rmf_task

Release 1.0.0

Open Source Robotics Corporation

Mar 09, 2022

CONTENTS:

A package for managing tasks in OpenRMF.

CHAPTER

ONE

RMF_TASK API

1.1 Class Hierarchy

1.2 File Hierarchy

1.3 Full API

1.3.1 Namespaces

Namespace rmf_task

Contents

Namespaces

Classes

- *Class Task::Booking*
- *Class Task::Description*
- *Class Task::Model*
- *Class Task::Tag*
- *Class TaskPlanner*
- *Class TaskPlanner::Assignment*
- *Class TaskPlanner::Configuration*
- *Class TaskPlanner::Options*
- *Class TravelEstimator*
- *Class TravelEstimator::Result*
- *Class VersionedString*
- *Class VersionedString::Reader*
- *Class VersionedString::View*

Functions

- *Function rmf_task::standard_waypoint_name*

Typedefs

- *Typedef rmf_task::ActivatorPtr*
- *Typedef rmf_task::ConstActivatorPtr*
- *Typedef rmf_task::ConstCostCalculatorPtr*
- *Typedef rmf_task::ConstLogPtr*
- *Typedef rmf_task::ConstParametersPtr*
- *Typedef rmf_task::ConstPriorityPtr*
- *Typedef rmf_task::ConstRequestFactoryPtr*
- *Typedef rmf_task::ConstRequestPtr*
- *Typedef rmf_task::ConstTravelEstimatorPtr*
- *Typedef rmf_task::CostCalculatorPtr*
- *Typedef rmf_task::PriorityPtr*
- *Typedef rmf_task::RequestFactoryPtr*
- *Typedef rmf_task::RequestPtr*

Namespace rmf_task::detail

Contents

Classes

Functions

Namespace rmf_task::events

Contents

Classes

Typedefs

Namespace rmf_task::phases

Contents

Classes

- *Class RestoreBackup*
- *Class RestoreBackup::Active*

Namespace rmf_task::requests

Contents

Classes

- *Class ChargeBattery*
- *Class ChargeBattery::Description*
- *Class ChargeBatteryFactory*
- *Class Clean*
- *Class Clean::Description*
- *Class Delivery*
- *Class Delivery::Description*
- *Class Loop*
- *Class Loop::Description*
- *Class ParkRobotFactory*

1.3.2 Classes and Structs

Template Struct CompositeData::InsertResult

- Defined in file_latest_rmf_task_include_rmf_task_CompositeData.hpp

Nested Relationships

This struct is a nested type of *Class CompositeData*.

Struct Documentation

template<typename **T**>
struct rmf_task::*CompositeData*::**InsertResult**
The result of performing an insertion operation.

Public Members

bool **inserted**
True if the value was inserted. This means that an entry of value T did not already exist before you performed the insertion.

T ***value**
A reference to the value of type T that currently exists within the *CompositeData*.

Struct Description::Info

- Defined in file_latest_rmf_task_include_rmf_task_Task.hpp

Nested Relationships

This struct is a nested type of *Class Task::Description*.

Struct Documentation

struct rmf_task::*Task*::*Description*::**Info**

Public Members

std::string **category**

std::string **detail**

Class Activator

- Defined in file_latest_rmf_task_include_rmf_task_Activator.hpp

Class Documentation

class rmf_task::**Activator**
A factory for generating *Task::Active* instances from requests.

Public Types

using Activate = std::function<*Task*::*ActivePtr* (**const** std::function<*State*) > &get_state**const** *ConstParametersPtr* &parameters, **const** *Task*::*ConstBookingPtr* &booking, **const** Description &description, std::optional<std::string> backup_state, std::function<void*Phase*::*ConstSnapshotPtr*> update, std::function<void*Task*::*Active*::*Backup*> checkpoint, std::function<void*Phase*::*ConstCompletedPtr*> phase_finished, std::function<void> task_finished>Signature for activating a task

Return an active, running instance of the requested task.

Template Parameters

- `Description`: A class that implements the *Task::Description* interface

Parameters

- `[in] get_state`: A callback for retrieving the current state of the robot
- `[in] parameters`: A reference to the parameters for the robot
- `[in] booking`: An immutable reference to the booking information for the task
- `[in] description`: The down-casted description of the task
- `[in] backup_state`: The serialized backup state of the *Task*, if the *Task* is being restored from a crash or disconnection. If the *Task* is not being restored, a std::nullopt will be passed in here.
- `[in] update`: A callback that will be triggered when the task has a significant update in its status.
- `[in] checkpoint`: A callback that will be triggered when the task has reached a task checkpoint whose state is worth backing up.
- `[in] finished`: A callback that will be triggered when the task has finished.

Public Functions

Activator()
Construct an empty TaskFactory.

template<typename **Description**>
void **add_activator**(*Activate*<*Description*> *activator*)
Add a callback to convert from a Description into an active *Task*.

Template Parameters

- `Description`: A class that implements the Request::Description interface

Parameters

- `[in] activator`: A callback that activates a *Task* matching the Description

Task::*ActivePtr* **activate**(**const** std::function<*State*) > &*get_state***const** *ConstParametersPtr* &*parameters*, **const** *Request* &*request*, std::function<void*Phase*::*ConstSnapshotPtr*> *update*, std::function<void*Task*::*Active*::*Backup*> *checkpoint*, std::function<void*Phase*::*ConstCompletedPtr*> *phase_finished*, std::function<void> *task_finished* **const**Activate a *Task* object based on a *Request*.

Return an active, running instance of the requested task.

Parameters

- `[in] get_state`: A callback for retrieving the current state of the robot
- `[in] parameters`: A reference to the parameters for the robot
- `[in] request`: The task request
- `[in] update`: A callback that will be triggered when the task has a significant update
- `[in] checkpoint`: A callback that will be triggered when the task has reached a task checkpoint whose state is worth backing up.
- `[in] phase_finished`: A callback that will be triggered whenever a task phase is finished
- `[in] task_finished`: A callback that will be triggered when the task has finished

Task::*ActivePtr* **restore**(**const** std::function<*State*) > &*get_state***const** *ConstParametersPtr* &*parameters*, **const** *Request* &*request*, std::string *backup_state*, std::function<void*Phase*::*ConstSnapshotPtr*> *update*, std::function<void*Task*::*Active*::*Backup*> *checkpoint*, std::function<void*Phase*::*ConstCompletedPtr*> *phase_finished*, std::function<void> *task_finished* **const**Restore a *Task* that crashed or disconnected.

Return an active, running instance of the requested task.

Parameters

- `[in] get_state`: A callback for retrieving the current state of the robot
- `[in] parameters`: A reference to the parameters for the robot
- `[in] request`: The task request
- `[in] backup_state`: The serialized backup state of the *Task*
- `[in] update`: A callback that will be triggered when the task has a significant update
- `[in] checkpoint`: A callback that will be triggered when the task has reached a task checkpoint whose state is worth backing up.
- `[in] phase_finished`: A callback that will be triggered whenever a task phase is finished
- `[in] task_finished`: A callback that will be triggered when the task has finished

Class BackupFileManager

- Defined in file_latest_rmf_task_include_rmf_task_BackupFileManager.hpp

Nested Relationships

Nested Types

- *Class BackupFileManager::Group*
- *Class BackupFileManager::Robot*

Class Documentation

class rmf_task::**BackupFileManager**

Public Functions

BackupFileManager(std::filesystem::path *root_directory*, std::function<void) std::string
> *info_logger* = nullptr, std::function<voidstd::string> *debug_logger* = nullptrConstruct a *BackupFileManager*

Parameters

- `[in] root_directory`: Specify the root directory that the backup files should live in

BackupFileManager &**clear_on_startup**(bool *value* = true)
Set whether any previously existing backups should be cleared out on startup. By default this behavior is turned OFF.

Parameters

- `[in] value`: True if the behavior should be turned on; false if it should be turned off.

BackupFileManager &**clear_on_shutdown**(bool *value* = true)
Set whether any currently existing backups should be cleared out on shutdown. By default this behavior is turned ON.

Parameters

- `[in] value`: True if the behavior should be turned on; false if it should be turned off.

std::shared_ptr<*Group*> **make_group**(std::string *name*)
Make a group (a.k.a. fleet) to back up.

class Group

Public Functions

std::shared_ptr<*Robot*> **make_robot**(std::string *name*)
Make a handle to backup a robot for this group

Parameters
- `[in] name`: The unique name of the robot that's being backed up

class Robot

Public Functions

std::optional<std::string> **read**() **const**
Read a backup state from file if a backup file exists for this robot. If a backup does not exist, return a nullopt.

void **write**(**const** *Task*::*Active*::*Backup* &*backup*)
Write a backup to file.

Class BackupFileManager::Group

- Defined in file_latest_rmf_task_include_rmf_task_BackupFileManager.hpp

Nested Relationships

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

Class Documentation

class rmf_task::*BackupFileManager*::**Group**

Public Functions

std::shared_ptr<*Robot*> **make_robot**(std::string *name*)
Make a handle to backup a robot for this group

Parameters
- `[in] name`: The unique name of the robot that's being backed up

Class BackupFileManager::Robot

- Defined in file_latest_rmf_task_include_rmf_task_BackupFileManager.hpp

Nested Relationships

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

Class Documentation

class rmf_task::*BackupFileManager*::**Robot**

Public Functions

std::optional<std::string> **read**() **const**
Read a backup state from file if a backup file exists for this robot. If a backup does not exist, return a nullopt.

void **write**(**const** *Task*::*Active*::*Backup* &*backup*)
Write a backup to file.

Class BinaryPriorityScheme

- Defined in file_latest_rmf_task_include_rmf_task_BinaryPriorityScheme.hpp

Class Documentation

class rmf_task::**BinaryPriorityScheme**
A class that serves as a binary prioritization scheme by genrating either high or low Priority objects for requests.

Public Static Functions

static std::shared_ptr<Priority> **make_low_priority**()
Use these to assign the task priority In the current implementation this returns a nullptr.

static std::shared_ptr<Priority> **make_high_priority**()
Get a shared pointer to a high priority object of the binary prioritization scheme.

static std::shared_ptr<CostCalculator> **make_cost_calculator**()
Use this to give the appropriate cost calculator to the task planner.

Class CompositeData

- Defined in file_latest_rmf_task_include_rmf_task_CompositeData.hpp

Nested Relationships

Nested Types

- *Template Struct CompositeData::InsertResult*

Inheritance Relationships

Derived Type

- `public rmf_task::State` (*Class State*)

Class Documentation

class `rmf_task::CompositeData`

A class that can store and return arbitrary data structures, as long as they are copyable.

Subclassed by *rmf_task::State*

Public Functions

CompositeData()

Create an empty *CompositeData*.

template<typename **T**>
InsertResult<*T*> **insert**(*T* &&*value*)

Attempt to insert some data structure into the *CompositeData*. If a data structure of type T already exists in the *CompositeData*, then this function will have no effect, and *InsertResult<T>::value* will point to the value that already existed in the *CompositeData*.

Parameters

- `[in] value`: The value to attempt to insert.

template<typename **T**>
InsertResult<*T*> **insert_or_assign**(*T* &&*value*)

Insert or assign some data structure into the *CompositeData*. If a data structure of type T already exists in the *CompositeData*, then this function will overwrite it with the new value.

Parameters

- `[in] value`: The value to insert or assign.

template<typename **T**>
CompositeData &**with**(*T* &&*value*)

Same as insert_or_assign, but *this is returned instead of the new value.

template<typename **T**>
T ***get**()
Get a reference to a data structure of type T if one is available in the *CompositeData*. If one is not available, this will return a nullptr.

template<typename **T**>
const *T* ***get**() **const**
Get a reference to an immutable data structure of type T if one is available in the *CompositeData*. If one is not available, this will return a nullptr.

template<typename **T**>
bool **erase**()
Erase the data structure of type T if one is available in the *CompositeData*. This will return true if it was erased, or false if type T was not available.

void **clear**()
Remove all data structures from this *CompositeData*.

template<typename **T**>
auto **insert**(*T* &&*value*) -> *InsertResult*<*T*>

template<typename **T**>
auto **insert_or_assign**(*T* &&*value*) -> *InsertResult*<*T*>

template<typename **T**>
struct InsertResult
The result of performing an insertion operation.

Public Members

bool **inserted**
True if the value was inserted. This means that an entry of value T did not already exist before you performed the insertion.

T ***value**
A reference to the value of type T that currently exists within the *CompositeData*.

