FOSDEM 2026

Introducing rclrs: the ROS 2 client library for Rust

Integrating Rust into the ROS 2 ecosystem

Esteve Fernández

About me

- Hi, I’m Esteve Fernández! 👋
- Member of the original ROS 2 team
- Former member of the ROS core team
- Member of the Apache Software Foundation
- Committer at many projects: Boost C++, Python Twisted
- Started the ros2-rust project in 2017

Rust programmers are the vegans of the developer community

Davide Faconti
ROSConES 2025

Introduction

- Rust for robotics
  - Advantages and disadvantages
- ros2-rust: Rust for ROS 2
  - Code generation for Rust (`rosidl_generator_rs`)
  - colcon integration (`colcon-cargo` and `colcon-ros-cargo`)
  - Client library (`rclrs`)


BUT WHY!?

Rust for robotics

- Advantages:
  - Fast: speed comparable to C
    - Rust in the Linux kernel
  - Reliable: compiler detects data races, concurrency issues and unsafe code
    - MISRA C guidelines “for free”
    - Fearless concurrency (e.g. Rayon)
  - Productive: modern tooling, including dependency management, cross-platform compiler
- Concerns:
  - Ecosystem not as mature as C and C++
  - Certification
    - Good news: Ferrocene compiler qualified at ISO 26262 (ASIL D), IEC 61508 (SIL) and IEC 62304

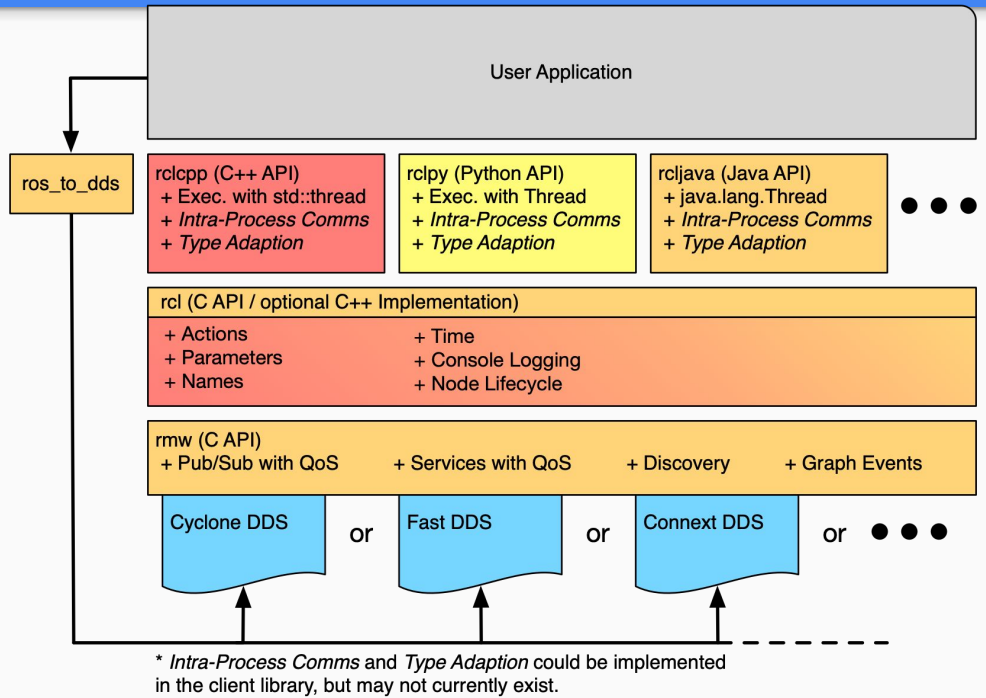

C++
WITH ROS
ROS2-RUST!!!
imgflip.com

ros2-rust: Rust for ROS 2

- Started in 2017
- Complete ROS 2 pipeline
  - Code generation for Rust (`rosidl_generator_rs`)
  - colcon integration (`colcon-cargo` and `colcon-ros-cargo`)
  - Client library (`rclrs`)
- Community driven
  - 51 contributors from J&J, Intrinsic, Clearpath and many others
- Used and developed by people from different organizations
  - e.g. Open-RMF

ROS 2 architecture

- Layered architecture
- Transport agnostic
- Each layer only interacts with its immediate layers
- Client libraries only have access to `rcl`



Tooling and ecosystem

- Rust
  - Build tool: `cargo`
  - Dependencies: crates.io
- ROS
  - Build tool: `colcon`
  - Dependencies: `rosdep`
- ros2-rust
  - Build tool: `colcon-cargo` and `colcon-ros-cargo` (akin to `colcon-cmake` and `colcon-ros-cmake`)
  - Dependencies: access to both crates.io and `rosdep`
    - Work is underway to support Cargo dependencies in the ROS build farm

Code generation

- `rosidl_generator_rs`
  - Released on the ROS build farm for Humble, Jazzy, Kilted and Rolling
  - ROS 2 Lyrical Luth (current Rolling) shipping it as one of the default generators
  - Integrated into the ROS 2 message generation pipeline
    - Support for `.msg` and `.idl` formats
    - No extra compile time cost
    - Seamless integration in any workspace

Rust client library (rclrs)

