OpenRMF

Release 1.0.0

Open Source Robotics Corporation

Jun 09, 2021

CONTENTS

CHAPTER

ONE

OVERVIEW

1.1 Getting started

1.1.1 Installation

Building *rmf_core* from source requires *gcc* version 8 or higher, or *clang* version 6 or higher.

```
mkdir ws_rmf/src -p
cd ws_rmf/src/
git clone https://github.com/osrf/rmf_core.git
cd ../
source /opt/ros/eloquent/setup.bash
rosdep update
rosdep install --from-paths src --ignore-src -yr
colcon build --cmake-args -DCMAKE_BUILD_TYPE=RELEASE
```

To manually override the compiler version, prefix the *colcon* command with the *CXX* parameter.

```
sudo apt update && sudo apt install g++-8
CXX=g++-8 colcon build --cmake-args -DCMAKE_BUILD_TYPE=RELEASE
```

1.1.2 Demonstrations

This repository holds a number of demonstrations and examples of working with *rmf_core* and the other packages in the RMF ecosystem.

1.2 Interfacing with OpenRMF

There are several interface points with RMF core, as shown in the arrows between the blue central box and the orange boxes in the diagram above. The goal of these interfaces is to create a “narrow” and simple set of messages that allow *rmf_core* to integrate with the following elements of a deployment.

1.2.1 Robot fleet integration

The *rmf_fleet_msgs* package contains four messages and is intended to carry the interactions between *rmf_core* and a vendor-provided (typically proprietary) fleet manager for a collection of robots. It is expected that a RMF deployment will consist of multiple robot fleets, often operating at different levels of RMF integration. For example, one fleet may only be willing to supply *FleetState* messages (observation-only) whereas another fleet in the same facility may be willing to follow RMF *DestinationRequest* messages. RMF is specifically designed to allow this type of "mixed levels of control" and to plan accordingly.

- *rmf_fleet_msgs/FleetState* on topic *fleet_states*. This message consists of a list of *rmf_fleet_msgs/RobotState* messages, each of which contains the state of a particular robot. This includes the level of the facility the robot is on, its X- and Y- offset (in meters) from the origin of that level's map, its current destination and path (if known), and so on.
- *rmf_fleet_msgs/DestinationRequest* on topic *destination_requests* is a request for a particular robot to go to a particular destination.
- *rmf_fleet_msgs/ModeRequest* on topic *mode_requests* is a request for a particular robot to change modes, for example, from *MOVING* to *PAUSED*, in order to preserve spatial separation between robots of different fleets.
- *rmf_fleet_msgs/PathRequest* on topic *path_requests* is a request for a particular robot to follow a particular path.

As mentioned above, several levels of integration are possible between RMF and vendor-controlled Fleet Managers. The following table captures the required messages for each integration feature. In general, the more integration features that are available for a particular fleet, the more efficient the combined system operations will be, because each integration feature gives additional options for *rmf_core* to perform traffic management. For example, a fleet that only supports the "state reporting" integration feature will always require that *rmf_core* totally clear its predicted travel lane of all other robots, whereas fleets that support "pause/resume motion" or "complete paths" allow many other potential options for de-conflicting robot traffic.

Integration Feature	Required Message	Default topic name
state reporting	*rmf_fleet_msgs/FleetState*	*fleet_states*
set destinations	*rmf_fleet_msgs/DestinationRequest*	*destination_requests*
pause/resume motion	*rmf_fleet_msgs/ModeRequest*	*mode_requests*
set complete paths	*rmf_fleet_msgs/PathRequest*	*path_requests*

1.2.2 Door integration

The *rmf_door_msgs* package contains two messages. This interface allows RMF to open and close motorized doors for robots as they move throughout a facility.

- *rmf_door_msgs/DoorState* messages are periodically sent by door controllers to *rmf_core*. These messages express the current mode of the door as *CLOSED*, *MOVING*, or *OPEN*
- *rmf_door_msgs/DoorRequest* messages are sent from *rmf_core* to doors when they need to open or close for robot operations.

1.3 OpenRMF modules

OpenRMF consists of several modules/libraries that together provide the combined feature set of OpenRMF. Each module provides usage and API documentation at the pages linked below.