Class Constraints

- Defined in file_latest_rmf_task_include_rmf_task_Constraints.hpp

Class Documentation

class rmf_task::**Constraints**
A class that describes constraints that are common among the agents/AGVs available for performing requests

Public Functions

Constraints(double *threshold_soc*, double *recharge_soc* = 1.0, bool *drain_battery* = true)
Constructor

Parameters

- `[in] threshold_soc`: Minimum charge level the vehicle is allowed to deplete to. This value needs to be between 0.0 and 1.0.
- `[in] recharge_soc`: The charge level the vehicle should be recharged to. This value needs to be between 0.0 and 1.0. Default value is 1.0.
- `[in] drain_battery`: If true, battery drain will be considered during task allocation and ChargeBattery tasks will automatically be included if necessary.

double **threshold_soc**() **const**
Gets the vehicle's state of charge threshold value.

Constraints &**threshold_soc**(double *threshold_soc*)
Sets the vehicle's state of charge threshold value. This value needs to be between 0.0 and 1.0.

double **recharge_soc**() **const**
Gets the vehicle's state of charge recharge value.

Constraints &**recharge_soc**(double *recharge_soc*)
Sets the vehicle's recharge state of charge value. This value needs to be between 0.0 and 1.0.

bool **drain_battery**() **const**
Get the value of drain_battery.

Constraints &**drain_battery**(bool *drain_battery*)
Set the value of drain_battery.

Class Backup

- Defined in file_latest_rmf_task_include_rmf_task_detail_Backup.hpp

Class Documentation

class rmf_task::detail::**Backup**

Public Functions

uint64_t **sequence**() **const**
Get the sequence number for this backup.

Backup &**sequence**(uint64_t *seq*)
Set the sequence number for this backup.

const std::string &**state**() **const**
Get the serialized state for this backup.

Backup &**state**(std::string *new_state*)
Set the serialized state for this backup.

Public Static Functions

static *Backup* **make**(uint64_t *seq*, std::string *state*)
Make a *Backup* state

Parameters

- `[in] seq`: Sequence number. The *Backup* from this phase with the highest sequence number will be held onto until a *Backup* with a higher sequence number is issued.
- `[in] state`: A serialization of the phase's state. This will be used by *Activator* when restoring a *Task*.

Class Resume

- Defined in file_latest_rmf_task_include_rmf_task_detail_Resume.hpp

Class Documentation

class `rmf_task::detail::`**Resume**

Public Functions

void **operator()**() **const**
Call this object to tell the *Task* to resume.

Public Static Functions

static *Resume* **make**(std::function<void)
> *callback*Make a *Resume* object. The callback will be triggered when the user triggers the *Resume*.

Class Estimate

- Defined in file_latest_rmf_task_include_rmf_task_Estimate.hpp

Class Documentation

class `rmf_task::`**Estimate**
A class to store the time that the AGV should wait till before executing the request and the state of the AGV after finishing the request. Note: The wait time is different from the earliest_start_time specified in the request definition. The wait time may be earlier to ensure that the AGV arrvies at the first location of the request by the earliest_start_time

Public Functions

Estimate(*State finish_state*, rmf_traffic::Time *wait_until*)
Constructor of an estimate of the request.

Parameters

- `[in] finish_state`: Finish state of the robot once it completes the request.
- `[in] wait_until`: The ideal time the robot starts executing this request.

State **finish_state**() **const**
Finish state of the robot once it completes the request.

Estimate &**finish_state**(*State new_finish_state*)
Sets a new finish state for the robot.

rmf_traffic::Time **wait_until**() **const**
The ideal time the robot starts executing this request.

Estimate &**wait_until**(rmf_traffic::Time *new_wait_until*)
Sets a new starting time for the robot to execute the request.

Class Event

- Defined in file_latest_rmf_task_include_rmf_task_Event.hpp

Nested Relationships

Nested Types

- *Class Event::AssignID*
- *Class Event::Snapshot*
- *Class Event::State*

Class Documentation

class rmf_task::**Event**

Public Types

enum Status
A simple computer-friendly indicator of the current status of this event. This enum may be used to automatically identify when an event requires special attention, e.g. logging a warning or alerting an operator.

Values:

enumerator Uninitialized
The event status has not been initialized. This is a sentinel value that should not generally be used.

enumerator Blocked
The event is underway but it has been blocked. The blockage may require manual intervention to fix.

enumerator Error
An error has occurred that the *Task* implementation does not know how to deal with. Manual intervention is needed to get the task back on track.

enumerator Failed
The event cannot ever finish correctly, even with manual intervention. This may mean that the *Task* cannot be completed if it does not have an automated way to recover from this failure state.

enumerator Standby
The event is on standby. It cannot be started yet, and that is its expected status.

enumerator Underway
The event is underway, and proceeding as expected.

enumerator Delayed
The event is underway but it has been temporarily delayed.

enumerator Skipped
An operator has instructed this event to be skipped.

enumerator Canceled
An operator has instructed this event to be canceled.

enumerator Killed
An operator has instructed this event to be killed.

enumerator Completed
The event has completed.

using ConstStatePtr = std::shared_ptr<**const** *State*>

using ConstSnapshotPtr = std::shared_ptr<**const** *Snapshot*>

using AssignIDPtr = std::shared_ptr<**const** *AssignID*>

Public Static Functions

static *Status* **sequence_status**(*Status earlier*, *Status later*)
Given the status of two events that are in sequence with each other, return the overall status of the sequence.

class AssignID
A utility class that helps to assign unique IDs to events.

Public Functions

AssignID()
Constructor.

uint64_t **assign**() **const**
Get a new unique ID.

Public Static Functions

static *AssignIDPtr* **make**()
Make a shared_ptr<AssignID>

class Snapshot : **public** rmf_task::*Event*::*State*
A snapshot of the state of an event. This snapshot can be read while the original event is arbitrarily changed, and there is no risk of a race condition, as long as the snapshot is not being created while the event is changing.

Public Functions

virtual uint64_t **id**() **const final**
The ID of this event, which is unique within its phase.

virtual *Status* **status**() **const final**
The current Status of this event.

virtual *VersionedString*::*View* **name**() **const final**
The "name" of this event. Ideally a short, simple piece of text that helps a human being intuit what this event is expecting at a glance.

virtual *VersionedString*::*View* **detail**() **const final**
A detailed explanation of this event.

virtual *Log*::*View* **log**() **const final**
A view of the log for this event.

virtual std::vector<*ConstStatePtr*> **dependencies**() **const final**
Get more granular dependencies of this event, if any exist.

Public Static Functions

static *ConstSnapshotPtr* **make**(**const** *State* &*other*)
Make a snapshot of the current state of an *Event*.

class State
The interface to an active event.

Subclassed by *rmf_task::Event::Snapshot*, *rmf_task::events::SimpleEventState*

Public Types

using Status = *Event*::*Status*

using ConstStatePtr = *Event*::*ConstStatePtr*

Public Functions

virtual uint64_t **id**() **const** = 0
The ID of this event, which is unique within its phase.

virtual *Status* **status**() **const** = 0
The current Status of this event.

bool **finished**() **const**
A convenience function which returns true if the event's status is any of Skipped, Canceled, Killed, or Completed.

virtual *VersionedString*::*View* **name**() **const** = 0
The "name" of this event. Ideally a short, simple piece of text that helps a human being intuit what this event is expecting at a glance.

virtual *VersionedString*::*View* **detail**() **const** = 0
A detailed explanation of this event.

virtual *Log*::*View* **log**() **const** = 0
A view of the log for this event.

virtual std::vector<*ConstStatePtr*> **dependencies**() **const** = 0
Get more granular dependencies of this event, if any exist.

virtual **~State**() = default

Class Event::AssignID

- Defined in file_latest_rmf_task_include_rmf_task_Event.hpp

Nested Relationships

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

Class Documentation

class rmf_task::*Event*::**AssignID**
A utility class that helps to assign unique IDs to events.

Public Functions

AssignID()
Constructor.

uint64_t **assign**() **const**
Get a new unique ID.

Public Static Functions

static *AssignIDPtr* **make**()
Make a shared_ptr<AssignID>

Class Event::Snapshot

- Defined in file_latest_rmf_task_include_rmf_task_Event.hpp

Nested Relationships

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

Inheritance Relationships

Base Type

- `public rmf_task::Event::State` (*Class Event::State*)

Class Documentation

class `rmf_task::`*`Event`*`::`**`Snapshot`** : **public** rmf_task::*Event*::*State*
A snapshot of the state of an event. This snapshot can be read while the original event is arbitrarily changed, and there is no risk of a race condition, as long as the snapshot is not being created while the event is changing.

Public Functions

virtual uint64_t **id**() **const final**
The ID of this event, which is unique within its phase.

virtual *Status* **status**() **const final**
The current Status of this event.

virtual *VersionedString*::*View* **name**() **const final**
The "name" of this event. Ideally a short, simple piece of text that helps a human being intuit what this event is expecting at a glance.

virtual *VersionedString*::*View* **detail**() **const final**
A detailed explanation of this event.

virtual *Log*::*View* **log**() **const final**
A view of the log for this event.

virtual std::vector<*ConstStatePtr*> **dependencies**() **const final**
Get more granular dependencies of this event, if any exist.

Public Static Functions

static *ConstSnapshotPtr* **make**(**const** *State* &*other*)
Make a snapshot of the current state of an *Event*.

Class Event::State

- Defined in file_latest_rmf_task_include_rmf_task_Event.hpp

Nested Relationships

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

Inheritance Relationships

Derived Types

- `public rmf_task::Event::Snapshot` (*Class Event::Snapshot*)
- `public rmf_task::events::SimpleEventState` (*Class SimpleEventState*)

Class Documentation

class `rmf_task::`*`Event`*`::`**`State`**
The interface to an active event.

Subclassed by *rmf_task::Event::Snapshot*, *rmf_task::events::SimpleEventState*

Public Types

using Status = *Event*::*Status*

using ConstStatePtr = *Event*::*ConstStatePtr*

Public Functions

virtual uint64_t **id**() **const** = 0
The ID of this event, which is unique within its phase.

virtual *Status* **status**() **const** = 0
The current Status of this event.

bool **finished**() **const**
A convenience function which returns true if the event's status is any of Skipped, Canceled, Killed, or Completed.

virtual *VersionedString*::*View* **name**() **const** = 0
The "name" of this event. Ideally a short, simple piece of text that helps a human being intuit what this event is expecting at a glance.

virtual *VersionedString*::*View* **detail**() **const** = 0
A detailed explanation of this event.

virtual *Log*::*View* **log**() **const** = 0
A view of the log for this event.

virtual std::vector<*ConstStatePtr*> **dependencies**() **const** = 0
Get more granular dependencies of this event, if any exist.

virtual **~State**() = default

Class SimpleEventState

- Defined in file_latest_rmf_task_include_rmf_task_events_SimpleEventState.hpp

Inheritance Relationships

Base Type

- `public rmf_task::Event::State` (*Class Event::State*)

Class Documentation

class `rmf_task::events::`**SimpleEventState** : **public** rmf_task::*Event*::*State*
This class is the simplest possible implementation for directly managing the required fields of the *Event* interface.

This may be useful if you have a *Phase* implementation that takes care of the logic for tracking your event(s) but you still need an *Event* object to satisfy the *Phase* interface's finish_event() function. Your *Phase* implementation can create an instance of this class and then manage its fields directly.

Public Functions

virtual uint64_t **id**() **const** **final**
The ID of this event, which is unique within its phase.

virtual Status **status**() **const** **final**
The current Status of this event.

SimpleEventState &**update_status**(Status *new_status*)
Update the status of this event.

virtual *VersionedString*::*View* **name**() **const** **final**
The "name" of this event. Ideally a short, simple piece of text that helps a human being intuit what this event is expecting at a glance.

SimpleEventState &**update_name**(std::string *new_name*)
Update the name of this event.

virtual *VersionedString*::*View* **detail**() **const** **final**
A detailed explanation of this event.

SimpleEventState &**update_detail**(std::string *new_detail*)
Update the detail of this event.

virtual *Log*::*View* **log**() **const** **final**
A view of the log for this event.

Log &**update_log**()
Update the log.

virtual std::vector<ConstStatePtr> **dependencies**() **const final**
Get more granular dependencies of this event, if any exist.

SimpleEventState &**update_dependencies**(std::vector<ConstStatePtr> *new_dependencies*)
Update the dependencies.

SimpleEventState &**add_dependency**(ConstStatePtr *new_dependency*)
Add one dependency to the state.

Public Static Functions

static std::shared_ptr<*SimpleEventState*> **make**(uint64_t *id*, std::string *name*, std::string *detail*, Status *initial_status*, std::vector<ConstStatePtr> *dependencies* = {}, std::function<rmf_traffic::Time)
> *clock* = nullptr

Class Header

- Defined in file_latest_rmf_task_include_rmf_task_Header.hpp

Class Documentation

class rmf_task::**Header**

Public Functions

Header(std::string *category_*, std::string *detail_*, rmf_traffic::Duration *estimate_*)
Constructor

Parameters

- [in] category_: Category of the subject
- [in] detail_: Details about the subject
- [in] estimate_: The original (ideal) estimate of how long the subject will last

const std::string &**category**() **const**
Category of the subject.

const std::string &**detail**() **const**
Details about the subject.

rmf_traffic::Duration **original_duration_estimate**() **const**
The original (ideal) estimate of how long the subject will last.