- Rust bindings for `rcl`
- Similar API to `rclcpp`
  - Rust idiomatic
  - Extra features (e.g. async workers)
- Written in Rust
- Released on crates.io
- Support for publishers, subscriptions, clients, services, timers, parameters and actions
- Support for zero copy pubsub

Key features of rclrs: Node management

Rust

```
let context = Context::default_from_env()?;
let executor = context.create_basic_executor();
let node = executor.create_node("my_node")?;
```

C++

```
rclcpp::init(argc, argv);
auto node =
  rclcpp::Node::make_shared("minimal_publisher");
rclcpp::executors::SingleThreadedExecutor executor;
executor.add_node(node);
```

Key features of rclrs: Publishers

```
let context = Context::default_from_env()?;
let executor = context.create_basic_executor();
let node = executor.create_node("minimal_publisher")?;
let publisher = node.create_publisher::<example_interfaces::msg::String>("topic")?;
let mut message = example_interfaces::msg::String::default();
let mut publish_count: u32 = 1;
while context.ok() {
    message.data = format!("Hello, world! {}", publish_count);
    println!("Publishing: [{}]", message.data);
    publisher.publish(&message)?;
    publish_count += 1;
    std::thread::sleep(std::time::Duration::from_millis(500));
}
```

Key features of rclrs: Subscriptions

```
let context = Context::default_from_env()?;
let mut executor = context.create_basic_executor();

let node = executor.create_node("minimal_subscriber")?;

let worker = node.create_worker::<usize>(0);
let _subscription = worker.create_subscription::<example_interfaces::msg::String, _>(
    "topic",
    move |num_messages: &mut usize, msg: example_interfaces::msg::String| {
        *num_messages += 1;
        println!("#{} | I heard: '{}'", *num_messages, msg.data);
    },
)?;

println!("Waiting for messages...");
executor.spin(SpinOptions::default()).first_error()?;
```

Key features of rclrs: Services

```rust
fn handle_service(request: AddTwoInts_Request, info: ServiceInfo) -> AddTwoInts_Response {
    let timestamp = info
        .received_timestamp
        .map(|t| format!(" at [{t:?}]"))
        .unwrap_or(String::new());
    println!("request{timestamp}: {} + {}", request.a, request.b);
    AddTwoInts_Response {
        sum: request.a + request.b,
    }
}
fn main() -> Result<(), Error> {
    let mut executor = Context::default_from_env()?.create_basic_executor();
    let node = executor.create_node("minimal_service")?;
    let _server = node.create_service::<AddTwoInts, _>("add_two_ints", handle_service)?;
    println!("Starting server");
    executor.spin(SpinOptions::default()).first_error()?;
    Ok(())
}
```

Key features of rclrs: Clients

```
let mut executor = Context::default_from_env()?.create_basic_executor();
let node = executor.create_node("minimal_client")?;
let client = node.create_client::<AddTwoInts>("add_two_ints")?;
let promise = executor.commands().run(async move {
    println!("Waiting for service...");
    client.notify_on_service_ready().await.unwrap();
    let request = AddTwoInts_Request { a: 41, b: 1 };
    println!("Waiting for response");
    let response: AddTwoInts_Response = client.call(&request).unwrap().await.unwrap();
    println!(
        "Result of {} + {} is: {}",
        request.a, request.b, response.sum,
    );
});

executor
    .spin(SpinOptions::new().until_promise_resolved(promise))
    .first_error()?;
```

Key features of rclrs: Async workers

```
let mut executor = Context::default_from_env()?.create_basic_executor();
let node = executor.create_node("worker_demo")?;
let publisher = node.create_publisher("output_topic")?;
let worker = node.create_worker(String::new());
let _subscription = worker.create_subscription(
    "input_topic",
    move |data: &mut String, msg: example_interfaces::msg::String| {
        *data = msg.data;
    },
)?;
std::thread::spawn(move || loop {
    std::thread::sleep(std::time::Duration::from_secs(1));
    let publisher = Arc::clone(&publisher);
    let _ = worker.run(move |data: &mut String| {
        let msg = example_interfaces::msg::String { data: data.clone() };
        publisher.publish(msg).unwrap();
    });
});
```


MORE FEATURES!!!!!!!!

Key features of rclrs: more

- Actions
- Dynamic publishers and subscribers
- Parameters
- Timers
- Zero copy pubsub - Loaned messages API
- `roslaunch` support
- Full async - Runtime agnostic (tokio, async_std, etc)
- tf2 https://github.com/olingo99/tf2_rs
- diagnostic_updater_rs https://github.com/romainreignier/diagnostic_updater_rs

Future

- `rclrs` on the ROS build farm
  - Work is being done for adding support for Cargo projects on the ROS build farm
- Support for ROS graph
- On-the-fly message generation
- Dynamic clients and services

Links

- GitHub - ros2-rust/ros2_rust: Rust bindings for ROS 2
- GitHub - PolySync/misra-rust: An investigation into what adhering to each MISRA-C rule looks like in Rust. The intention is to decipher how much we "get for free" from the Rust compiler.
- rayon-rs/rayon - A data parallelism library for Rust
- Rust for the Linux kernel
- OSRA's Technical Governance Committee Approves $250,000 Funding for Infrastructure and Documentation Enhancement

Thank you!

https://github.com/esteve

https://www.linkedin.com/in/estevefernandez

esteve@apache.org