<rclcpp::Node> &*node*, rmf_traffic::schedule::Query *query*, *MirrorManager*::*Options* *options* = *MirrorManager*::*Options*())

Initiate creation of a mirror manager that uses the given node to communicate to the RMF Traffic Schedule. It will filter its contents according to the Spacetime Query description that it is given.

Creating a mirror manager involves some asynchronous service calls to

Parameters

- [in] node: The rclcpp node to use. This will be stored as a weak_ptr, so it is okay to store the mirror manager inside of the node.
- [in] spacetime: The spacetime description to filter the query
- [in] options: Options to configure the management of the mirror

Function rmf_traffic_ros2::schedule::make_monitor_node

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_MonitorNode.hpp

Function Documentation

std::shared_ptr<rclcpp::Node> rmf_traffic_ros2::schedule::**make_monitor_node**(std::function<void()> *callback*, **const** rclcpp::NodeOptions &*options* = rclcpp::NodeOptions(), std::chrono::seconds *startup_timeout* = 10sMake a monitor node to monitor a heartbeat and restart a node when the heartbeat is lost

Function rmf_traffic_ros2::schedule::make_node

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Node.hpp

Function Documentation

std::shared_ptr<rclcpp::Node> rmf_traffic_ros2::schedule::**make_node**(**const** rclcpp::NodeOptions *&options* = rclcpp::NodeOptions())

Make a ScheduleNode instance.

Function rmf_traffic_ros2::to_ros2

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_Time.hpp

Function Documentation

rclcpp::Time rmf_traffic_ros2::**to_ros2**(rmf_traffic::Time *time*)

1.3.4 Variables

Variable rmf_fleet_adapter::AdapterDoorRequestTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**AdapterDoorRequestTopicName** = "adapter_door_requests"

Variable rmf_fleet_adapter::AdapterLiftRequestTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**AdapterLiftRequestTopicName** = "adapter_lift_requests"

Variable rmf_fleet_adapter::BidNoticeTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**BidNoticeTopicName** = "rmf_task/bid_notice"

Variable rmf_fleet_adapter::BidProposalTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**BidProposalTopicName** = "rmf_task/bid_proposal"

Variable rmf_fleet_adapter::ClosedLaneTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**ClosedLaneTopicName** = "closed_lanes"

Variable rmf_fleet_adapter::DeliveryTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**DeliveryTopicName** = "delivery_requests"

Variable rmf_fleet_adapter::DestinationRequestTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**DestinationRequestTopicName** = "destination_requests"

Variable rmf_fleet_adapter::DispatchAckTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**DispatchAckTopicName** = "rmf_task/dispatch_ack"

Variable rmf_fleet_adapter::DispatchRequestTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**DispatchRequestTopicName** = "rmf_task/dispatch_request"

Variable rmf_fleet_adapter::DispenserRequestTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**DispenserRequestTopicName** = "dispenser_requests"

Variable rmf_fleet_adapter::DispenserResultTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**DispenserResultTopicName** = "dispenser_results"

Variable rmf_fleet_adapter::DispenserStateTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**DispenserStateTopicName** = "dispenser_states"

Variable rmf_fleet_adapter::DockSummaryTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**DockSummaryTopicName** = "dock_summary"

Variable rmf_fleet_adapter::DoorStateTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**DoorStateTopicName** = "door_states"

Variable rmf_fleet_adapter::DoorSupervisorHeartbeatTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**DoorSupervisorHeartbeatTopicName** = "door_supervisor_heartbeat"

Variable rmf_fleet_adapter::FinalDoorRequestTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**FinalDoorRequestTopicName** = "door_requests"

Variable rmf_fleet_adapter::FinalLiftRequestTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**FinalLiftRequestTopicName** = "lift_requests"

Variable rmf_fleet_adapter::FleetStateTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**FleetStateTopicName** = "/fleet_states"

Variable rmf_fleet_adapter::IngestorRequestTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**IngestorRequestTopicName** = "ingestor_requests"

Variable rmf_fleet_adapter::IngestorResultTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**IngestorResultTopicName** = "ingestor_results"

Variable rmf_fleet_adapter::IngestorStateTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**IngestorStateTopicName** = "ingestor_states"

Variable rmf_fleet_adapter::InterruptRequestTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**InterruptRequestTopicName** = "robot_interrupt_request"