Class Log

- Defined in file_latest_rmf_task_include_rmf_task_Log.hpp

Nested Relationships

Nested Types

- *Class Log::Entry*
- *Class Log::Reader*
- *Class Reader::Iterable*
- *Class Iterable::iterator*
- *Class Log::View*

Class Documentation

class rmf_task::**Log**

Public Types

enum Tier
A computer-friendly ranking of how serious the log entry is.

Values:

enumerator Uninitialized
This is a sentinel value that should not generally be used.

enumerator Info
An expected occurrence took place.

enumerator Warning
An unexpected, problematic occurrence took place, but it can be recovered from. Human attention is recommended but not necessary.

enumerator Error
A problem happened, and humans should be alerted.

Public Functions

Log(std::function<rmf_traffic::Time)
> *clock* = nullptrConstruct a log.

Parameters

- `[in] clock`: Specify a clock for this log to use. If nullptr is given, then std::chrono::system_clock::now() will be used.

void **info**(std::string *text*)
Add an informational entry to the log.

void **warn**(std::string *text*)
Add a warning to the log.

void **error**(std::string *text*)
Add an error to the log.

void **push**(*Tier* *tier*, std::string *text*)
Push an entry of the specified severity.

void **insert**(*Log*::*Entry* *entry*)
Insert an arbitrary entry into the log.

View **view**() **const**
Get a *View* of the current state of this log. Any new entries that are added after calling this function will not be visible to the *View* that is returned.

class Entry
A single entry within the log.

Public Functions

Tier **tier**() **const**
What was the tier of this entry.

uint32_t **seq**() **const**
Sequence number for this log entry. This increments once for each new log entry, until overflowing and wrapping around to 0.

rmf_traffic::Time **time**() **const**
What was the timestamp of this entry.

const std::string &**text**() **const**
What was the text of this entry.

class Reader
A *Reader* that can iterate through the Views of Logs. The *Reader* will keep track of which Entries have already been viewed, so every *Entry* read by a single *Reader* instance is unique.

Public Functions

Reader()
Construct a *Reader*.

Iterable **read**(**const** *View* &*view*)
Create an object that can iterate through the entries of a *View*. Any entries that have been read by this *Reader* in the past will be skipped. This can be used in a range-based for loop, e.g.:

```
for (const auto& entry : reader.read(view))
{
  std::cout << entry.text() << std::endl;
}
```

class Iterable

Public Types

using const_iterator = *iterator*

Public Functions

iterator **begin**() **const**
Get the beginning iterator of the read.

iterator **end**() **const**
Get the ending iterator of the read.

class iterator

Public Functions

const *Entry* &**operator***() **const**
Dereference operator.

const *Entry* ***operator->**() **const**
Drill-down operator.

iterator &**operator++**()
Pre-increment operator: ++it

Note This is more efficient than the post-increment operator.
Warning It is undefined behavior to perform this operation on an iterator that is equal to *Log::Reader::Iterable::end()*.
Return a reference to the iterator itself

iterator **operator++**(int)
Post-increment operator: it++

Warning It is undefined behavior to perform this operation on an iterator that is equal to *Log::Reader::Iterable::end()*.
Return a copy of the iterator before it was incremented.

bool **operator==**(**const** *iterator* &*other*) **const**
Equality comparison operator.

bool **operator!=**(**const** *iterator* &*other*) **const**
Inequality comparison operator.

class View
A snapshot view of the log's contents. This is thread-safe to read even while new entries are being added to the log, but those new entries will not be seen by this *View*. You must retrieve a new *View* to see new entries.

Class Log::Entry

- Defined in file_latest_rmf_task_include_rmf_task_Log.hpp

Nested Relationships

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

Class Documentation

class rmf_task::*Log*::**Entry**
A single entry within the log.

Public Functions

Tier **tier**() **const**
What was the tier of this entry.

uint32_t **seq**() **const**
Sequence number for this log entry. This increments once for each new log entry, until overflowing and wrapping around to 0.

rmf_traffic::Time **time**() **const**
What was the timestamp of this entry.

const std::string &**text**() **const**
What was the text of this entry.

Class Log::Reader

- Defined in file_latest_rmf_task_include_rmf_task_Log.hpp

Nested Relationships

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

Nested Types

- *Class Reader::Iterable*
- *Class Iterable::iterator*

Class Documentation

class rmf_task::*Log*::**Reader**

A *Reader* that can iterate through the Views of Logs. The *Reader* will keep track of which Entries have already been viewed, so every *Entry* read by a single *Reader* instance is unique.

Public Functions

Reader()

Construct a *Reader*.

Iterable **read**(**const** *View* &*view*)

Create an object that can iterate through the entries of a *View*. Any entries that have been read by this *Reader* in the past will be skipped. This can be used in a range-based for loop, e.g.:

```
for (const auto& entry : reader.read(view))
{
  std::cout << entry.text() << std::endl;
}
```

class Iterable

Public Types

using const_iterator = *iterator*

Public Functions

iterator **begin**() **const**

Get the beginning iterator of the read.

iterator **end**() **const**

Get the ending iterator of the read.

class iterator

Public Functions

const *Entry* &**operator***() **const**

Dereference operator.

const *Entry* ***operator->**() **const**

Drill-down operator.

iterator &**operator++**()

Pre-increment operator: ++it

Note This is more efficient than the post-increment operator.

Warning It is undefined behavior to perform this operation on an iterator that is equal to *Log::Reader::Iterable::end()*.

Return a reference to the iterator itself

iterator **operator++**(int)
Post-increment operator: it++

Warning It is undefined behavior to perform this operation on an iterator that is equal to *Log::Reader::Iterable::end()*.
Return a copy of the iterator before it was incremented.

bool **operator==**(**const** *iterator* &*other*) **const**
Equality comparison operator.

bool **operator!=**(**const** *iterator* &*other*) **const**
Inequality comparison operator.

Class Reader::Iterable

- Defined in file_latest_rmf_task_include_rmf_task_Log.hpp

Nested Relationships

This class is a nested type of *Class Log::Reader*.

Nested Types

- *Class Iterable::iterator*

Class Documentation

class rmf_task::*Log*::*Reader*::**Iterable**

Public Types

using const_iterator = *iterator*

Public Functions

iterator **begin**() **const**
Get the beginning iterator of the read.

iterator **end**() **const**
Get the ending iterator of the read.

class iterator

Public Functions

const *Entry* &**operator*** () **const**
Dereference operator.

const *Entry* ***operator->** () **const**
Drill-down operator.

iterator &**operator++** ()
Pre-increment operator: ++it

Note This is more efficient than the post-increment operator.
Warning It is undefined behavior to perform this operation on an iterator that is equal to *Log::Reader::Iterable::end()*.
Return a reference to the iterator itself

iterator **operator++** (int)
Post-increment operator: it++

Warning It is undefined behavior to perform this operation on an iterator that is equal to *Log::Reader::Iterable::end()*.
Return a copy of the iterator before it was incremented.

bool **operator==** (**const** *iterator* &*other*) **const**
Equality comparison operator.

bool **operator!=** (**const** *iterator* &*other*) **const**
Inequality comparison operator.

Class Iterable::iterator

- Defined in file_latest_rmf_task_include_rmf_task_Log.hpp

Nested Relationships

This class is a nested type of *Class Reader::Iterable*.

Class Documentation

class rmf_task::*Log*::*Reader*::*Iterable*::**iterator**

Public Functions

const *Entry* &**operator*** () **const**
Dereference operator.

const *Entry* ***operator->** () **const**
Drill-down operator.

iterator &**operator++** ()
Pre-increment operator: ++it

Note This is more efficient than the post-increment operator.

Warning It is undefined behavior to perform this operation on an iterator that is equal to *Log::Reader::Iterable::end()*.

Return a reference to the iterator itself

iterator **operator++**(int)
Post-increment operator: it++

Warning It is undefined behavior to perform this operation on an iterator that is equal to *Log::Reader::Iterable::end()*.

Return a copy of the iterator before it was incremented.

bool **operator==**(**const** *iterator* &*other*) **const**
Equality comparison operator.

bool **operator!=**(**const** *iterator* &*other*) **const**
Inequality comparison operator.

Class Log::View

- Defined in file_latest_rmf_task_include_rmf_task_Log.hpp

Nested Relationships

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

Class Documentation

class View
A snapshot view of the log's contents. This is thread-safe to read even while new entries are being added to the log, but those new entries will not be seen by this *View*. You must retrieve a new *View* to see new entries.

Class Parameters

- Defined in file_latest_rmf_task_include_rmf_task_Parameters.hpp

Class Documentation

class rmf_task::**Parameters**
A class that containts parameters that are common among the agents/AGVs available for performing requests

Public Functions

Parameters(std::shared_ptr<**const** rmf_traffic::agv::Planner> *planner*, rmf_battery::agv::BatterySystem *battery_system*, rmf_battery::ConstMotionPowerSinkPtr *motion_sink*, rmf_battery::ConstDevicePowerSinkPtr *ambient_sink*, rmf_battery::ConstDevicePowerSinkPtr *tool_sink* = nullptr)
Constructor

Parameters

- `[in] battery_system`: The battery system of the agent
- `[in] motion_sink`: The motion sink of the agent. This describes how power gets drained while the agent is moving.
- `[in] ambient_sink`: The ambient device sink of the agent. This describes how power gets drained at all times from passive use of the agent's electronics.
- `[in] planner`: The planner for a agent in this fleet
- `[in] tool_sink`: An additional device sink of the agent. This describes how power gets drained when a tool/device is active.

const std::shared_ptr<**const** rmf_traffic::agv::Planner> &**planner**() **const**
Get the planner.

Parameters &**planner**(std::shared_ptr<**const** rmf_traffic::agv::Planner> *planner*)
Set the planner.

const rmf_battery::agv::BatterySystem &**battery_system**() **const**
Get the battery system.

Parameters &**battery_system**(rmf_battery::agv::BatterySystem *battery_system*)
Set the battery_system.

const rmf_battery::ConstMotionPowerSinkPtr &**motion_sink**() **const**
Get the motion sink.

Parameters &**motion_sink**(rmf_battery::ConstMotionPowerSinkPtr *motion_sink*)
Set the motion_sink.

const rmf_battery::ConstDevicePowerSinkPtr &**ambient_sink**() **const**
Get the ambient device sink.

Parameters &**ambient_sink**(rmf_battery::ConstDevicePowerSinkPtr *ambient_sink*)
Set the ambient device sink.

const rmf_battery::ConstDevicePowerSinkPtr &**tool_sink**() **const**
Get the tool device sink.

Parameters &**tool_sink**(rmf_battery::ConstDevicePowerSinkPtr *tool_sink*)
Set the tool device sink.

Class Payload

- Defined in file_latest_rmf_task_include_rmf_task_Payload.hpp

Nested Relationships

Nested Types

- *Class Payload::Component*

Class Documentation

class rmf_task::**Payload**

Public Functions

Payload(std::vector<*Component*> *components*)
Constructor.

const std::vector<*Component*> &**components**() **const**
Components in the payload.

std::string **brief**(**const** std::string &*compartment_prefix* = "in") **const**
A brief human-friendly description of the payload

Parameters

- [in] compartment_prefix: The prefix to use when describing the compartments

class Component

Public Functions

Component(std::string *sku*, uint32_t *quantity*, std::string *compartment*)
Constructor.

const std::string &**sku**() **const**
Stock Keeping Unit (SKU) for this component of the payload.

uint32_t **quantity**() **const**
The quantity of the specified item in this component of the payload.

const std::string &**compartment**() **const**
The name of the compartment.

Class Payload::Component

- Defined in file_latest_rmf_task_include_rmf_task_Payload.hpp

Nested Relationships

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

Class Documentation

class rmf_task::*Payload*::**Component**

Public Functions

Component(std::string *sku*, uint32_t *quantity*, std::string *compartment*)
Constructor.

const std::string &**sku**() **const**
Stock Keeping Unit (SKU) for this component of the payload.

uint32_t **quantity**() **const**
The quantity of the specified item in this component of the payload.

const std::string &**compartment**() **const**
The name of the compartment.