- rmf_battery
- rmf_ros2
- rmf_simulation
- rmf_task
- rmf_traffic
- rmf_utils
- rmf_visualization

1.4 Contributing to OpenRMF

Contributions via pull requests to the various RMF-related repositories are welcome.

In general, we expect contributors to follow the ROS 2 contributing guidelines, with the exception of code style (see *Coding style guide*, below).

1.4.1 Coding style guide

The RMF code adopts slightly different guidelines from the ROS 2 style guide. The objective of this section is to highlight the divergences of RMF code style from the ROS 2 style guide for C++.

C++

Standard

- The *eloquent* release uses C++14
- The *foxy* release uses C++17

Style

Line Length

Maximum line length is 80 characters.

File Naming and Extensions

- Original files with public interfaces should follow *CamelCase* convention
- Header files should use the *.hpp* extension and must include header guards
- Implementation files should use the *.cpp* extension.

Braces

- Use open braces without indenting the braces for function, class, and struct definitions and on *if*, *else*, *while*, *for*, etc.
- Cuddle braces on namespace definitions only

Indentations

- Two-space indentation per level
- Two-space continued indentation for function definition/call parameters

Variable Naming

- All variable names should use *snake_case*
- Private member variables should be prefixed with an underscore _

Function and Method Naming

- All function names including class member functions should use *snake_case*

Classes

- Class names should always use *CamelCase*.
- Privacy specifiers (*public*, *private*, *protected*) should not be indented.
- Only member functions are allowed in *public* scope in public APIs (no *public* data members).
- Two-space indentation for other class statements.
- Leading colon between a constructor and its member initialization list.
- No space/indent for constructor initialization lists.
- Trailing commas between members in the initialization lists.
- Do not use *struct* in public APIs. Usage in internal implementation is allowed.
- Abstract interface classes should contain only pure abstract member functions. No data fields or function implementations are allowed.

Namespaces

- Cuddle brace for namespaces with a space between the name and the opening brace.
- No indentation for namespace contents (including nested namespaces).
- Closing brace to be followed with a comment with the namespace name, e.g.:

```
namespace A {

/* code */

} // namespace A
```

Pointer and Reference Syntax Alignment

Left-align * and &

Comments and Doc Comments

Use /// and /** */ comments for documentation purposes and // style comments for notes and general comments

Examples

Description of a class

```
namespace A {
namespace B {
class UpperCamelCase
{
public:

  UpperCamelCase(Foo foo, Bar bar)
  : _foo(foo),
    _bar(bar),
    _baz(Baz(foo, bar))
  {
  }

  /// This is an example function
  ///
  /// And here is a longer description blah blah blah
  ///
  /// \param[in] in_value
  ///   It takes in a value
  ///
  /// \param[out] out_value
  ///   It puts out a value
  ///
  /// \return some result
  ResultType snake_case_member_functions(
```

(continues on next page)

(continued from previous page)

```
    InputValue in_value,
    OutputValue& out_value) const;

private:
  /* ... etc ... */
};

} // namespace B
} // namespace A
```

A class that is only used internally.

```
class ImplementationClass
{
public:

  /// documentation
  double snake_case_public_members_uncommon;

  void foo(Bar bar);

private:

  int _lead_with_underscore;

};
```

A class that defines an interface.

```
class AbstractInterfaceClass
{
public:

  /// Only pure abstract member functions.
  /// No data fields or implemented
  ///functions
  virtual ReturnType pure_virtual_function(
    SomeArgType arg1,
    SomeArgType arg2) = 0;

};
```

RMF Linter

Most of these styles and restrictions can be checked with the *ament_uncrustify* command line tool using this configuration file.

Example usage.

```
cd workspace/
wget https://raw.githubusercontent.com/osrf/rmf_core/master/rmf_utils/test/format/rmf_
↪code_style.cfg
```

(continues on next page)

(continued from previous page)

```
source /opt/ros/foxy/setup.bash
ament_uncrustify -c rmf_code_style.cfg .
```

The *–reformat* option may be passed into the *ament_uncrustify* call to apply the changes in place. However, this is recommended only after manually reviewing the changes.