Variable rmf_fleet_adapter::LaneClosureRequestTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**LaneClosureRequestTopicName** = "lane_closure_requests"

Variable rmf_fleet_adapter::LiftStateTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**LiftStateTopicName** = "lift_states"

Variable rmf_fleet_adapter::LoopRequestTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**LoopRequestTopicName** = "loop_requests"

Variable rmf_fleet_adapter::ModeRequestTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**ModeRequestTopicName** = "robot_mode_requests"

Variable rmf_fleet_adapter::PathRequestTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**PathRequestTopicName** = "robot_path_requests"

Variable rmf_fleet_adapter::PauseRequestTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**PauseRequestTopicName** = "robot_pause_requests"

Variable rmf_fleet_adapter::TaskApiRequests

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**TaskApiRequests** = "task_api_requests"

Variable rmf_fleet_adapter::TaskApiResponses

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**TaskApiResponses** = "task_api_responses"

Variable rmf_fleet_adapter::TaskSummaryTopicName

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_StandardNames.hpp

Variable Documentation

const std::string rmf_fleet_adapter::**TaskSummaryTopicName** = "task_summaries"

Variable rmf_task_ros2::BidNoticeTopicName

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_task_ros2::**BidNoticeTopicName** = *Prefix* + "bid_notice"

Variable rmf_task_ros2::BidResponseTopicName

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_task_ros2**::BidResponseTopicName** = *Prefix* + "bid_response"

Variable rmf_task_ros2::CancelTaskSrvName

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_task_ros2**::CancelTaskSrvName** = "cancel_task"

Variable rmf_task_ros2::DispatchAckTopicName

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_task_ros2**::DispatchAckTopicName** = *Prefix* + "dispatch_ack"

Variable rmf_task_ros2::DispatchCommandTopicName

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_task_ros2**::DispatchCommandTopicName** = *Prefix* + "dispatch_request"

Variable rmf_task_ros2::DispatchStatesTopicName

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_task_ros2**::DispatchStatesTopicName** = "dispatch_states"

Variable rmf_task_ros2::GetDispatchStatesSrvName

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_task_ros2::**GetDispatchStatesSrvName** = "get_dispatches"

Variable rmf_task_ros2::Prefix

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_task_ros2::**Prefix** = "rmf_task/"

Variable rmf_task_ros2::SubmitTaskSrvName

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_task_ros2::**SubmitTaskSrvName** = "submit_task"

Variable rmf_task_ros2::TaskStatusTopicName

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_task_ros2::**TaskStatusTopicName** = "task_summaries"

Variable rmf_traffic_ros2::BlockadeCancelTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**BlockadeCancelTopicName** = *Prefix* + "blockade_cancel"

Variable rmf_traffic_ros2::BlockadeHeartbeatTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**BlockadeHeartbeatTopicName** = *Prefix* + "blockade_heartbeat"

Variable rmf_traffic_ros2::BlockadeReachedTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**BlockadeReachedTopicName** = *Prefix* + "blockade_reached"

Variable rmf_traffic_ros2::BlockadeReadyTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**BlockadeReadyTopicName** = *Prefix* + "blockade_ready"

Variable rmf_traffic_ros2::BlockadeReleaseTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**BlockadeReleaseTopicName** = *Prefix* + "blockade_release"

Variable rmf_traffic_ros2::BlockadeSetTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**BlockadeSetTopicName** = *Prefix* + "blockade_set"

Variable rmf_traffic_ros2::EmergencyTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**EmergencyTopicName** = "fire_alarm_trigger"

Variable rmf_traffic_ros2::FailOverEventTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**FailOverEventTopicName** = *Prefix* + "fail_over_event"

Variable rmf_traffic_ros2::HeartbeatTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**HeartbeatTopicName** = *Prefix* + "heartbeat"

Variable rmf_traffic_ros2::ItineraryClearTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**ItineraryClearTopicName** = *Prefix* + "itinerary_clear"

Variable rmf_traffic_ros2::ItineraryDelayTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**ItineraryDelayTopicName** = *Prefix* + "itinerary_delay"

Variable rmf_traffic_ros2::ItineraryExtendTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**ItineraryExtendTopicName** = *Prefix* + "itinerary_extend"

Variable rmf_traffic_ros2::ItineraryReachedTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**ItineraryReachedTopicName** = *Prefix* + "itinerary_reached"