Class Phase

- Defined in file_latest_rmf_task_include_rmf_task_Phase.hpp

Nested Relationships

Nested Types

- *Class Phase::Active*
- *Class Phase::Completed*
- *Class Phase::Pending*
- *Class Phase::Snapshot*
- *Class Phase::Tag*

Inheritance Relationships

Derived Type

- `public rmf_task::phases::RestoreBackup` (*Class RestoreBackup*)

Class Documentation

class `rmf_task::`**Phase**

Subclassed by *rmf_task::phases::RestoreBackup*

Public Types

using **ConstTagPtr** = std::shared_ptr<**const** *Tag*>

using **ConstCompletedPtr** = std::shared_ptr<**const** *Completed*>

using **ConstActivePtr** = std::shared_ptr<**const** *Active*>

using **ConstSnapshotPtr** = std::shared_ptr<**const** *Snapshot*>

class **Active**

Subclassed by *rmf_task::Phase::Snapshot*, *rmf_task::phases::RestoreBackup::Active*

Public Functions

virtual *ConstTagPtr* **tag**() **const** = 0

Tag of the phase.

virtual *Event*::*ConstStatePtr* **final_event**() **const** = 0

The *Event* that needs to finish for this phase to be complete.

virtual rmf_traffic::Duration **estimate_remaining_time**() **const** = 0

The estimated finish time for this phase.

virtual **~Active**() = default

class **Completed**

Information about a phase that has been completed.

Public Functions

Completed(*ConstSnapshotPtr* *snapshot_*, rmf_traffic::Time *start_*, rmf_traffic::Time *finish_*)

Constructor.

const *ConstSnapshotPtr* &**snapshot**() **const**

The final log of the phase.

rmf_traffic::Time **start_time**() **const**

The actual time that the phase started.

rmf_traffic::Time **finish_time**() **const**

The actual time that the phase finished.

class **Pending**

Public Functions

Pending(*ConstTagPtr tag*)
Constructor.

const *ConstTagPtr* &**tag**() **const**
Tag of the phase.

Pending &**will_be_skipped**(bool *value*)
Change whether this pending phase will be skipped.

bool **will_be_skipped**() **const**
Check if this phase is set to be skipped.

class Snapshot : **public** rmf_task::*Phase*::*Active*

Public Functions

virtual *ConstTagPtr* **tag**() **const final**
Tag of the phase.

virtual *Event*::*ConstStatePtr* **final_event**() **const final**
The *Event* that needs to finish for this phase to be complete.

virtual rmf_traffic::Duration **estimate_remaining_time**() **const final**
The estimated finish time for this phase.

Public Static Functions

static *ConstSnapshotPtr* **make**(**const** *Active* &*active*)
Make a snapshot of an *Active* phase.

class Tag
Basic static information about a phase. This information should go unchanged from the *Pending* state, through the *Active* state, and into the *Completed* state.

Public Types

using Id = uint64_t

Public Functions

Tag(*Id id_*, *Header header_*)
Constructor

Parameters
- `[in] id_`: ID of the phase. This phase ID must be unique within its *Task* instance.
- `[in] header_`: *Header* of the phase.

Id **id**() **const**
Unique ID of the phase.

const *Header* &**header**() **const**
Header of the phase, containing human-friendly high-level information about the phase.

Class Phase::Active

- Defined in file_latest_rmf_task_include_rmf_task_Phase.hpp

Nested Relationships

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

Inheritance Relationships

Derived Types

- `public rmf_task::Phase::Snapshot` (*Class Phase::Snapshot*)
- `public rmf_task::phases::RestoreBackup::Active` (*Class RestoreBackup::Active*)

Class Documentation

class `rmf_task::Phase::Active`
Subclassed by *rmf_task::Phase::Snapshot*, *rmf_task::phases::RestoreBackup::Active*

Public Functions

virtual *ConstTagPtr* **tag**`()` **const** = 0
Tag of the phase.

virtual *Event*::*ConstStatePtr* **final_event**`()` **const** = 0
The *Event* that needs to finish for this phase to be complete.

virtual rmf_traffic::Duration **estimate_remaining_time**`()` **const** = 0
The estimated finish time for this phase.

virtual **~Active**`()` = default

Class Phase::Completed

- Defined in file_latest_rmf_task_include_rmf_task_Phase.hpp

Nested Relationships

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

Class Documentation

class rmf_task::*Phase*::**Completed**

Information about a phase that has been completed.

Public Functions

Completed(*ConstSnapshotPtr* *snapshot_*, rmf_traffic::Time *start_*, rmf_traffic::Time *finish_*)

Constructor.

const *ConstSnapshotPtr* &**snapshot**() **const**

The final log of the phase.

rmf_traffic::Time **start_time**() **const**

The actual time that the phase started.

rmf_traffic::Time **finish_time**() **const**

The actual time that the phase finished.

Class Phase::Pending

- Defined in file_latest_rmf_task_include_rmf_task_Phase.hpp

Nested Relationships

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

Class Documentation

class rmf_task::*Phase*::**Pending**

Public Functions

Pending(*ConstTagPtr* *tag*)

Constructor.

const *ConstTagPtr* &**tag**() **const**

Tag of the phase.

Pending &**will_be_skipped**(bool *value*)

Change whether this pending phase will be skipped.

bool **will_be_skipped**() **const**

Check if this phase is set to be skipped.

Class Phase::Snapshot

- Defined in file_latest_rmf_task_include_rmf_task_Phase.hpp

Nested Relationships

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

Inheritance Relationships

Base Type

- `public rmf_task::Phase::Active` (*Class Phase::Active*)

Class Documentation

class rmf_task::*Phase*::**Snapshot** : **public** rmf_task::*Phase*::*Active*

Public Functions

virtual *ConstTagPtr* **tag**() **const final**
Tag of the phase.

virtual *Event*::*ConstStatePtr* **final_event**() **const final**
The *Event* that needs to finish for this phase to be complete.

virtual rmf_traffic::Duration **estimate_remaining_time**() **const final**
The estimated finish time for this phase.

Public Static Functions

static *ConstSnapshotPtr* **make**(**const** *Active* &*active*)
Make a snapshot of an *Active* phase.

Class Phase::Tag

- Defined in file_latest_rmf_task_include_rmf_task_Phase.hpp

Nested Relationships

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

Class Documentation

class rmf_task::*Phase*::**Tag**

Basic static information about a phase. This information should go unchanged from the *Pending* state, through the *Active* state, and into the *Completed* state.

Public Types

using Id = uint64_t

Public Functions

Tag(*Id id_*, *Header header_*)

Constructor

Parameters

- [in] id_: ID of the phase. This phase ID must be unique within its *Task* instance.
- [in] header_: *Header* of the phase.

Id **id**() **const**

Unique ID of the phase.

const *Header* &**header**() **const**

Header of the phase, containing human-friendly high-level information about the phase.

Class RestoreBackup

- Defined in file_latest_rmf_task_include_rmf_task_phases_RestoreBackup.hpp

Nested Relationships

Nested Types

- *Class RestoreBackup::Active*

Inheritance Relationships

Base Type

- public rmf_task::Phase (*Class Phase*)

Class Documentation

class rmf_task::phases::**RestoreBackup** : **public** rmf_task::*Phase*

Public Types

using **ActivePtr** = std::shared_ptr<*Active*>

class **Active** : **public** rmf_task::*Phase*::*Active*

This is a special phase type designated for restoring the backup of a task.

This phase type uses a reserved phase ID of 0.

Public Functions

virtual ConstTagPtr **tag**() **const** **final**

Tag of the phase.

virtual *Event*::*ConstStatePtr* **final_event**() **const** **final**

The *Event* that needs to finish for this phase to be complete.

virtual rmf_traffic::Duration **estimate_remaining_time**() **const** **final**

The estimated finish time for this phase.

void **parsing_failed**(**const** std::string &*error_message*)

Call this function if the parsing fails.

void **restoration_failed**(**const** std::string &*error_message*)

Call this function if the restoration fails for some reason besides parsing

void **restoration_succeeded**()

Call this function if the parsing succeeds.

Log &**update_log**()

Get the log to pass in some other kind of message.

Public Static Functions

static *ActivePtr* **make**(**const** std::string &*backup_state_str*, rmf_traffic::Duration *estimated_remaining_time*)

Make an active *RestoreBackup* phase.

Class RestoreBackup::Active

- Defined in file_latest_rmf_task_include_rmf_task_phases_RestoreBackup.hpp

Nested Relationships

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

Inheritance Relationships

Base Type

- `public rmf_task::Phase::Active` (*Class Phase::Active*)

Class Documentation

class `rmf_task::phases::RestoreBackup::Active` : **public** rmf_task::*Phase*::*Active*

This is a special phase type designated for restoring the backup of a task.

This phase type uses a reserved phase ID of 0.

Public Functions

virtual ConstTagPtr **tag**() **const final**

Tag of the phase.

virtual *Event*::*ConstStatePtr* **final_event**() **const final**

The *Event* that needs to finish for this phase to be complete.

virtual rmf_traffic::Duration **estimate_remaining_time**() **const final**

The estimated finish time for this phase.

void **parsing_failed**(**const** std::string &*error_message*)

Call this function if the parsing fails.

void **restoration_failed**(**const** std::string &*error_message*)

Call this function if the restoration fails for some reason besides parsing

void **restoration_succeeded**()

Call this function if the parsing succeeds.

Log &**update_log**()

Get the log to pass in some other kind of message.

Public Static Functions

static *ActivePtr* **make**(**const** std::string &*backup_state_str*, rmf_traffic::Duration *estimated_remaining_time*)

Make an active *RestoreBackup* phase.

Class Request

- Defined in file_latest_rmf_task_include_rmf_task_Request.hpp

Class Documentation

class rmf_task::**Request**

Public Functions

Request(**const** std::string &*id*, rmf_traffic::Time *earliest_start_time*, *ConstPriorityPtr* *priority*, *Task*::*ConstDescriptionPtr* *description*, bool *automatic* = false)
Constructor

Parameters

- [in] earliest_start_time: The desired start time for this request
- [in] priority: The priority for this request. This is provided by the Priority Scheme. For requests that do not have any priority this is a nullptr.
- [in] description: The description for this request
- [in] automatic: True if this request is auto-generated

Request(*Task*::*ConstBookingPtr* *booking*, *Task*::*ConstDescriptionPtr* *description*)
Constructor

Parameters

- [in] booking: Booking information for this request
- [in] description: Description for this request

const *Task*::*ConstBookingPtr* &**booking**() **const**
Get the tag of this request.

const *Task*::*ConstDescriptionPtr* &**description**() **const**
Get the description of this request.

Class RequestFactory

- Defined in file_latest_rmf_task_include_rmf_task_RequestFactory.hpp

Inheritance Relationships

Derived Types

- `public rmf_task::requests::ChargeBatteryFactory` (*Class ChargeBatteryFactory*)
- `public rmf_task::requests::ParkRobotFactory` (*Class ParkRobotFactory*)

Class Documentation

class `rmf_task::`**RequestFactory**

An abstract interface for generating a tailored request for an AGV given.

Subclassed by *rmf_task::requests::ChargeBatteryFactory*, *rmf_task::requests::ParkRobotFactory*

Public Functions

virtual *ConstRequestPtr* **make_request**(**const** *State* &*state*) **const** = 0

Generate a request for the AGV given the state that the robot will have when it is ready to perform the request

virtual ~RequestFactory() = default

Class ChargeBattery

- Defined in file_latest_rmf_task_include_rmf_task_requests_ChargeBattery.hpp

Nested Relationships

Nested Types

- *Class ChargeBattery::Description*

Class Documentation

class `rmf_task::requests::`**ChargeBattery**

A class that generates a *Request* which requires an AGV to return to its desginated charging_waypoint as specified in its agv::State and wait till its battery charges up to the recharge_soc confugred in agv::Constraints

Public Static Functions

static *ConstRequestPtr* **make**(rmf_traffic::Time *earliest_start_time*, *ConstPriorityPtr* *priority* = nullptr, bool *automatic* = true)

Generate a chargebattery request

Parameters

- `[in] earliest_start_time`: The desired start time for this request
- `[in] priority`: The priority for this request

- `[in] automatic`: True if this request is auto-generated

class Description : **public** rmf_task::*Task*::*Description*

Public Functions

virtual *Task*::*ConstModelPtr* **make_model**(rmf_traffic::Time *earliest_start_time*, **const** *Parameters* &*parameters*) **const final**

Generate a Model for the task based on the unique traits of this description

Parameters

- `[in] earliest_start_time`: The earliest time this task should begin execution. This is usually the requested start time for the task.
- `[in] parameters`: The parameters that describe this AGV

virtual Info **generate_info**(**const** *State* &*initial_state*, **const** *Parameters* &*parameters*) **const final**

Generate a plain text info description for the task, given the predicted initial state and the task planning parameters.

Parameters

- `[in] initial_state`: The predicted initial state for the task
- `[in] parameters`: The task planning parameters

Public Static Functions

static *Task*::*ConstDescriptionPtr* **make**()

Generate the description for this request.

