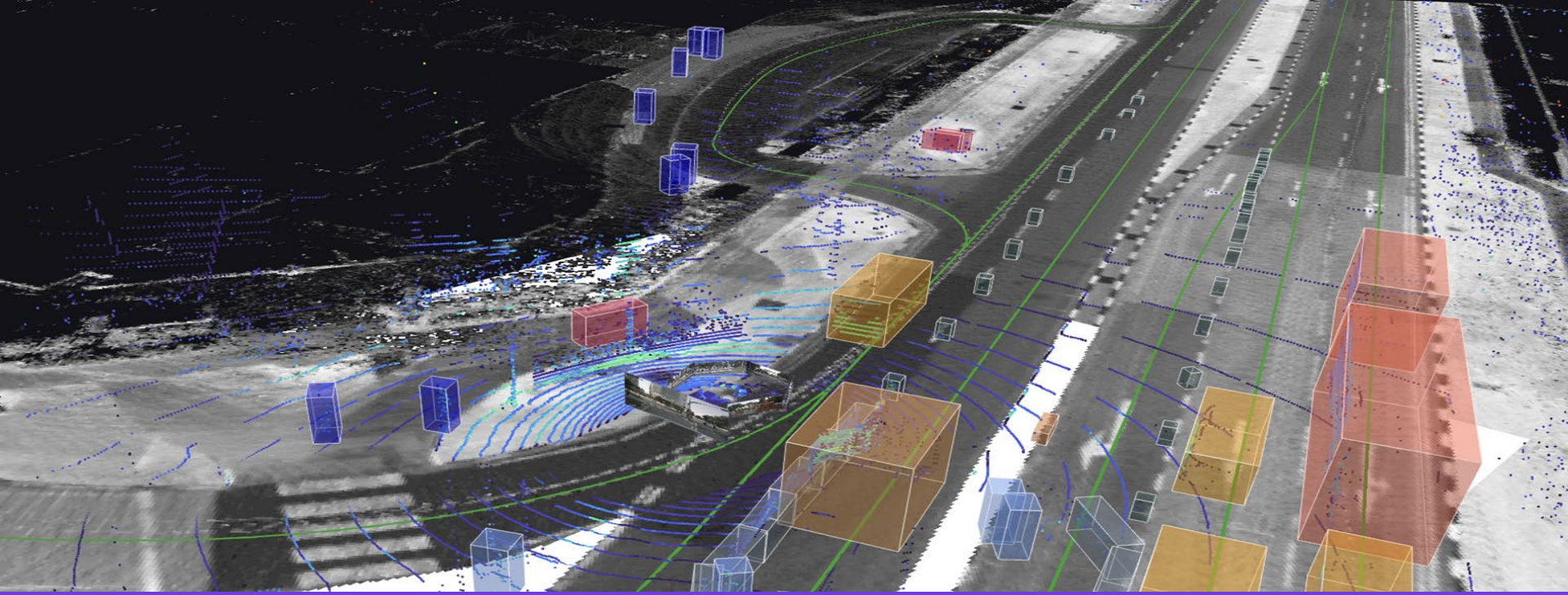
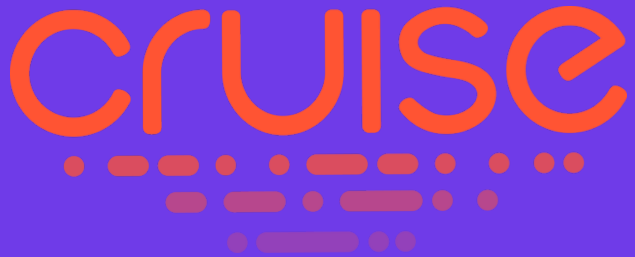