Variable rmf_traffic_ros2::ItinerarySetTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**ItinerarySetTopicName** = *Prefix* + "itinerary_set"

Variable rmf_traffic_ros2::NegotiationAckTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**NegotiationAckTopicName** = *Prefix* + "negotiation_ack"

Variable rmf_traffic_ros2::NegotiationConclusionTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**NegotiationConclusionTopicName** = *Prefix* + "negotiation_conclusion"

Variable rmf_traffic_ros2::NegotiationForfeitTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**NegotiationForfeitTopicName** = *Prefix* + "negotiation_forfeit"

Variable rmf_traffic_ros2::NegotiationNoticeTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**NegotiationNoticeTopicName** = *Prefix* + "negotiation_notice"

Variable rmf_traffic_ros2::NegotiationProposalTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**NegotiationProposalTopicName** = *Prefix* + "negotiation_proposal"

Variable rmf_traffic_ros2::NegotiationRefusalTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**NegotiationRefusalTopicName** = *Prefix* + "negotiation_refusal"

Variable rmf_traffic_ros2::NegotiationRejectionTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**NegotiationRejectionTopicName** = *Prefix* + "negotiation_rejection"

Variable rmf_traffic_ros2::NegotiationRepeatTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**NegotiationRepeatTopicName** = *Prefix* + "negotiation_repeat"

Variable rmf_traffic_ros2::ParticipantsInfoTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**ParticipantsInfoTopicName** = *Prefix* + "participants"

Variable rmf_traffic_ros2::Prefix

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**Prefix** = "rmf_traffic/"

Variable rmf_traffic_ros2::QueriesInfoTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**QueriesInfoTopicName** = *Prefix* + "registered_queries"

Variable rmf_traffic_ros2::QueryUpdateTopicNameBase

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**QueryUpdateTopicNameBase** = *Prefix* + "query_update_"

Variable rmf_traffic_ros2::RegisterParticipantSrvName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**RegisterParticipantSrvName** = *Prefix* + "register_participant"

Variable rmf_traffic_ros2::RegisterQueryServiceName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**RegisterQueryServiceName** = *Prefix* + "register_query"

Variable rmf_traffic_ros2::RequestChangesServiceName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**RequestChangesServiceName** = *Prefix* + "request_changes"

Variable rmf_traffic_ros2::ScheduleInconsistencyTopicName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**ScheduleInconsistencyTopicName** = *Prefix* + "schedule_inconsistency"

Variable rmf_traffic_ros2::UnregisterParticipantSrvName

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_StandardNames.hpp

Variable Documentation

const std::string rmf_traffic_ros2::**UnregisterParticipantSrvName** = *Prefix* + "unregister_participant"

1.3.5 Typedefs

Typedef Duration

- Defined in file_latest_rmf_fleet_adapter_python_include_rmf_fleet_adapter_python_graph_PyEvent.hpp

Typedef Documentation

using rmf_fleet_adapter::agv::*RobotCommandHandle*::**Duration** = rmf_traffic::Duration

Typedef EventPtr

- Defined in file_latest_rmf_fleet_adapter_python_include_rmf_fleet_adapter_python_graph_PyEvent.hpp

Typedef Documentation

using EventPtr = *Lane*::EventPtr

Typedef Graph

- Defined in file_latest_rmf_fleet_adapter_python_include_rmf_fleet_adapter_python_graph_PyOrientationConstraint.hpp

Typedef Documentation

using Graph = rmf_traffic::agv::Graph

Typedef Lane

- Defined in file_latest_rmf_fleet_adapter_python_include_rmf_fleet_adapter_python_graph_PyEvent.hpp

Typedef Documentation

using Lane = rmf_traffic::agv::Graph::Lane

Typedef Lane

- Defined in file_latest_rmf_fleet_adapter_python_include_rmf_fleet_adapter_python_graph_PyExecutor.hpp

Typedef Documentation

using Lane = rmf_traffic::agv::Graph::Lane

Typedef rmf_fleet_adapter::agv::AdapterPtr

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_Adapter.hpp

Typedef Documentation

using rmf_fleet_adapter::agv::**AdapterPtr** = std::shared_ptr<*Adapter*>

Typedef rmf_fleet_adapter::agv::ConstAdapterPtr

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_Adapter.hpp

Typedef Documentation

using rmf_fleet_adapter::agv::**ConstAdapterPtr** = std::shared_ptr<**const** *Adapter*>