Class ChargeBattery::Description

- Defined in file_latest_rmf_task_include_rmf_task_requests_ChargeBattery.hpp

Nested Relationships

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

Inheritance Relationships

Base Type

- `public rmf_task::Task::Description` (*Class Task::Description*)

Class Documentation

class rmf_task::requests::*ChargeBattery*::**Description** : **public** rmf_task::*Task*::*Description*

Public Functions

virtual *Task*::*ConstModelPtr* **make_model**(rmf_traffic::Time *earliest_start_time*, **const** *Parameters* &*parameters*) **const final**

Generate a Model for the task based on the unique traits of this description

Parameters

- `[in] earliest_start_time`: The earliest time this task should begin execution. This is usually the requested start time for the task.
- `[in] parameters`: The parameters that describe this AGV

virtual Info **generate_info**(**const** *State* &*initial_state*, **const** *Parameters* &*parameters*) **const final**

Generate a plain text info description for the task, given the predicted initial state and the task planning parameters.

Parameters

- `[in] initial_state`: The predicted initial state for the task
- `[in] parameters`: The task planning parameters

Public Static Functions

static *Task*::*ConstDescriptionPtr* **make**()

Generate the description for this request.

Class ChargeBatteryFactory

- Defined in file_latest_rmf_task_include_rmf_task_requests_ChargeBatteryFactory.hpp

Inheritance Relationships

Base Type

- `public rmf_task::RequestFactory` (*Class RequestFactory*)

Class Documentation

class rmf_task::requests::**ChargeBatteryFactory** : **public** rmf_task::*RequestFactory*

The *ChargeBatteryFactory* will generate a *ChargeBattery* request which requires an AGV to return to its desginated charging_waypoint as specified in its agv::State and wait till its battery charges up to the recharge_soc confugred in agv::Constraints recharge_soc specified in its agv::Constraints

Public Functions

ChargeBatteryFactory()

virtual *ConstRequestPtr* **make_request**(**const** *State* &*state*) **const final**

Documentation inherited.

Class Clean

- Defined in file_latest_rmf_task_include_rmf_task_requests_Clean.hpp

Nested Relationships

Nested Types

- *Class Clean::Description*

Class Documentation

class rmf_task::requests::**Clean**

A class that generates a *Request* which requires an AGV to perform a cleaning operation at a location

Public Static Functions

static *ConstRequestPtr* **make**(std::size_t *start_waypoint*, std::size_t *end_waypoint*, **const** rmf_traffic::Trajectory &*cleaning_path*, **const** std::string &*id*, rmf_traffic::Time *earliest_start_time*, *ConstPriorityPtr* *priority* = nullptr, bool *automatic* = false)

Generate a clean request

Parameters

- [in] start_waypoint: The graph index for the location where the AGV should begin its cleaning operation
- [in] end_waypoint: The graph index for the location where the AGV ends up after its cleaning operation
- [in] cleaning_path: A trajectory that describes the motion of the AGV during the cleaning operation. This is used to determine the process duration and expected battery drain
- [in] id: A unique id for this request
- [in] earliest_start_time: The desired start time for this request

- `[in] priority`: The priority for this request
- `[in] automatic`: True if this request is auto-generated

class Description : **public** rmf_task::*Task*::*Description*

Public Functions

virtual *Task*::*ConstModelPtr* **make_model**(rmf_traffic::Time *earliest_start_time*, **const** *Parameters* &*parameters*) **const final**

Generate a Model for the task based on the unique traits of this description

Parameters

- `[in] earliest_start_time`: The earliest time this task should begin execution. This is usually the requested start time for the task.
- `[in] parameters`: The parameters that describe this AGV

virtual Info **generate_info**(**const** *State* &*initial_state*, **const** *Parameters* &*parameters*) **const final**

Generate a plain text info description for the task, given the predicted initial state and the task planning parameters.

Parameters

- `[in] initial_state`: The predicted initial state for the task
- `[in] parameters`: The task planning parameters

std::size_t **start_waypoint**() **const**

Get the start waypoint in this request.

std::size_t **end_waypoint**() **const**

Get the end waypoint in this request.

Public Static Functions

static std::shared_ptr<*Description*> **make**(std::size_t *start_waypoint*, std::size_t *end_waypoint*, **const** rmf_traffic::Trajectory &*cleaning_path*)

Generate the description for this request.

Class Clean::Description

- Defined in file_latest_rmf_task_include_rmf_task_requests_Clean.hpp

Nested Relationships

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

Inheritance Relationships

Base Type

- `public rmf_task::Task::Description` (*Class Task::Description*)

Class Documentation

class `rmf_task::requests::Clean::Description` : **public** rmf_task::*Task*::*Description*

Public Functions

virtual *Task*::*ConstModelPtr* **make_model**(rmf_traffic::Time *earliest_start_time*, **const** *Parameters* &*parameters*) **const final**

Generate a Model for the task based on the unique traits of this description

Parameters

- `[in] earliest_start_time`: The earliest time this task should begin execution. This is usually the requested start time for the task.
- `[in] parameters`: The parameters that describe this AGV

virtual Info **generate_info**(**const** *State* &*initial_state*, **const** *Parameters* &*parameters*) **const final**

Generate a plain text info description for the task, given the predicted initial state and the task planning parameters.

Parameters

- `[in] initial_state`: The predicted initial state for the task
- `[in] parameters`: The task planning parameters

std::size_t **start_waypoint**() **const**

Get the start waypoint in this request.

std::size_t **end_waypoint**() **const**

Get the end waypoint in this request.

Public Static Functions

static std::shared_ptr<*Description*> **make**(std::size_t *start_waypoint*, std::size_t *end_waypoint*, **const** rmf_traffic::Trajectory &*cleaning_path*)

Generate the description for this request.

Class Delivery

- Defined in file_latest_rmf_task_include_rmf_task_requests_Delivery.hpp

Nested Relationships

Nested Types

- *Class Delivery::Description*

Class Documentation

class `rmf_task::requests::Delivery`

A class that generates a *Request* which requires an AGV to pickup items from one location and deliver them to another

Public Static Functions

static *ConstRequestPtr* **make**(std::size_t *pickup_waypoint*, rmf_traffic::Duration *pickup_wait*, std::size_t *dropoff_waypoint*, rmf_traffic::Duration *dropoff_wait*, *Payload* *payload*, **const** std::string &*id*, rmf_traffic::Time *earliest_start_time*, *ConstPriorityPtr* *priority* = nullptr, bool *automatic* = false, std::string *pickup_from_dispenser* = "", std::string *dropoff_to_ingestor* = "")

Generate a delivery request

Parameters

- `[in] pickup_waypoint`: The graph index for the pickup location
- `[in] pickup_wait`: The expected duration the AGV has to wait at the pickup location for the items to be loaded
- `[in] dropoff_waypoint`: The graph index for the dropoff location
- `[in] dropoff_wait`: The expected duration the AGV has to wait at the dropoff location for the items to be unloaded
- `[in] id`: A unique id for this request
- `[in] earliest_start_time`: The desired start time for this request
- `[in] priority`: The priority for this request
- `[in] automatic`: True if this request is auto-generated

class Description : **public** rmf_task::*Task*::*Description*

Public Types

using Start = rmf_traffic::agv::Planner::Start

Public Functions

virtual *Task*::*ConstModelPtr* **make_model**(rmf_traffic::Time *earliest_start_time*, **const** *Parameters* &*parameters*) **const final**

Generate a Model for the task based on the unique traits of this description

Parameters

- `[in] earliest_start_time`: The earliest time this task should begin execution. This is usually the requested start time for the task.
- `[in] parameters`: The parameters that describe this AGV

virtual Info **generate_info**(**const** *State* &*initial_state*, **const** *Parameters* &*parameters*) **const final**

Generate a plain text info description for the task, given the predicted initial state and the task planning parameters.

Parameters

- `[in] initial_state`: The predicted initial state for the task
- `[in] parameters`: The task planning parameters

std::size_t **pickup_waypoint**() **const**

Get the pickup waypoint in this request.

std::string **pickup_from_dispenser**() **const**

Get the name of the dispenser that we're picking up from.

rmf_traffic::Duration **pickup_wait**() **const**

Get the duration over which delivery items are loaded.

std::size_t **dropoff_waypoint**() **const**

Get the dropoff waypoint in this request.

std::string **dropoff_to_ingestor**() **const**

Get the name of the ingestor that we're dropping off to.

rmf_traffic::Duration **dropoff_wait**() **const**

Get the duration over which delivery items are unloaded.

const *Payload* &**payload**() **const**

Get the payload that is being delivered.

Public Static Functions

static *Task*::*ConstDescriptionPtr* **make**(std::size_t *pickup_waypoint*, rmf_traffic::Duration *pickup_duration*, std::size_t *dropoff_waypoint*, rmf_traffic::Duration *dropoff_duration*, *Payload* *payload*, std::string *pickup_from_dispenser* = "", std::string *dropoff_to_ingestor* = "")

Generate the description for this request.

Class Delivery::Description

- Defined in file_latest_rmf_task_include_rmf_task_requests_Delivery.hpp

Nested Relationships

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

Inheritance Relationships

Base Type

- `public rmf_task::Task::Description` (*Class Task::Description*)

Class Documentation

class `rmf_task::requests::Delivery::Description` : **public** rmf_task::*Task*::*Description*

Public Types

using Start = rmf_traffic::agv::Planner::Start

Public Functions

virtual *Task*::*ConstModelPtr* **make_model**(rmf_traffic::Time *earliest_start_time*, **const** *Parameters* &*parameters*) **const final**

Generate a Model for the task based on the unique traits of this description

Parameters

- `[in] earliest_start_time`: The earliest time this task should begin execution. This is usually the requested start time for the task.
- `[in] parameters`: The parameters that describe this AGV

virtual Info **generate_info**(**const** *State* &*initial_state*, **const** *Parameters* &*parameters*) **const final**

Generate a plain text info description for the task, given the predicted initial state and the task planning parameters.

Parameters

- `[in] initial_state`: The predicted initial state for the task
- `[in] parameters`: The task planning parameters

std::size_t **pickup_waypoint**() **const**

Get the pickup waypoint in this request.

std::string **pickup_from_dispenser**() **const**

Get the name of the dispenser that we're picking up from.

rmf_traffic::Duration **pickup_wait**() **const**
Get the duration over which delivery items are loaded.

std::size_t **dropoff_waypoint**() **const**
Get the dropoff waypoint in this request.

std::string **dropoff_to_ingestor**() **const**
Get the name of the ingestor that we're dropping off to.

rmf_traffic::Duration **dropoff_wait**() **const**
Get the duration over which delivery items are unloaded.

const *Payload* &**payload**() **const**
Get the payload that is being delivered.

Public Static Functions

static *Task*::*ConstDescriptionPtr* **make**(std::size_t *pickup_waypoint*, rmf_traffic::Duration *pickup_duration*, std::size_t *dropoff_waypoint*, rmf_traffic::Duration *dropoff_duration*, *Payload* *payload*, std::string *pickup_from_dispenser* = "", std::string *dropoff_to_ingestor* = "")
Generate the description for this request.

Class Loop

- Defined in file_latest_rmf_task_include_rmf_task_requests_Loop.hpp

Nested Relationships

Nested Types

- *Class Loop::Description*

Class Documentation

class `rmf_task::requests::Loop`
A class that generates a *Request* which requires an AGV to repeatedly travel between two locations

Public Static Functions

static *ConstRequestPtr* **make**(std::size_t *start_waypoint*, std::size_t *finish_waypoint*, std::size_t *num_loops*, **const** std::string &*id*, rmf_traffic::Time *earliest_start_time*, *ConstPriorityPtr* *priority* = nullptr, bool *automatic* = false)
Generate a loop request

Parameters

- `[in] start_waypoint`: The graph index for the starting waypoint of the loop
- `[in] finish_waypoint`: The graph index for the finishing waypoint of the loop

- `[in] num_loops`: The number of times the AGV should loop between the start_waypoint and finish_waypoint
- `[in] id`: A unique id for this request
- `[in] earliest_start_time`: The desired start time for this request
- `[in] priority`: The priority for this request
- `[in] automatic`: True if this request is auto-generated

class Description : **public** rmf_task::*Task*::*Description*

Public Functions

virtual *Task*::*ConstModelPtr* **make_model**(rmf_traffic::Time *earliest_start_time*, **const** *Parameters* &*parameters*) **const final**

Generate a Model for the task based on the unique traits of this description

Parameters

- `[in] earliest_start_time`: The earliest time this task should begin execution. This is usually the requested start time for the task.
- `[in] parameters`: The parameters that describe this AGV

virtual Info **generate_info**(**const** *State* &*initial_state*, **const** *Parameters* &*parameters*) **const final**

Generate a plain text info description for the task, given the predicted initial state and the task planning parameters.