1.5 Getting support

We provide several mechanism for getting help. Whether you have run into a problem using OpenRMF, or you just want more information about some aspect of it not covered in the documentation, one of the below options should have you covered. Using the correct resource, particularly the interactive forums, will ensure you get a helpful response to your query quickly.

1.5.1 Documentation

User guide

The primary documentation for OpenRMF is the Programming Multiple Robots with ROS 2 book. This documentation should be your first stop when learning how to use OpenRMF, or how a particular module works.

Developer's guide

If you are interested in contributing a bug fix, a new feature, or an entire new module to OpenRMF, stop by the Developer's Guide first. Complying with the Developer's Guide will ensure that your pull request gets merged faster.

1.5.2 OpenRMF Discussions

When you have a problem that you cannot solve yourself using the documentation, your next stop should be the OpenRMF Discussions website. This is an Question and Answer site for OpenRMF.

When using OpenRMF Discussions, it is important to act as a responsible member of the OpenRMF community. Follow the ROS guidelines about support, especially the section on etiquette.

Most importantly, before asking your question **conduct a search for another question with the same problem**. It is possible that someone else has already found the answer, so conducting a search may get you a solution to your problem much faster than asking a question and waiting for someone to respond.

If you don't find an answer to your problem by searching, then ask a new question explaining what is wrong in as much detail as you can. Please provide **as much as possible** of the following information. The more information you provide, the sooner you will get a useful response.

- Post the complete output for error messages, starting from the command that you ran. If the output is long, use a service such as GitHub Gists and link that from your post. Copy the text using copy-and-paste, do not re-type it by hand.
- Do not post screenshots of text. Post the text itself. This aides people in searching for your error and makes it more likely that they will help you.
- If you are having a problem with a GUI tool, post a screenshot or movie of the tool showing the problem.

- Describe the environment in which you are running the software in as much detail as possible. Provide the names and versions of packages you are using, your platform/OS and version, hardware used, your compiler tool chain and version (if relevant), how you installed OpenRMF, etc.
- Provide a complete and detailed set of steps to reproduce the problem. If someone can reproduce your problem, they are more likely to find a way to fix it quickly.
- If you are following a tutorial, provide a link to it and state where in the tutorial your problem occurs.
- Make the topic of your post or bug report or feature request detailed. A topic that says "Problem making it go" won't attract many people to help you.
- If you include code snippets, use the formatting function to make sure they appear correctly. Check the preview before you post.
- When you have a problem, Short, Self Contained, Correct (Compilable) Examples or Minimal, Complete, and Verifiable Examples help us reproduce your error quickly and thus get help to you quickly.
- Describe what you have done already to try and fix the problem or to find its cause. Understanding what you have already tried will ensure others do not waste time asking if you have tried something and allow them to more rapidly identify the cause of your problem.

1.5.3 Bug reports

If your problem is actually a bug in OpenRMF, then the issue tracker for the relevant repository should be your next port of call. If you do not know which repository is relevant (although hopefully discussion of your problem on the OpenRMF discussion board has identified that), then post an issue on the general OpenRMF repository and a developer will help you shift it to the correct place.

When posting a bug report, it is important to be as detailed and accurate as possible. The bug report template will help you fill in the necessary information, but in general all the same advice that applies to the Discourse discussion board also applies to filing a bug report. You should also post a debug trace if possible, to identify where in the source code the problem occurs.

When giving your bug report a title, do *not* include something in the title that indicates it is a bug report, such as starting the title with *[Bug]* or *Bug report:*. Instead, use the *Bug* issue label to indicate that it is a bug report.

Do not post requests for help on the issue trackers. They will be closed and you will be directed to the correct place to ask, slowing down the process of you getting help.

1.5.4 Feature requests

If OpenRMF doesn't do something you need or want, please file a feature request. We want OpenRMF to cover as many use cases as possible, but if we don't know about your necessary features, we can't put them on the roadmap.

When making a feature request, it helps to know as much about what you want. A feature request along the lines of "I want it to control my robot" will likely be ignored, but a feature request that begins "Add support for robot fleets from Company" and provides detailed use cases is likely to attract attention.