MCAP: A Next-Generation File Format for ROS Recording

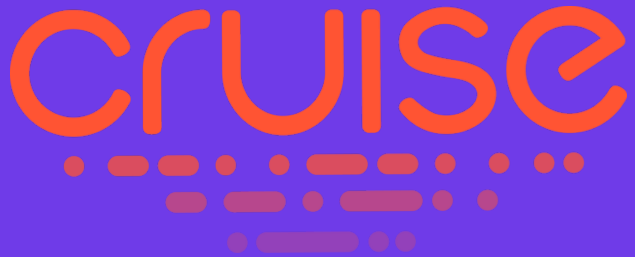
John Hurliman
john@foxglove.dev

foxglove

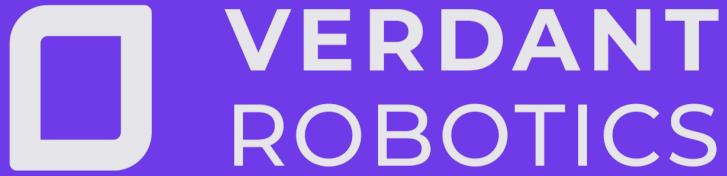

foxglove



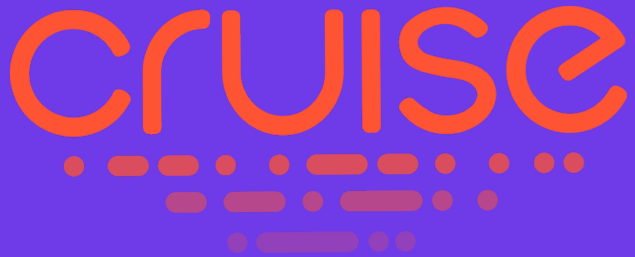
How can we make rosbag recordings more crash resilient and improve write throughput?



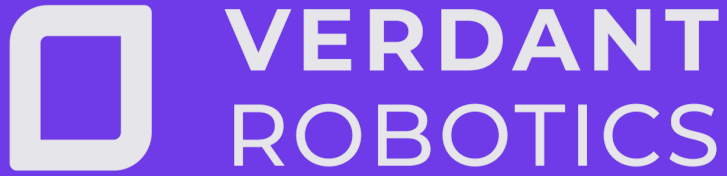
How can we make rosbag recordings more crash resilient and improve write throughput?



How can we leverage recording SDKs and bag tooling outside the ROS ecosystem?



How can we make rosbag recordings more crash resilient and improve write throughput?



How can we leverage recording SDKs and bag tooling outside the ROS ecosystem?

How can we make recordings self-contained and remotely streamable?

Design Goals

Design Goals

- Serialization Agnostic

Design Goals

- Serialization Agnostic
- Determinism

Design Goals

- Serialization Agnostic
- Determinism
- Self-Describing

Design Goals

- Serialization Agnostic
- Determinism
- Self-Describing
- Big and Small Data

Design Goals

- Serialization Agnostic
- Determinism
- Self-Describing
- Big and Small Data
- Write-Optimized

Design Goals

- Serialization Agnostic
- Determinism
- Self-Describing
- Big and Small Data
- Write-Optimized
- Dynamically Add New Data Streams

Design Goals

- Serialization Agnostic
- Determinism
- Self-Describing
- Big and Small Data
- Write-Optimized
- Dynamically Add New Data Streams
- Corruption Resilient

Design Goals

- Serialization Agnostic
- Determinism
- Self-Describing
- Big and Small Data
- Write-Optimized
- Dynamically Add New Data Streams
- Corruption Resilient
- Indexing

Design Goals

- Serialization Agnostic
- Determinism
- Self-Describing
- Big and Small Data
- Write-Optimized
- Dynamically Add New Data Streams
- Corruption Resilient
- Indexing
- Standards Compatible

Why a New File Format?