Parameters

- `[in] initial_state`: The predicted initial state for the task
- `[in] parameters`: The task planning parameters

std::size_t **start_waypoint**() **const**

Get the start waypoint of the loop in this request.

std::size_t **finish_waypoint**() **const**

Get the finish waypoint of the loop in this request.

std::size_t **num_loops**() **const**

Get the numbert of loops in this request.

Public Static Functions

static *Task*::*ConstDescriptionPtr* **make**(std::size_t *start_waypoint*, std::size_t *finish_waypoint*, std::size_t *num_loops*)

Generate the description for this request.

Class Loop::Description

- Defined in file_latest_rmf_task_include_rmf_task_requests_Loop.hpp

Nested Relationships

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

Inheritance Relationships

Base Type

- `public rmf_task::Task::Description` (*Class Task::Description*)

Class Documentation

class `rmf_task::requests::Loop::Description` : **public** rmf_task::*Task*::*Description*

Public Functions

virtual *Task*::*ConstModelPtr* **make_model**(rmf_traffic::Time *earliest_start_time*, **const** *Parameters* &*parameters*) **const final**

Generate a Model for the task based on the unique traits of this description

Parameters

- `[in] earliest_start_time`: The earliest time this task should begin execution. This is usually the requested start time for the task.
- `[in] parameters`: The parameters that describe this AGV

virtual Info **generate_info**(**const** *State* &*initial_state*, **const** *Parameters* &*parameters*) **const final**

Generate a plain text info description for the task, given the predicted initial state and the task planning parameters.

Parameters

- `[in] initial_state`: The predicted initial state for the task
- `[in] parameters`: The task planning parameters

std::size_t **start_waypoint**() **const**

Get the start waypoint of the loop in this request.

std::size_t **finish_waypoint**() **const**

Get the finish waypoint of the loop in this request.

std::size_t **num_loops**() **const**

Get the numbert of loops in this request.

Public Static Functions

static *Task*::*ConstDescriptionPtr* **make**(std::size_t *start_waypoint*, std::size_t *finish_waypoint*, std::size_t *num_loops*)

Generate the description for this request.

Class ParkRobotFactory

- Defined in file_latest_rmf_task_include_rmf_task_requests_ParkRobotFactory.hpp

Inheritance Relationships

Base Type

- `public rmf_task::RequestFactory` (*Class RequestFactory*)

Class Documentation

class `rmf_task::requests::`**ParkRobotFactory** : **public** rmf_task::*RequestFactory*

The *ParkRobotFactory* will generate a request for the AGV to return to its desginated parking spot and remain idle there. This factory may be used when AGVs should not remain idle at the location of their last task but rather wait for new orders at their designated parking spots.

Public Functions

ParkRobotFactory(std::optional<std::size_t> *parking_waypoint* = std::nullopt)

Constructor

Parameters

- `[in] parking_waypoint`: The graph index of the waypoint assigned to this AGV for parking. If nullopt, the AGV will return to its charging_waypoint and remain idle there. It will not wait for its battery to charge up before undertaking new tasks.

virtual *ConstRequestPtr* **make_request**(**const** *State* &*state*) **const final**

Documentation inherited.

Class State

- Defined in file_latest_rmf_task_include_rmf_task_State.hpp

Inheritance Relationships

Base Type

- `public rmf_task::CompositeData` (*Class CompositeData*)

Class Documentation

class `rmf_task::State` : **public** rmf_task::*CompositeData*

Public Functions

RMF_TASK_DEFINE_COMPONENT(std::size_t, CurrentWaypoint)
The current waypoint of the robot state.

std::optional<std::size_t> **waypoint**() **const**

State &**waypoint**(std::size_t *new_waypoint*)

RMF_TASK_DEFINE_COMPONENT(double, CurrentOrientation)
The current orientation of the robot state.

std::optional<double> **orientation**() **const**

State &**orientation**(double *new_orientation*)

RMF_TASK_DEFINE_COMPONENT(rmf_traffic::Time, CurrentTime)
The current time for the robot.

std::optional<rmf_traffic::Time> **time**() **const**

State &**time**(rmf_traffic::Time *new_time*)

RMF_TASK_DEFINE_COMPONENT(std::size_t, DedicatedChargingPoint)
The dedicated charging point for this robot.

std::optional<std::size_t> **dedicated_charging_waypoint**() **const**

State &**dedicated_charging_waypoint**(std::size_t *new_charging_waypoint*)

RMF_TASK_DEFINE_COMPONENT(double, CurrentBatterySoC)
The current battery state of charge of the robot. This value is between 0.0 and 1.0.

std::optional<double> **battery_soc**() **const**

State &**battery_soc**(double *new_battery_soc*)

State &**load_basic**(**const** rmf_traffic::agv::Plan::Start &*location*, std::size_t *charging_point*, double *battery_soc*)
Load the basic state components expected for the planner.

Parameters

- `[in] location`: The robot's initial location data.
- `[in] charging_point`: The robot's dedicated charging point.
- `[in] battery_soc`: The robot's initial battery state of charge.

State &**load**(**const** rmf_traffic::agv::Plan::Start &*location*)
Load the plan start into the *State*. The location info will be split into CurrentWaypoint, CurrentOrientation, and CurrentTime data.

std::optional<rmf_traffic::agv::Plan::Start> **project_plan_start**(double *default_orientation* = 0.0, rmf_traffic::Time *default_time* = rmf_traffic::Time()) **const**
Project an rmf_traffic::agv::Plan::Start from this *State*.

If CurrentWaypoint is unavailable, this will return a std::nullopt. For any other components that are unavailable (CurrentOrientation or CurrentTime), the given default values will be used.

Parameters

- `[in] default_orientation`: The orientation value that will be used if CurrentOrientation is not available in this *State*.
- `[in] default_time`: The time value that will be used if default_time is not available in this *State*.

std::optional<rmf_traffic::agv::Plan::Start> **extract_plan_start**() **const**
Extract an rmf_traffic::agv::Plan::Start from this *State*.

If any necessary component is missing (i.e. CurrentWaypoint, CurrentOrientation, or CurrentTime) then this will return a std::nullopt.

Class Task

- Defined in file_latest_rmf_task_include_rmf_task_Task.hpp

Nested Relationships

Nested Types

- *Class Task::Active*
- *Class Task::Booking*
- *Class Task::Description*
- *Struct Description::Info*
- *Class Task::Model*
- *Class Task::Tag*

Class Documentation

class `rmf_task::`**Task**
Pure abstract interface for an executable *Task*.

Public Types

using ConstBookingPtr = std::shared_ptr<**const** *Booking*>

using ConstTagPtr = std::shared_ptr<**const** *Tag*>

using ConstModelPtr = std::shared_ptr<**const** *Model*>

using ConstDescriptionPtr = std::shared_ptr<**const** *Description*>

using ActivePtr = std::shared_ptr<*Active*>

class Active

Public Types

using Backup = detail::*Backup*

Backup data for the task. The state of the task is represented by a string. The meaning and format of the string is up to the *Task* implementation to decide.

Each Backup is tagged with a sequence number. As the *Task* makes progress, it can issue new Backups with higher sequence numbers. Only the Backup with the highest sequence number will be kept.

using Resume = detail::*Resume*

The Resume class keeps track of when the *Task* is allowed to Resume. You can either call the Resume object's operator() or let the object expire to tell the *Task* that it may resume.

Public Functions

virtual *Event*::*Status* **status_overview**() **const** = 0

Get a quick overview status of how the task is going.

virtual bool **finished**() **const** = 0

Check if this task is finished, which could include successful completion or cancellation.

virtual const std::vector<*Phase*::*ConstCompletedPtr*> &**completed_phases**() **const** = 0

Descriptions of the phases that have been completed.

virtual *Phase*::*ConstActivePtr* **active_phase**() **const** = 0

Interface for the phase that is currently active.

virtual std::optional<rmf_traffic::Time> **active_phase_start_time**() **const** = 0

Time that the current active phase started.

virtual const std::vector<*Phase*::*Pending*> &**pending_phases**() **const** = 0

Descriptions of the phases that are expected in the future.

virtual const *ConstTagPtr* &**tag**() **const** = 0

The tag of this *Task*.

virtual rmf_traffic::Duration **estimate_remaining_time**() **const** = 0

Estimate the overall finishing time of the task.

virtual *Backup* **backup**() **const** = 0

Get a backup for this *Task*.

virtual *Resume* **interrupt**(std::function<void()> *task_is_interrupted* = 0Tell this *Task* that it needs to be interrupted. An interruption means the robot may be commanded to do other tasks before this task resumes.

Interruptions may occur to allow operators to take manual control of the robot, or to engage automatic behaviors in response to emergencies, e.g. fire alarms or code blues.

Return an object to inform the *Task* when it is allowed to resume.
Parameters
- `[in] task_is_interrupted`: This callback will be triggered when the *Task* has reached a state where it is okay to start issuing other commands to the robot.

virtual void **cancel**() = 0

Tell the *Task* that it has been canceled. The behavior that follows a cancellation will vary between different Tasks, but generally it means that the robot should no longer try to complete its *Task* and should instead try to return itself to an unencumbered state as quickly as possible.

The *Task* may continue to perform some phases after being canceled. The pending_phases are likely to change after the *Task* is canceled, being replaced with phases that will help to relieve the robot so it can return to an unencumbered state.

The *Task* should continue to be tracked as normal. When its finished callback is triggered, the cancellation is complete.

virtual void **kill**() = 0

Kill this *Task*. The behavior that follows a kill will vary between different Tasks, but generally it means that the robot should be returned to a safe idle state as soon as possible, even if it remains encumbered by something related to this *Task*.

The *Task* should continue to be tracked as normal. When its finished callback is triggered, the killing is complete.

The *kill()* command supersedes the *cancel()* command. Calling *cancel()* after calling *kill()* will have no effect.

virtual void **skip**(uint64_t *phase_id*, bool *value* = true) = 0

Skip a specific phase within the task. This can be issued by operators if manual intervention is needed to unblock a task.

If a pending phase is specified, that phase will be skipped when the *Task* reaches it.

Parameters
- `[in] phase_id`: The ID of the phase that should be skipped.
- `[in] value`: True if the phase should be skipped, false otherwise.

virtual void **rewind**(uint64_t *phase_id*) = 0

Rewind the *Task* to a specific phase. This can be issued by operators if a phase did not actually go as intended and needs to be repeated.

It is possible that the *Task* will rewind further back than the specified phase_id if the specified phase depends on an earlier one. This is up to the discretion of the *Task* implementation.

virtual ~Active() = default

Protected Static Functions

static *Resume* **make_resumer**(std::function<void)
> *callback*Used by classes that inherit the *Task* interface to create a Resumer object

Parameters
- `[in] callback`: Provide the callback that should be triggered when the *Task* is allowed to resume

class Booking
Basic information about how the task was booked, e.g. what its name is, when it should start, and what its priority is.

Public Functions

Booking(std::string *id_*, rmf_traffic::Time *earliest_start_time_*, *ConstPriorityPtr* *priority_*, bool *automatic_* = false)
Constructor

Parameters
- `[in] id_`: The identity of the booking
- `[in] earliest_start_time_`: The earliest time that the task may begin
- `[in] priority_`: The priority of the booking
- `[in] automatic_`: Whether this booking was automatically generated

const std::string &**id**() **const**
The unique id for this booking.

rmf_traffic::Time **earliest_start_time**() **const**
Get the earliest time that this booking may begin.

ConstPriorityPtr **priority**() **const**
Get the priority of this booking.

bool **automatic**() **const**

class Description
An abstract interface to define the specifics of this task. This implemented description will differentiate this task from others.

Subclassed by *rmf_task::requests::ChargeBattery::Description*, *rmf_task::requests::Clean::Description*, *rmf_task::requests::Delivery::Description*, *rmf_task::requests::Loop::Description*

Public Functions

virtual *ConstModelPtr* **make_model**(rmf_traffic::Time *earliest_start_time*, **const** *Parameters* &*parameters*) **const** = 0
Generate a *Model* for the task based on the unique traits of this description

Parameters
- `[in] earliest_start_time`: The earliest time this task should begin execution. This is usually the requested start time for the task.
- `[in] parameters`: The parameters that describe this AGV

virtual *Info* **generate_info**(**const** *State* &*initial_state*, **const** *Parameters* &*parameters*) **const** = 0

Generate a plain text info description for the task, given the predicted initial state and the task planning parameters.

Parameters

- `[in] initial_state`: The predicted initial state for the task
- `[in] parameters`: The task planning parameters

virtual ~Description() = default

struct Info

Public Members

std::string **category**

std::string **detail**

class Model

An abstract interface for computing the estimate and invariant durations of this request

Public Functions

virtual std::optional<*Estimate*> **estimate_finish**(**const** *State* &*initial_state*, **const** *Constraints* &*task_planning_constraints*, **const** *TravelEstimator* &*travel_estimator*) **const** = 0

Estimate the state of the robot when the task is finished along with the time the robot has to wait before commencing the task

virtual rmf_traffic::Duration **invariant_duration**() **const** = 0

Estimate the invariant component of the task's duration.

virtual ~Model() = default

class Tag

Basic static information about the task.