When giving your feature request a title, do *not* include something in the title that indicates it is a feature request, such as starting the title with *[Feature]* or *Feature request:*. Instead, use the *Enhancement* issue label to indicate that it is a bug report.

1.5.5 General ROS-related support

If you have problems or queries that are related to ROS in general rather than specific to OpenRMF, please see the ROS support page for how to get help.

1.5.6 How not to get help

Please don't post support requests that are not related to OpenRMF, or are about general programming problems. There are more appropriate venues for those, such as Stack Overflow.

Do not contact the developers directly. Using the correct channels means everyone can see your question, and you are more likely to get a response. If you contact a few developers directly, that is a lot less eyes on your problem (and a lot more annoyed busy developers), and it also means that others cannot see the solution to your problem.

Do not post a request for help that just says "It doesn't work." No one will know how to help you.

1.6 Frequently Asked Questions

1.6.1 What is the difference between RoMi-H and rmf_core, in high level?

RoMi-H is an umbrella term for a wide range of open specifications and software tools that aim to ease the integration and interoperability of robotic systems, building infrastructure, smart medical devices, and user interfaces with a focus on the healthcare sector.

rmf_core is a collection of repositories and software libraries for an implementation of some of the core systems that compose RoMi-H.

1.6.2 Why is RoMi-H spread across multiple GitHub accounts and repositories?

The development of RoMi-H is a collaborative research and development effort. Several different organizations are involved in its development, and there is not yet a fixed protocol for where and how the constituent parts of RoMi-H will be deposited. The organization of these packages may converge as the project continues to progress.

1.6.3 In an actual deployment, is RoMi-H something that is deployed into a server or a robot?

RoMi-H is a collection of open specifications and software tools. Some of the specifications and software that falls under the RoMi-H umbrella might run on robots, but for the most part it will be used as an intermediary to communicate and negotiate between standalone systems. Since robots are usually deployed with their own proprietary fleet managers, in most cases we expect RoMi-H to communicate with a fleet manager instead of running directly on a robot. However users will be provided with RoMi-H software tools that can run directly on a robot to assist in cases where a robot platform does not come with its own fleet manager.

In short, some parts of RoMi-H may run on servers, some on desktops, some on mobile devices, and some on robot platforms.

1.6.4 Does RoMi-H support high availability (e.g failover cluster)? Can it be load-balanced?

The systems within RoMi-H are heterogeneous. Some systems are fully distributed, so there is no master that would be a failure point. Other systems do require certain centralized services, and those services are being designed to have failovers as well as ways to distribute their load, e.g. by using mirror servers.

1.6.5 Does RoMi-H run on ROS nodes?

Certain components in RoMi-H are being implemented using ROS2. For non-ROS2 systems, we are working on various options for bridging between different middlewares. Much of that effort is concentrated in the SOSS project.

1.6.6 What is SOSS?

Please refer to the SOSS GitHub page. SOSS is a plugin-based framework for performing simple translations between pub/sub middlewares. Since we expect RoMi-H to bridge many systems that are already running their own middlewares, we are providing SOSS as one option for integrating a middleware into RoMi-H.

1.6.7 Are there design guidelines for integrating with RoMi-H?

RoMi-H is still in research and development, moving towards production deployment, and many of the APIs and specifications are under active development. Design and integration guidelines will be coming out as the core APIs solidify.

1.6.8 If I want a CI/CD pipeline to build my custom RoMi-H components, is there already a template or docker image to help with this?

There is an ongoing effort to provide this, but it is not ready for public consumption yet.

1.6.9 Is RoMi-H production-ready?

RoMi-H is still in research and development, but we are aggressively moving towards deployability. We aim to have the APIs stable and key features implemented by mid-2020.

1.6.10 Is there python version of RoMi-H?

We intend to provide Python bindings for the core APIs of RoMi-H, especially for robot fleet management. This work has not yet begun, but it should be straightforward once the C++ APIs have stabilized.

1.6.11 Is RoMi-H constrained to a particular DDS?

Just like how ROS2 is not constrained to any particular DDS, neither is RoMi-H. The choice of which DDS implementation to use will be determined by the system integrators who deploy a RoMi-H system in a given facility.