- We evaluated many existing formats:
  - rosbag1
  - rosbag2 (SQLite)
  - Length-delimited Protobuf
  - Avro
  - HDF5
  - Parquet
  - EBML

mcap.dev/motivation/evaluation-of-robotics-data-recording-file-formats


Introducing: MCAP

Heterogeneous data

- Store messages encoded in multiple serialization formats in a single file
- Can store ROS 1, ROS 2 (CDR), Protobuf, Flatbuffer, JSON, etc
- Include metadata and attachments

Performant writing

- Write-optimized for fast recording on resource-constrained robots
- Append-only structure
- Recover partially-written files when recording is interrupted
- Writes can be streamed across a network
- Optional compression

Efficient seeking

- Extract data without scanning or decompressing the entire file
- Fast access to indexed summary data
- Remote reading via HTTP Range requests

Self-contained files

- Embed all message schemas in the file
- No extra dependencies needed for decoding
- CRCs ensure data integrity per chunk

Design Checklist

- [x] Serialization Agnostic
- [x] Determinism
- [x] Self-Describing
- [x] Big and Small Data
- [x] Write-Optimized
- [x] Dynamically Add New Data Streams
- [x] Corruption Resilient
- [x] Indexing
- [x] Standards Compatible

Simple Specification

Overview

MCAP is a modular container file format for recording timestamped pub/sub messages with arbitrary serialization formats.

MCAP files are designed to work well under various workloads, resource constraints, and durability requirements.

A Kaitai Struct description for the MCAP format is provided at mcap.ksy.

File Structure

A valid MCAP file is structured as follows. The Summary and Summary Offset sections are optional.

```
<Magic><Header><Data section>[<Summary section>][<Summary Offset section>]<Footer><Magic>
```

The Data, Summary, and Summary Offset sections are structured as sequences of **records**:

```
[<record type><record content length><record><record type><record content length><record>...]
```

Files not conforming to this structure are considered malformed.

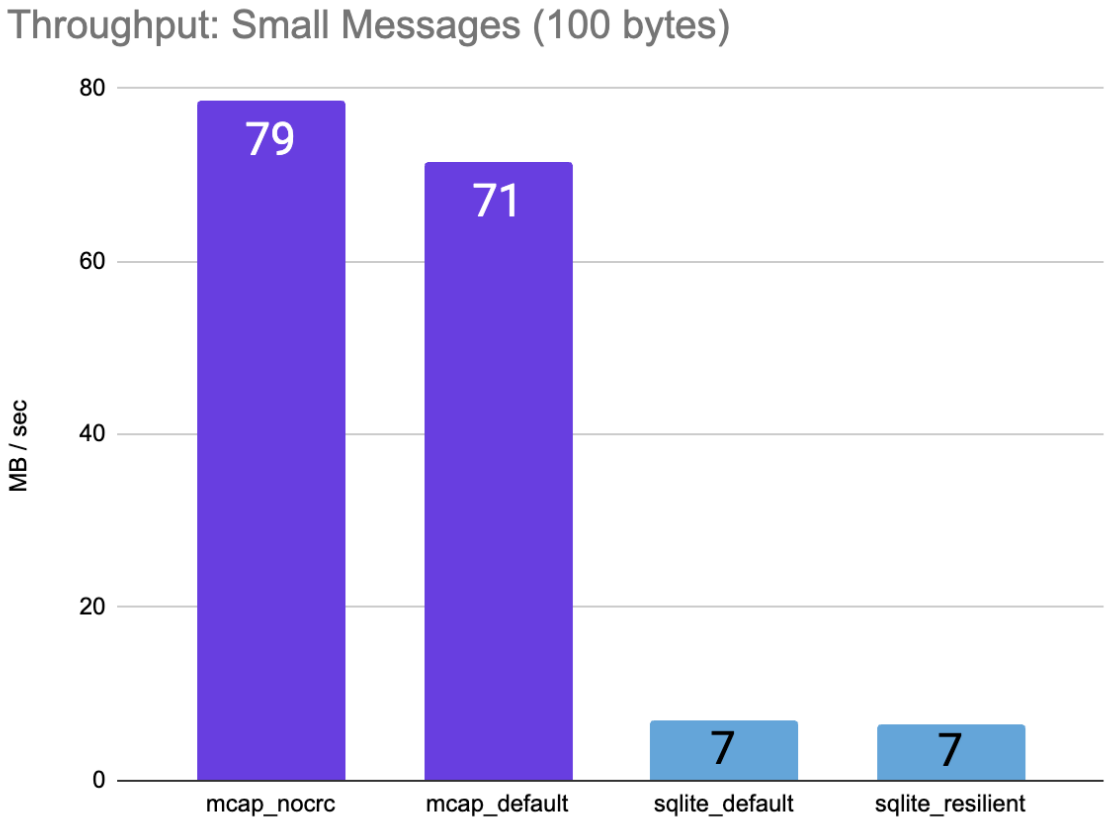
Magic

An MCAP file must begin and end with the following magic bytes:

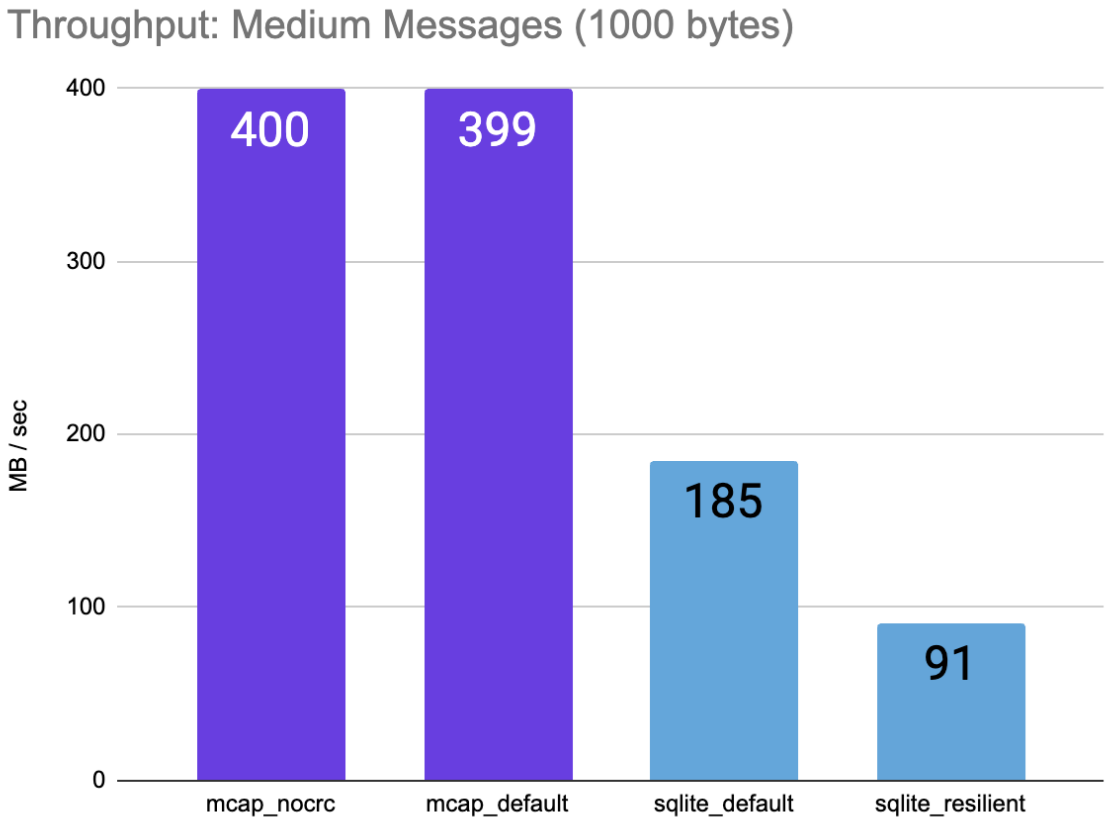
```
0x89, M, C, A, P, 0x30, \r, \n
```

```
[Header]
[Chunk A]
  [Schema A]
  [Channel 1 (A)]
  [Channel 2 (B)]
  [Message on 1]
  [Message on 1]
  [Message on 2]
[Message Index 1]
[Message Index 2]
[Attachment 1]
[Chunk B]
  [Schema B]
  [Channel 3 (B)]
  [Message on 3]
  [Message on 1]
[Message Index 3]
[Message Index 1]
[Data End]
[Schema A]
[Schema B]
[Channel 1]
[Channel 2]
[Channel 3]
[Chunk Index A]
[Chunk Index B]
[Attachment Index 1]
[Statistics]
[Summary Offset 0x01]
[Summary Offset 0x05]
[Summary Offset 0x07]
[Summary Offset 0x08]
[Footer]
```