Public Functions

Tag(*ConstBookingPtr booking_*, *Header header_*)

Constructor.

const *ConstBookingPtr* &**booking**() **const**

The booking information of the request that this *Task* is carrying out.

const *Header* &**header**() **const**

The header for this *Task*.

Class Task::Active

- Defined in file_latest_rmf_task_include_rmf_task_Task.hpp

Nested Relationships

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

Class Documentation

class rmf_task::*Task*::**Active**

Public Types

using Backup = detail::*Backup*
Backup data for the task. The state of the task is represented by a string. The meaning and format of the string is up to the *Task* implementation to decide.

Each Backup is tagged with a sequence number. As the *Task* makes progress, it can issue new Backups with higher sequence numbers. Only the Backup with the highest sequence number will be kept.

using Resume = detail::*Resume*
The Resume class keeps track of when the *Task* is allowed to Resume. You can either call the Resume object's operator() or let the object expire to tell the *Task* that it may resume.

Public Functions

virtual *Event*::*Status* **status_overview**() **const** = 0
Get a quick overview status of how the task is going.

virtual bool **finished**() **const** = 0
Check if this task is finished, which could include successful completion or cancellation.

virtual const std::vector<*Phase*::*ConstCompletedPtr*> &**completed_phases**() **const** = 0
Descriptions of the phases that have been completed.

virtual *Phase*::*ConstActivePtr* **active_phase**() **const** = 0
Interface for the phase that is currently active.

virtual std::optional<rmf_traffic::Time> **active_phase_start_time**() **const** = 0
Time that the current active phase started.

virtual const std::vector<*Phase*::*Pending*> &**pending_phases**() **const** = 0
Descriptions of the phases that are expected in the future.

virtual const *ConstTagPtr* &**tag**() **const** = 0
The tag of this *Task*.

virtual rmf_traffic::Duration **estimate_remaining_time**() **const** = 0
Estimate the overall finishing time of the task.

virtual *Backup* **backup**() **const** = 0
Get a backup for this *Task*.

virtual *Resume* **interrupt**(std::function<void)
> *task_is_interrupted* = 0Tell this *Task* that it needs to be interrupted. An interruption means the robot may be commanded to do other tasks before this task resumes.

Interruptions may occur to allow operators to take manual control of the robot, or to engage automatic behaviors in response to emergencies, e.g. fire alarms or code blues.

Return an object to inform the *Task* when it is allowed to resume.

Parameters

- `[in] task_is_interrupted`: This callback will be triggered when the *Task* has reached a state where it is okay to start issuing other commands to the robot.

virtual void **cancel**() = 0
Tell the *Task* that it has been canceled. The behavior that follows a cancellation will vary between different Tasks, but generally it means that the robot should no longer try to complete its *Task* and should instead try to return itself to an unencumbered state as quickly as possible.

The *Task* may continue to perform some phases after being canceled. The pending_phases are likely to change after the *Task* is canceled, being replaced with phases that will help to relieve the robot so it can return to an unencumbered state.

The *Task* should continue to be tracked as normal. When its finished callback is triggered, the cancellation is complete.

virtual void **kill**() = 0
Kill this *Task*. The behavior that follows a kill will vary between different Tasks, but generally it means that the robot should be returned to a safe idle state as soon as possible, even if it remains encumbered by something related to this *Task*.

The *Task* should continue to be tracked as normal. When its finished callback is triggered, the killing is complete.

The *kill()* command supersedes the *cancel()* command. Calling *cancel()* after calling *kill()* will have no effect.

virtual void **skip**(uint64_t *phase_id*, bool *value* = true) = 0
Skip a specific phase within the task. This can be issued by operators if manual intervention is needed to unblock a task.

If a pending phase is specified, that phase will be skipped when the *Task* reaches it.

Parameters

- `[in] phase_id`: The ID of the phase that should be skipped.
- `[in] value`: True if the phase should be skipped, false otherwise.

virtual void **rewind**(uint64_t *phase_id*) = 0
Rewind the *Task* to a specific phase. This can be issued by operators if a phase did not actually go as intended and needs to be repeated.

It is possible that the *Task* will rewind further back than the specified phase_id if the specified phase depends on an earlier one. This is up to the discretion of the *Task* implementation.

virtual ~Active() = default

Protected Static Functions

static *Resume* **make_resumer**(std::function<void)
> *callback*Used by classes that inherit the *Task* interface to create a Resumer object

Parameters

- `[in] callback`: Provide the callback that should be triggered when the *Task* is allowed to resume

Class Task::Booking

- Defined in file_latest_rmf_task_include_rmf_task_Task.hpp

Nested Relationships

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

Class Documentation

class `rmf_task::`*Task*`::Booking`
Basic information about how the task was booked, e.g. what its name is, when it should start, and what its priority is.

Public Functions

Booking(std::string *id_*, rmf_traffic::Time *earliest_start_time_*, *ConstPriorityPtr* *priority_*, bool *automatic_* = false)
Constructor

Parameters

- `[in] id_`: The identity of the booking
- `[in] earliest_start_time_`: The earliest time that the task may begin
- `[in] priority_`: The priority of the booking
- `[in] automatic_`: Whether this booking was automatically generated

const std::string &**id**() **const**
The unique id for this booking.

rmf_traffic::Time **earliest_start_time**() **const**
Get the earliest time that this booking may begin.

ConstPriorityPtr **priority**() **const**
Get the priority of this booking.

bool **automatic**() **const**

Class Task::Description

- Defined in file_latest_rmf_task_include_rmf_task_Task.hpp

Nested Relationships

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

Nested Types

- *Struct Description::Info*

Inheritance Relationships

Derived Types

- `public rmf_task::requests::ChargeBattery::Description` (*Class ChargeBattery::Description*)
- `public rmf_task::requests::Clean::Description` (*Class Clean::Description*)
- `public rmf_task::requests::Delivery::Description` (*Class Delivery::Description*)
- `public rmf_task::requests::Loop::Description` (*Class Loop::Description*)

Class Documentation

class `rmf_task::Task::Description`

An abstract interface to define the specifics of this task. This implemented description will differentiate this task from others.

Subclassed by *rmf_task::requests::ChargeBattery::Description*, *rmf_task::requests::Clean::Description*, *rmf_task::requests::Delivery::Description*, *rmf_task::requests::Loop::Description*

Public Functions

virtual *ConstModelPtr* **make_model**(rmf_traffic::Time *earliest_start_time*, **const** *Parameters* &*parameters*) **const** = 0

Generate a *Model* for the task based on the unique traits of this description

Parameters

- `[in] earliest_start_time`: The earliest time this task should begin execution. This is usually the requested start time for the task.
- `[in] parameters`: The parameters that describe this AGV

virtual *Info* **generate_info**(**const** *State* &*initial_state*, **const** *Parameters* &*parameters*) **const** = 0

Generate a plain text info description for the task, given the predicted initial state and the task planning parameters.

Parameters

- `[in] initial_state`: The predicted initial state for the task
- `[in] parameters`: The task planning parameters

virtual ~Description() = default

struct Info

Public Members

std::string **category**

std::string **detail**

Class Task::Model

- Defined in file_latest_rmf_task_include_rmf_task_Task.hpp

Nested Relationships

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

Class Documentation

class rmf_task::*Task*::**Model**

An abstract interface for computing the estimate and invariant durations of this request

Public Functions

virtual std::optional<*Estimate*> **estimate_finish**(**const** *State* *&initial_state*, **const** *Constraints* *&task_planning_constraints*, **const** *TravelEstimator* *&travel_estimator*) **const** = 0

Estimate the state of the robot when the task is finished along with the time the robot has to wait before commencing the task

virtual rmf_traffic::Duration **invariant_duration**() **const** = 0

Estimate the invariant component of the task's duration.

virtual ~Model() = default

Class Task::Tag

- Defined in file_latest_rmf_task_include_rmf_task_Task.hpp

Nested Relationships

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

Class Documentation

class rmf_task::*Task*::**Tag**
Basic static information about the task.

Public Functions

Tag(*ConstBookingPtr* *booking_*, *Header* *header_*)
Constructor.

const *ConstBookingPtr* &**booking**() **const**
The booking information of the request that this *Task* is carrying out.

const *Header* &**header**() **const**
The header for this *Task*.

Class TaskPlanner

- Defined in file_latest_rmf_task_include_rmf_task_TaskPlanner.hpp

Nested Relationships

Nested Types

- *Class TaskPlanner::Assignment*
- *Class TaskPlanner::Configuration*
- *Class TaskPlanner::Options*

Class Documentation

class rmf_task::**TaskPlanner**

Public Types

enum TaskPlannerError
Values:

enumerator low_battery
None of the agents in the initial states have sufficient initial charge to even head back to their charging stations. Manual intervention is needed to recharge one or more agents.

enumerator limited_capacity
None of the agents in the initial states have sufficient battery capacity to accommodate one or more requests. This may be remedied by increasing the battery capacity or by lowering the threshold_soc in the state configs of the agents or by modifying the original request.

using Assignments = std::vector<std::vector<*Assignment*>>
Container for assignments for each agent.

using Result = std::variant<*Assignments*, *TaskPlannerError*>

Public Functions

TaskPlanner(*Configuration configuration*, *Options default_options*)
Constructor

Parameters

- `[in] configuration`: The configuration for the planner
- `[in] default_options`: Default options for the task planner to use when solving for assignments. These options can be overriden each time a plan is requested.

const *Configuration* &**configuration**() **const**
Get a const reference to configuration of this task planner.

const *Options* &**default_options**() **const**
Get a const reference to the default planning options.

Options &**default_options**()
Get a mutable reference to the default planning options.

Result **plan**(rmf_traffic::Time *time_now*, std::vector<*State*> *agents*, std::vector<*ConstRequestPtr*> *requests*)
Generate assignments for requests among available agents. The default *Options* of this *TaskPlanner* instance will be used.

Parameters

- `[in] time_now`: The current time when this plan is requested
- `[in] agents`: The initial states of the agents/AGVs that can undertake the requests
- `[in] requests`: The set of requests that need to be assigned among the agents/AGVs

Result **plan**(rmf_traffic::Time *time_now*, std::vector<*State*> *agents*, std::vector<*ConstRequestPtr*> *requests*, *Options options*)
Generate assignments for requests among available agents. Override the default parameters

Parameters

- `[in] time_now`: The current time when this plan is requested
- `[in] agents`: The initial states of the agents/AGVs that can undertake the requests
- `[in] requests`: The set of requests that need to be assigned among the agents/AGVs
- `[in] options`: The options to use for this plan. This overrides the default *Options* of the *TaskPlanner* instance

double **compute_cost**(**const** *Assignments* &*assignments*) **const**
Compute the cost of a set of assignments.

class Assignment

Public Functions

Assignment(rmf_task::*ConstRequestPtr* *request*, *State* *finish_state*, rmf_traffic::Time *deployment_time*)
Constructor

Parameters
- `[in] request`: The task request for this assignment
- `[in] state`: The state of the agent at the end of the assigned task
- `[in] earliest_start_time`: The earliest time the agent will begin exececuting this task

const rmf_task::*ConstRequestPtr* &**request**() **const**

const *State* &**finish_state**() **const**

const rmf_traffic::Time **deployment_time**() **const**

class Configuration
The *Configuration* class contains planning parameters that are immutable for each *TaskPlanner* instance and should not change in between plans.

Public Functions

Configuration(*Parameters* *parameters*, *Constraints* *constraints*, *ConstCostCalculatorPtr* *cost_calculator*)
Constructor

Parameters
- `[in] parameters`: The parameters that describe the agents
- `[in] constraints`: The constraints that apply to the agents
- `[in] cost_calculator`: An object that tells the planner how to calculate cost

const *Parameters* &**parameters**() **const**
Get the parameters that describe the agents.

Configuration &**parameters**(*Parameters* *parameters*)
Set the parameters that describe the agents.

const *Constraints* &**constraints**() **const**
Get the constraints that are applicable to the agents.

Configuration &**constraints**(*Constraints* *constraints*)
Set the constraints that are applicable to the agents.

const *ConstCostCalculatorPtr* &**cost_calculator**() **const**
Get the CostCalculator.

Configuration &**cost_calculator**(*ConstCostCalculatorPtr* *cost_calculator*)
Set the CostCalculator. If a nullptr is passed, the BinaryPriorityCostCalculator is used by the planner.

class Options
The *Options* class contains planning parameters that can change between each planning attempt.