1.6.12 How do we specify the map layouts of a building, and tie together multiple floors for that building?

Our tool for managing map layouts is available at here. Specifying multiple floors for a building is a feature that should be finished in the very near future.

1.6.13 How can we specify the schedule of a fleet?

The API for specifying robot traffic schedules is undergoing enormous changes right now. A preliminary version of it already exists, but I do not recommend familiarizing yourself with it, because it will be completely different very soon.

1.6.14 Which distribution of ROS2 is compatible with RoMi-H for production purpose?

Currently rmf_core requires ROS2 eloquent for certain launch file features. In general, we are likely to be using the latest release of ROS2 while doing research and development on RoMi-H.

1.6.15 How does *rmf_traffic* avoid mobile robot traffic conflicts?

When we are done implementing the traffic management solution, we will be doing a more extensive write-up on the conflict avoidance and negotiation methods than what can reasonably fit in an FAQ, but here is a quick outline of the methodology. There are two levels to traffic deconfliction: (1) prevention, and (2) resolution.

1. **Prevention.** Whenever possible, it would be good to prevent traffic conflicts from happening in the first place. To facilitate this, we have implemented a platform agnostic Traffic Schedule Database. The traffic schedule is a living database whose contents will change over time to reflect delays, cancelations, or route changes. All fleet managers that are integrated into RoMi-H must report the expected itineraries of their vehicles to the traffic schedule. With the information available on the schedule, compliant fleet managers can plan routes for their vehicles that avoid conflicts with any other vehicles (no matter which fleet they belong to). *rmf_traffic* provides a Planner class to help facilitate this for vehicles that behave like standard AGVs. In the future we intend to provide a similar utility for AMRs.

2. **Resolution.** It is not always possible to perfectly prevent traffic conflicts. Mobile robots may experience delays because of unanticipated obstacles in their environment, or the predicted schedule may be flawed for any number of reasons. In cases where a conflict does arise, *rmf_traffic* has a Negotiation scheme. When the Traffic Schedule Database detects an upcoming conflict between two or more schedule participants, it will send a conflict notice out to the relevant fleet managers, and a negotiation between the fleet managers will begin. Each fleet manager will submit its preferred itineraries, and each will respond with itineraries that can accommodate the others. A third-party judge (deployed by the system integrator) will choose the set of proposals that is considered preferable and notify the fleet managers about which itineraries they should follow.

1.6.16 Why is this traffic management system so complicated?

RoMi-H has a number of system design constraints that create unique challenges for traffic management. The core goal of RoMi-H is to facilitate system integration for heterogeneous mobile robot fleets that may be provided by different vendors and may have different technical capabilities.

1. Vendors tend to want to keep their computing systems independent from other vendors. Since vendors are often responsible for ensuring uptime and reliability on their computing infrastructure, they may view it as an unacceptable liability to share computing resources with another vendor. This means that the traffic management system must be able to function while being distributed across different machines on a network.
2. Different robot platforms may have different capabilities. Many valuable AGV platforms that are currently deployed are not able to change their itineraries dynamically. Some AGV platforms can change course when instructed to, as long as they stick to a predefined navigation graph. Some AMR platforms can dynamically navigate themselves around unanticipated obstacles in their environment. Since RoMi-H is meant to be an enabling technology, it is important that we design a system that can maximize the utility of all these different types of systems without placing detrimental constraints on any of them.

These considerations led to the current design of distributed conflict prevention and distributed schedule negotiation. There is plenty of space within the design to create simpler and more efficient subsets for categories of mobile robots that fit certain sets of requirements, but these optimizations can be added later, building on top of the existing completely generalized framework.