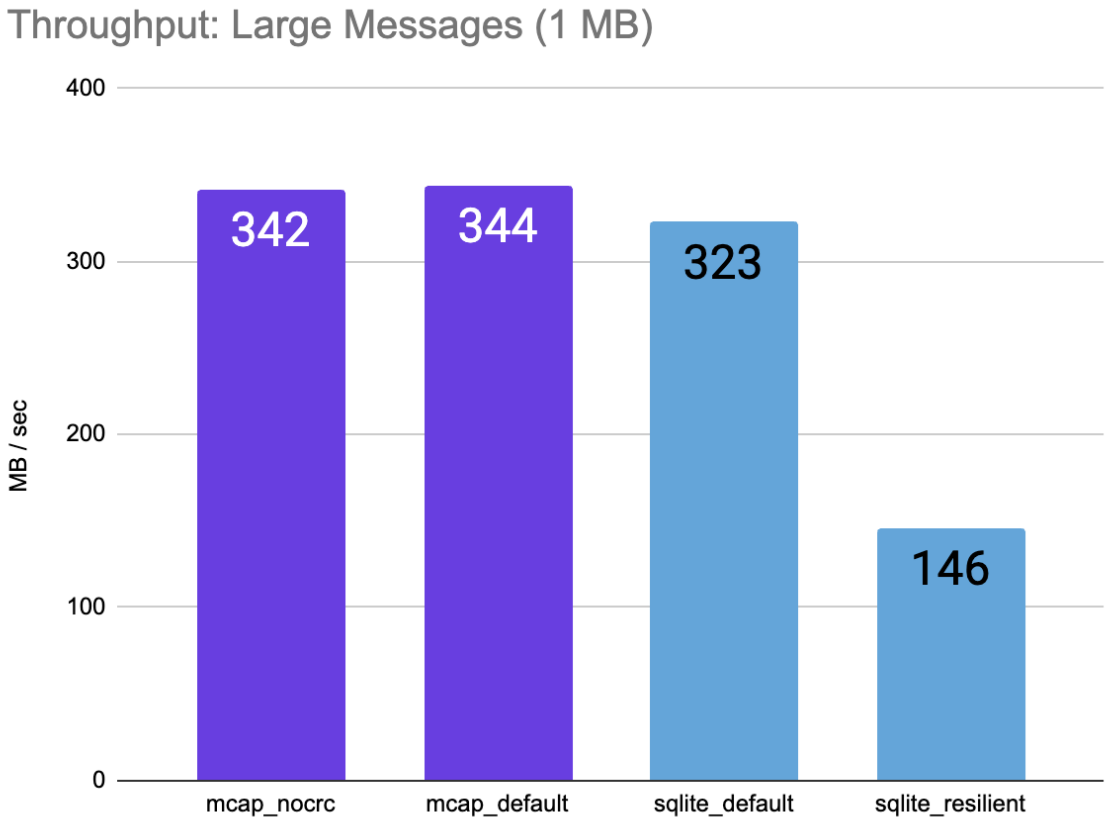
Benchmark: Throughput



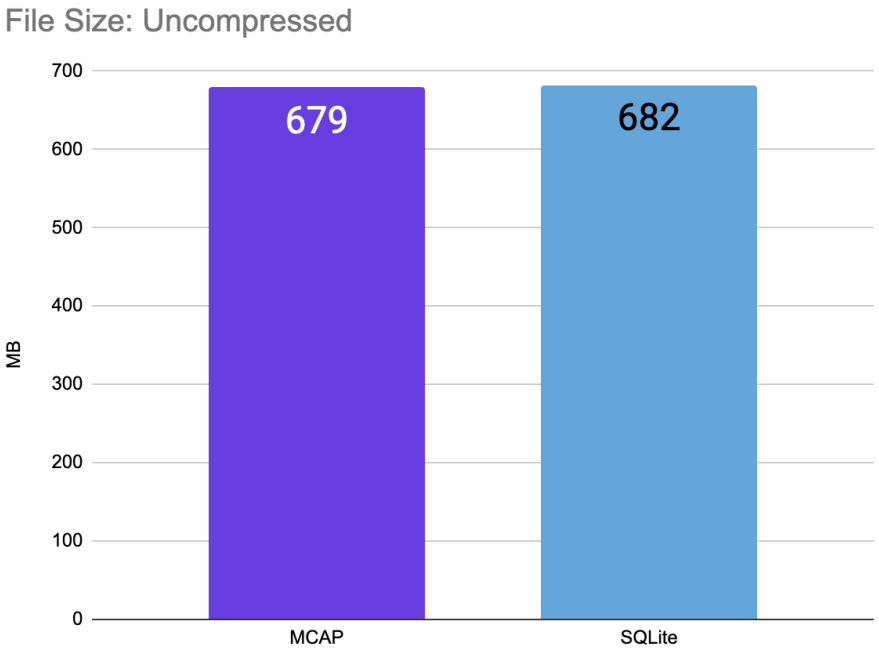
Benchmark: Throughput

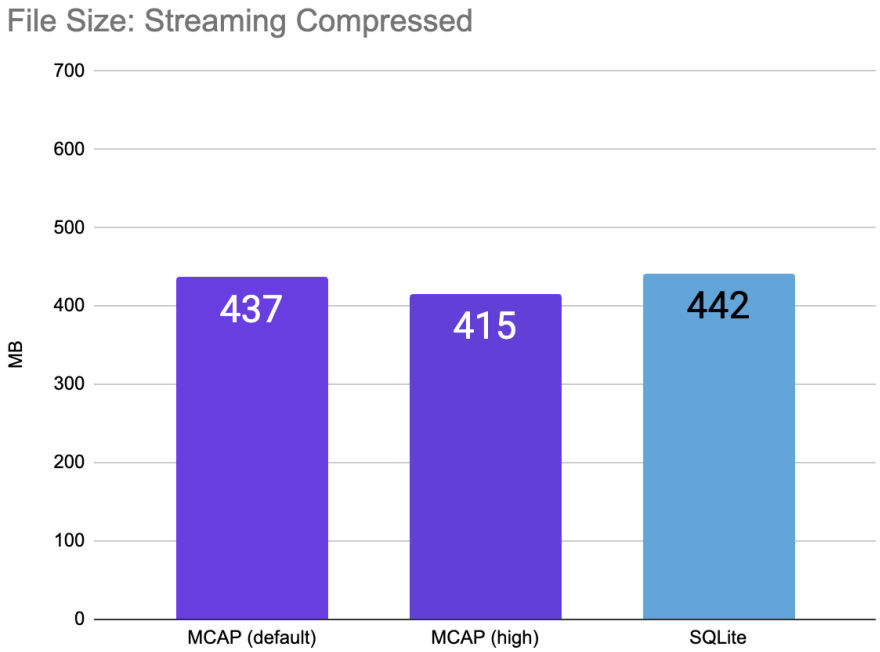


Benchmark: Throughput



Benchmark: File Size



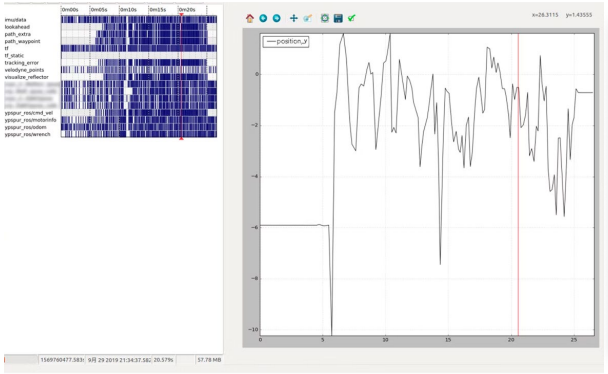


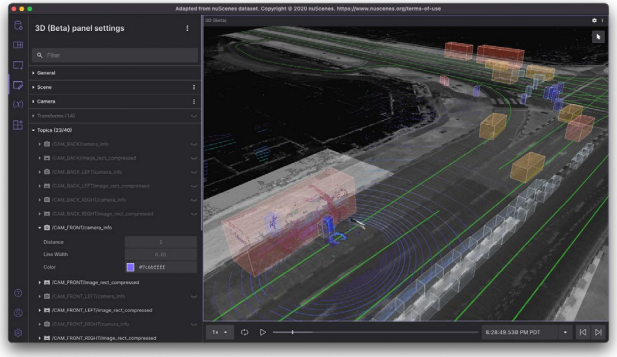
Command Line Interface

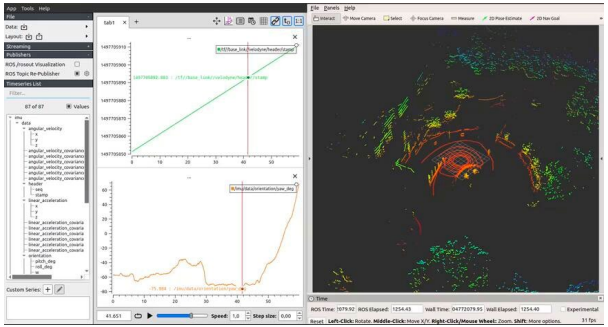
	ROS 1	ROS 2 + SQLite	ROS 2 + MCAP
	JGK:9? J=;GJ<	ros2 bag record	ros2 bag record -s mcap
	JGK:9? AF>G	ros2 bag info	mcap info