Typedef rmf_fleet_adapter::agv::ConstFleetUpdateHandlePtr

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_FleetUpdateHandle.hpp

Typedef Documentation

using rmf_fleet_adapter::agv::**ConstFleetUpdateHandlePtr** = std::shared_ptr<**const** *FleetUpdateHandle*>

Typedef rmf_fleet_adapter::agv::ConstRobotUpdateHandlePtr

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_RobotUpdateHandle.hpp

Typedef Documentation

using rmf_fleet_adapter::agv::**ConstRobotUpdateHandlePtr** = std::shared_ptr<**const** *RobotUpdateHandle*>

Typedef rmf_fleet_adapter::agv::EasyTrafficLightPtr

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_EasyTrafficLight.hpp

Typedef Documentation

using rmf_fleet_adapter::agv::**EasyTrafficLightPtr** = std::shared_ptr<*EasyTrafficLight*>

Typedef rmf_fleet_adapter::agv::FleetUpdateHandlePtr

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_FleetUpdateHandle.hpp

Typedef Documentation

using rmf_fleet_adapter::agv::**FleetUpdateHandlePtr** = std::shared_ptr<*FleetUpdateHandle*>

Typedef rmf_fleet_adapter::agv::RobotUpdateHandlePtr

- Defined in file_latest_rmf_fleet_adapter_include_rmf_fleet_adapter_agv_RobotUpdateHandle.hpp

Typedef Documentation

using rmf_fleet_adapter::agv::**RobotUpdateHandlePtr** = std::shared_ptr<*RobotUpdateHandle*>

Typedef rmf_task_ros2::bidding::BidNoticeMsg

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_bidding_Response.hpp

Typedef Documentation

using rmf_task_ros2::bidding::**BidNoticeMsg** = rmf_task_msgs::msg::BidNotice

Typedef rmf_task_ros2::bidding::BidProposalMsg

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_bidding_Response.hpp

Typedef Documentation

using rmf_task_ros2::bidding::**BidProposalMsg** = rmf_task_msgs::msg::BidProposal

Typedef rmf_task_ros2::bidding::BidResponseMsg

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_bidding_Response.hpp

Typedef Documentation

using rmf_task_ros2::bidding::**BidResponseMsg** = rmf_task_msgs::msg::BidResponse

Typedef rmf_task_ros2::bidding::Responses

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_bidding_Response.hpp

Typedef Documentation

using rmf_task_ros2::bidding::**Responses** = std::vector<*Response*>

Typedef rmf_task_ros2::DispatchStatePtr

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_DispatchState.hpp

Typedef Documentation

using rmf_task_ros2::**DispatchStatePtr** = std::shared_ptr<*DispatchState*>

Typedef rmf_task_ros2::TaskID

- Defined in file_latest_rmf_task_ros2_include_rmf_task_ros2_DispatchState.hpp

Typedef Documentation

using rmf_task_ros2::**TaskID** = std::string

Typedef rmf_traffic_ros2::blockade::WriterPtr

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_blockade_Writer.hpp

Typedef Documentation

using rmf_traffic_ros2::blockade::**WriterPtr** = std::shared_ptr<*Writer*>

Typedef rmf_traffic_ros2::schedule::Database

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_ParticipantRegistry.hpp

Typedef Documentation

using rmf_traffic_ros2::schedule::**Database** = rmf_traffic::schedule::Database

Typedef rmf_traffic_ros2::schedule::ParticipantDescription

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_ParticipantRegistry.hpp

Typedef Documentation

using rmf_traffic_ros2::schedule::**ParticipantDescription** = rmf_traffic::schedule::ParticipantDescription

Typedef rmf_traffic_ros2::schedule::ParticipantId

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_ParticipantRegistry.hpp

Typedef Documentation

using rmf_traffic_ros2::schedule::**ParticipantId** = rmf_traffic::schedule::ParticipantId

Typedef rmf_traffic_ros2::schedule::WriterPtr

- Defined in file_latest_rmf_traffic_ros2_include_rmf_traffic_ros2_schedule_Writer.hpp

Typedef Documentation

using rmf_traffic_ros2::schedule::**WriterPtr** = std::shared_ptr<*Writer*>

INDEX