

# Parameters Should Be Boring

*generate\_parameter\_library*

October 20, 2023

Tyler Weaver  
Staff Software Engineer  
tyler@picknik.ai

# Tyler Weaver



- Racing Kart Driver
- Movelt Maintainer
- Rust Evangelist
- Docker Skeptic



# RCLCPP

# Parameters

Part 1

# Getting Started



```
int main(int argc, char ** argv)
{
    rclcpp::init(argc, argv);

    auto node = std::make_shared<rclcpp::Node>("minimal_param_node");
    auto my_string = node->declare_parameter("my_string", "world");
    auto my_number = node->declare_parameter("my_number", 23);

    rclcpp::spin(node);
    rclcpp::shutdown();
}
```

# Parameter Struct



```
struct Params {
    std::string my_string = "world";
    int my_number = 23;
};

int main(int argc, char ** argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<rclcpp::Node>("minimal_param_node");
    auto params = Params{};
    params.my_string = node->declare_parameter("my_string", params.my_string);
    params.my_number = node->declare_parameter("my_number", params.my_number);

    rclcpp::spin(node);
    rclcpp::shutdown();
}
```

# ParameterDescriptor



```
int main(int argc, char ** argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<rclcpp::Node>("minimal_param_node");
    auto params = Params{};

    auto param_desc = rcl_interfaces::msg::ParameterDescriptor{};
    param_desc.description = "Mine!";
    param_desc.additional_constraints = "One of [world, base, home]";
    params.my_string = node->declare_parameter("my_string",
        params.my_string, param_desc);

    param_desc = rcl_interfaces::msg::ParameterDescriptor{};
    param_desc.description = "Who controls the universe?";
    param_desc.additional_constraints = "A multiple of 23";
    params.my_number = node->declare_parameter("my_number",
        params.my_number, param_desc);
    //...
```

```
auto const _ = node->add_on_set_parameters_callback(
    [] (std::vector<rclcpp::Parameter> const& params)
    {
        for (auto const& param : params) {
            if (param.get_name() == "my_string") {
                auto const value = param.get_value<std::string>();
                auto const valid = std::vector<std::string>{"world", "base", "home"};
                if (std::find(valid.cbegin(), valid.cend(), value) == valid.end()) {
                    auto result = rcl_interfaces::msg::SetParametersResult{};
                    result.successful = false;
                    result.reason = std::string("my_string: {")
                        .append(value)
                        .append("} not one of: [world, base, home]");
                    return result;
                }
            }
        }
        return rcl_interfaces::msg::SetParametersResult{};
    });
```



- parameter name: 6 separate copies
- declaration: re-init description for each parameter
- validation: convert vector to map



30 lines of C++ boilerplate per parameter

30 lines of C++ boilerplate per parameter  
Before handling of dynamic parameters

# generate\_ parameter\_library

Part 2

```
minimal_param_node:
  my_string: {
    type: string,
    description: "Mine!"
    validation: {
      one_of<>: ["world", "base", "home"]
    }
  }
  my_number: {
    type: int
    description: "Mine!"
    validation: {
      multiple_of_23: []
    }
  }
}
```

```
find_package(generate_parameter_library REQUIRED)

generate_parameter_library(
    minimal_param_node_parameters
    src/minimal_param_node.yaml
)

add_executable(minimal_node src/minimal_param_node.cpp)
target_link_libraries(minimal_node PRIVATE
    rclcpp::rclcpp
    minimal_param_node_parameters
)
```

```
#include <rclcpp/rclcpp.hpp>
#include "minimal_param_node_parameters.hpp"

int main(int argc, char * argv[])
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<rclcpp::Node>("minimal_param_node");
    auto param_listener =
        std::make_shared<minimal_param_node::ParamListener>(node);
    auto params = param_listener->get_params();

    // ...
}
```

# Built-In Validation Functions



- bounds (inclusive)
- less than
- greater than
- less than or equal
- greater than or equal



# Built-In Validation Functions



- bounds (inclusive)
- less than
- greater than
- less than or equal
- greater than or equal
- fixed string/array length
- size of string/array length greater than
- size of string/array length less than
- array contains no duplicates
- array is a subset of another array
- bounds checking for elements of an array

# Custom Validation



```
#include <rclcpp/rclcpp.hpp>
#include <fmt/core.h>
#include <tl_expected/expected.hpp>

tl::expected<void, std::string> multiple_of_23(
    rclcpp::Parameter const& parameter) {
    int param_value = parameter.as_int();
    if (param_value % 23 != 0) {
        return tl::make_unexpected(fmt::format(
            "Invalid value '{} for parameter {}. Must be multiple of 23.",
            param_value, parameter.get_name()));
    }
    return {};
}
```

- Dynamic parameters
- Generation of RCLPY Parameter Libraries
- Generation of Markdown Docs
- Examples and docs at  
[github.com/pickNikRobotics/generate\\_parameter\\_library](https://github.com/pickNikRobotics/generate_parameter_library)

# Boring?

Part 3

# Why so many parameters?



- Users use defaults for most parameters

# Why so many parameters?



- Users use defaults for most parameters
- Authors only test default values

# Why so many parameters?



- Users use defaults for most parameters
- Authors only test default values
- Permutations of parameters grow exponentially



# Why so many parameters?



- Users use defaults for most parameters
- Authors only test default values
- Permutations of parameters grow exponentially
- The more complex your interface the less useful your abstraction

# Why so many parameters?



- Users use defaults for most parameters
- Authors only test default values
- Permutations of parameters grow exponentially
- The more complex your interface the less useful your abstraction
- Resist the urge to expose interior details as parameters

# What is a good parameter?



- Express user intent (latency or throughput)

# What is a good parameter?



- Express user intent (latency or throughput)
- Details like buffer sizes scale with hardware

# What is a good parameter?



- Express user intent (latency or throughput)
- Details like buffer sizes scale with hardware
- Leave the door open to improvements in behavior for the user

Questions?

[github.com/pickNikRobotics/generate\\_parameter\\_library](https://github.com/pickNikRobotics/generate_parameter_library)