1.7 About and contact

1.7.1 About

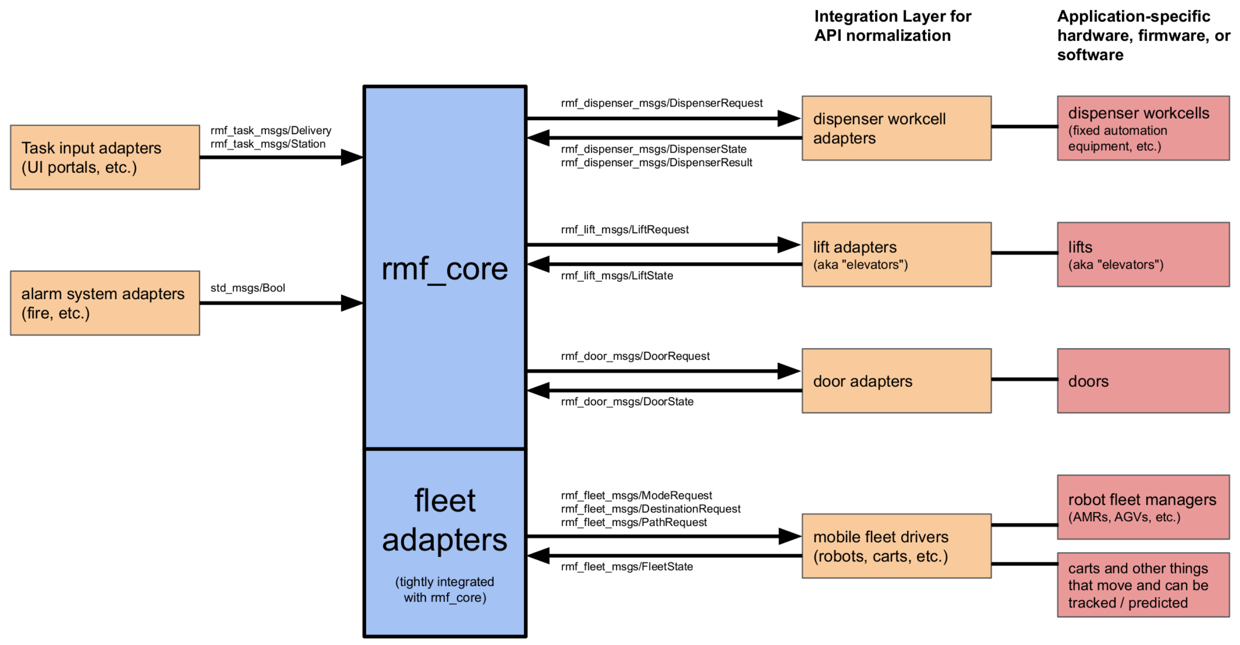
1.7.2 Contact

To contact the developers of OpenRMF, please use the OpenRMF Discussions website. If you need help, please see *Getting support* for more information on how to get it.

The core *rmf* packages provide the centralized functions of the Open Robotics Middleware Framework (OpenRMF). These include task queuing, conflict-free resource scheduling, utilities to help create robot fleet adapters, and so on.

OpenRMF is built on ROS 2. However direct use of ROS 2 is not required to use Open-RMF.

To create a useful deployment, the core of *rmf* must be connected to many other subsystems, as shown in the following diagram.


Integration Layer for API normalization
Application-specific hardware, firmware, or software
Task input adapters (UI portals, etc.)
rmf_task_msgs/Delivery
rmf_task_msgs/Station
alarm system adapters (fire, etc.)
std_msgs/Bool
rmf_core
fleet adapters
(tightly integrated with rmf_core)
rmf_dispenser_msgs/DispenserRequest
rmf_dispenser_msgs/DispenserState
rmf_dispenser_msgs/DispenserResult
dispenser workcell adapters
dispenser workcells (fixed automation equipment, etc.)
rmf_lift_msgs/LiftRequest
rmf_lift_msgs/LiftState
lift adapters (aka "elevators")
lifts (aka "elevators")
rmf_door_msgs/DoorRequest
rmf_door_msgs/DoorState
door adapters
doors
rmf_fleet_msgs/ModeRequest
rmf_fleet_msgs/DestinationRequest
rmf_fleet_msgs/PathRequest
rmf_fleet_msgs/FleetState
mobile fleet drivers (robots, carts, etc.)
robot fleet managers (AMRs, AGVs, etc.)
carts and other things that move and can be tracked / predicted

1.8 Roadmap

A near-term roadmap of the entire RMF project (including and beyond *rmf_core*) can be found in the user manual here.