Public Functions

Options(bool *greedy*, std::function<bool)
> *interrupter* = nullptr*ConstRequestFactoryPtr finishing_request* = nullptrConstructor

Parameters
- `[in] greedy`: If true, a greedy approach will be used to solve for the task assignments. Optimality is not guaranteed but the solution time may be faster. If false, an A* based approach will be used within the planner which guarantees optimality but may take longer to solve.
- `[in] interrupter`: A function that can determine whether the planning should be interrupted.
- `[in] finishing_request`: A request factory that generates a tailored task for each agent/AGV to perform at the end of their assignments

Options &**greedy**(bool *value*)
Set whether a greedy approach should be used.

bool **greedy**() **const**
Get whether a greedy approach will be used.

Options &**interrupter**(std::function<bool)
> *interrupter*Set an interrupter callback that will indicate to the planner if it should stop trying to plan

const std::function<bool()> &**interrupter**
constGet the interrupter that will be used in this *Options*.

Options &**finishing_request**(*ConstRequestFactoryPtr finishing_request*)
Set the request factory that will generate a finishing task.

ConstRequestFactoryPtr **finishing_request**() **const**
Get the request factory that will generate a finishing task.

Class TaskPlanner::Assignment

- Defined in file_latest_rmf_task_include_rmf_task_TaskPlanner.hpp

Nested Relationships

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

Class Documentation

class rmf_task::*TaskPlanner*::**Assignment**

Public Functions

Assignment(rmf_task::*ConstRequestPtr* *request*, *State* *finish_state*, rmf_traffic::Time *deployment_time*)
Constructor

Parameters

- `[in] request`: The task request for this assignment
- `[in] state`: The state of the agent at the end of the assigned task
- `[in] earliest_start_time`: The earliest time the agent will begin exececuting this task

const rmf_task::*ConstRequestPtr* &**request**() **const**

const *State* &**finish_state**() **const**

const rmf_traffic::Time **deployment_time**() **const**

Class TaskPlanner::Configuration

- Defined in file_latest_rmf_task_include_rmf_task_TaskPlanner.hpp

Nested Relationships

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

Class Documentation

class rmf_task::*TaskPlanner*::**Configuration**
The *Configuration* class contains planning parameters that are immutable for each *TaskPlanner* instance and should not change in between plans.

Public Functions

Configuration(*Parameters* *parameters*, *Constraints* *constraints*, *ConstCostCalculatorPtr* *cost_calculator*)
Constructor

Parameters

- `[in] parameters`: The parameters that describe the agents
- `[in] constraints`: The constraints that apply to the agents
- `[in] cost_calculator`: An object that tells the planner how to calculate cost

const *Parameters* &**parameters**() **const**
Get the parameters that describe the agents.

Configuration &**parameters**(*Parameters* *parameters*)
Set the parameters that describe the agents.

const *Constraints* &**constraints**() **const**
Get the constraints that are applicable to the agents.

Configuration &**constraints**(*Constraints constraints*)
Set the constraints that are applicable to the agents.

const *ConstCostCalculatorPtr* &**cost_calculator**() **const**
Get the CostCalculator.

Configuration &**cost_calculator**(*ConstCostCalculatorPtr cost_calculator*)
Set the CostCalculator. If a nullptr is passed, the BinaryPriorityCostCalculator is used by the planner.

Class TaskPlanner::Options

- Defined in file_latest_rmf_task_include_rmf_task_TaskPlanner.hpp

Nested Relationships

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

Class Documentation

class rmf_task::*TaskPlanner*::**Options**
The *Options* class contains planning parameters that can change between each planning attempt.

Public Functions

Options(bool *greedy*, std::function<bool)
> *interrupter* = nullptr*ConstRequestFactoryPtr finishing_request* = nullptrConstructor

Parameters

- `[in] greedy`: If true, a greedy approach will be used to solve for the task assignments. Optimality is not guaranteed but the solution time may be faster. If false, an A* based approach will be used within the planner which guarantees optimality but may take longer to solve.
- `[in] interrupter`: A function that can determine whether the planning should be interrupted.
- `[in] finishing_request`: A request factory that generates a tailored task for each agent/AGV to perform at the end of their assignments

Options &**greedy**(bool *value*)
Set whether a greedy approach should be used.

bool **greedy**() **const**
Get whether a greedy approach will be used.

Options &**interrupter**(std::function<bool)
> *interrupter*Set an interrupter callback that will indicate to the planner if it should stop trying to plan

const std::function<bool()> &**interrupter**
constGet the interrupter that will be used in this *Options*.

Options &**finishing_request**(*ConstRequestFactoryPtr finishing_request*)
Set the request factory that will generate a finishing task.

ConstRequestFactoryPtr **finishing_request**() **const**
Get the request factory that will generate a finishing task.

Class TravelEstimator

- Defined in file_latest_rmf_task_include_rmf_task_Estimate.hpp

Nested Relationships

Nested Types

- *Class TravelEstimator::Result*

Class Documentation

class rmf_task::**TravelEstimator**
A class to estimate the cost of travelling between any two points in a navigation graph. Results will be memoized for efficiency.

Public Functions

TravelEstimator(**const** *Parameters* &*parameters*)
Constructor

Parameters

- `[in] parameters`: The parameters for the robot

std::optional<*Result*> **estimate**(**const** rmf_traffic::agv::Plan::Start &*start*, **const** rmf_traffic::agv::Plan::Goal &*goal*) **const**
Estimate the cost of travelling.

class Result
The result of a travel estimation.

Public Functions

rmf_traffic::Duration **duration**() **const**
How long the travelling will take.

double **change_in_charge**() **const**
How much the battery will drain while travelling.

Class TravelEstimator::Result

- Defined in file_latest_rmf_task_include_rmf_task_Estimate.hpp

Nested Relationships

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

Class Documentation

class rmf_task::*TravelEstimator*::**Result**
The result of a travel estimation.

Public Functions

rmf_traffic::Duration **duration**() **const**
How long the travelling will take.

double **change_in_charge**() **const**
How much the battery will drain while travelling.

Class VersionedString

- Defined in file_latest_rmf_task_include_rmf_task_VersionedString.hpp

Nested Relationships

Nested Types

- *Class VersionedString::Reader*
- *Class VersionedString::View*

Class Documentation

class rmf_task::**VersionedString**

Public Functions

VersionedString(std::string *initial_value*)
Construct a versioned string

Parameters

- `[in] initial_value`: The initial value of this versioned string

void **update**(std::string *new_value*)
Update the value of this versioned string

Parameters

- `[in] new_value`: The new value for this versioned string

View **view**() **const**
Get a view of the current version of the string.

class Reader

Public Functions

Reader()
Construct a *Reader*.

std::shared_ptr<**const** std::string> **read**(**const** *View* &*view*)
Read from the *View*.

If this *Reader* has never seen this *View* before, then this function will return a reference to the string that the *View* contains. Otherwise, if this *Reader* has seen this *View* before, then this function will return a nullptr.

Parameters

- `[in] view`: The view that the *Reader* should look at

class View
A snapshot view of a *VersionedString*. This is thread-safe to read even while the *VersionedString* is being modified. Each *VersionedString::Reader* instance will only view this object once; after the first viewing it will return a nullptr.

The contents of this *View* can only be retrieved by a *VersionedString::Reader*

Class VersionedString::Reader

- Defined in file_latest_rmf_task_include_rmf_task_VersionedString.hpp

Nested Relationships

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

Class Documentation

class rmf_task::*VersionedString*::**Reader**

Public Functions

Reader()
Construct a *Reader*.

std::shared_ptr<**const** std::string> **read**(**const** *View* &*view*)
Read from the *View*.

If this *Reader* has never seen this *View* before, then this function will return a reference to the string that the *View* contains. Otherwise, if this *Reader* has seen this *View* before, then this function will return a nullptr.

Parameters

- `[in] view`: The view that the *Reader* should look at

Class VersionedString::View

- Defined in file_latest_rmf_task_include_rmf_task_VersionedString.hpp

Nested Relationships

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

Class Documentation

class View
A snapshot view of a *VersionedString*. This is thread-safe to read even while the *VersionedString* is being modified. Each *VersionedString::Reader* instance will only view this object once; after the first viewing it will return a nullptr.

The contents of this *View* can only be retrieved by a *VersionedString::Reader*

1.3.3 Functions

Template Function rmf_task::detail::insertion_cast

- Defined in file_latest_rmf_task_include_rmf_task_detail_impl_CompositeData.hpp

Function Documentation

template<typename **T**>
CompositeData::*InsertResult*<*T*> `rmf_task::detail::`**insertion_cast**(*CompositeData*::*InsertResult*<std::any> *result*)

Function rmf_task::standard_waypoint_name

- Defined in file_latest_rmf_task_include_rmf_task_Header.hpp

Function Documentation

std::string rmf_task::**standard_waypoint_name**(**const** rmf_traffic::agv::Graph &*graph*, std::size_t *waypoint*)

1.3.4 Defines

Define RMF_TASK_DEFINE_COMPONENT

- Defined in file_latest_rmf_task_include_rmf_task_CompositeData.hpp

Define Documentation

RMF_TASK_DEFINE_COMPONENT(*Type*, *Name*)

Define a component class that is convenient to use in a CompositeData instance. The defined class will contain only one field whose type is specified by Type. The name of the class will be Name.

1.3.5 Typedefs

Typedef rmf_task::ActivatorPtr

- Defined in file_latest_rmf_task_include_rmf_task_Activator.hpp

Typedef Documentation

using rmf_task::**ActivatorPtr** = std::shared_ptr<*Activator*>

Typedef rmf_task::ConstActivatorPtr

- Defined in file_latest_rmf_task_include_rmf_task_Activator.hpp

Typedef Documentation

using rmf_task::**ConstActivatorPtr** = std::shared_ptr<**const** *Activator*>

Typedef rmf_task::ConstCostCalculatorPtr

- Defined in file_latest_rmf_task_include_rmf_task_CostCalculator.hpp

Typedef Documentation

using rmf_task::**ConstCostCalculatorPtr** = std::shared_ptr<**const** CostCalculator>

Typedef rmf_task::ConstLogPtr

- Defined in file_latest_rmf_task_include_rmf_task_Log.hpp

Typedef Documentation

using rmf_task::**ConstLogPtr** = std::shared_ptr<**const** *Log*>

Typedef rmf_task::ConstParametersPtr

- Defined in file_latest_rmf_task_include_rmf_task_Parameters.hpp

Typedef Documentation

using rmf_task::**ConstParametersPtr** = std::shared_ptr<**const** *Parameters*>

Typedef rmf_task::ConstPriorityPtr

- Defined in file_latest_rmf_task_include_rmf_task_Priority.hpp

Typedef Documentation

using rmf_task::**ConstPriorityPtr** = std::shared_ptr<**const** Priority>

Typedef rmf_task::ConstRequestFactoryPtr

- Defined in file_latest_rmf_task_include_rmf_task_RequestFactory.hpp

Typedef Documentation

using rmf_task::**ConstRequestFactoryPtr** = std::shared_ptr<**const** *RequestFactory*>

Typedef rmf_task::ConstRequestPtr

- Defined in file_latest_rmf_task_include_rmf_task_Request.hpp

Typedef Documentation

using rmf_task::**ConstRequestPtr** = std::shared_ptr<**const** *Request*>

Typedef rmf_task::ConstTravelEstimatorPtr

- Defined in file_latest_rmf_task_include_rmf_task_Estimate.hpp

Typedef Documentation

using rmf_task::**ConstTravelEstimatorPtr** = std::shared_ptr<**const** *TravelEstimator*>

Typedef rmf_task::CostCalculatorPtr

- Defined in file_latest_rmf_task_include_rmf_task_CostCalculator.hpp

Typedef Documentation

using rmf_task::**CostCalculatorPtr** = std::shared_ptr<CostCalculator>

Typedef rmf_task::events::SimpleEventStatePtr

- Defined in file_latest_rmf_task_include_rmf_task_events_SimpleEventState.hpp

Typedef Documentation

using rmf_task::events::**SimpleEventStatePtr** = std::shared_ptr<*SimpleEventState*>

Typedef rmf_task::PriorityPtr

- Defined in file_latest_rmf_task_include_rmf_task_Priority.hpp

Typedef Documentation

using rmf_task::**PriorityPtr** = std::shared_ptr<Priority>

Typedef rmf_task::RequestFactoryPtr

- Defined in file_latest_rmf_task_include_rmf_task_RequestFactory.hpp

Typedef Documentation

using rmf_task::**RequestFactoryPtr** = std::shared_ptr<*RequestFactory*>

Typedef rmf_task::RequestPtr

- Defined in file_latest_rmf_task_include_rmf_task_Request.hpp

Typedef Documentation

using rmf_task::**RequestPtr** = std::shared_ptr<*Request*>

INDEX

R