	JGK:9? HD9Q	ros2 bag play	ros2 bag play
	rosbag check		mcap doctor
	rosbag fix		mcap recover
	JGK:9? >ADL=J	ros2 bag convert	mcap filter
	JGK:9? ;GEHJ=KK	ros2 bag convert	mcap compress
	rosbag decompress	ros2 bag convert	mcap decompress
	JGK:9? J=AF<=P	ros2 bag reindex	mcap recover

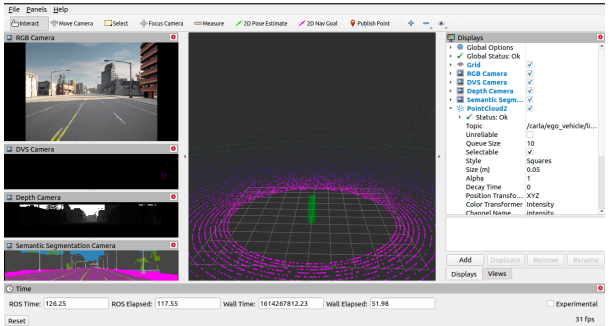
Tooling

- rosbag2
- rviz
- rqt_bag
- Foxglove Studio
- PlotJuggler







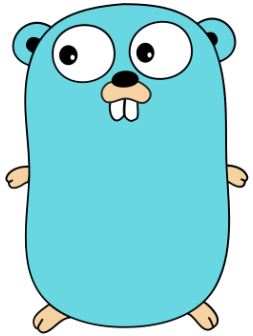


Libraries

Python

C++

TypeScript



Go

Swift

Rust

Example: Python Reader


```python
import sys
from mcap_ros1.reader import read_ros1_messages

for msg in read_ros1_messages(sys.argv[1]):
    print(f"{msg.topic} [{msg.log_time}] "\
          f"({type(msg.ros_msg).__name__}): {msg.ros_msg}")
```

Example: Python Writer

```python
import sys
from mcap_ros2.writer import Writer as McapWriter

with open(sys.argv[1], "wb") as f:
    writer = McapWriter(f)
    schema = writer.register_msgdef("std_msgs/msg/String", "string data")
    for i in range(0, 10):
        msg = {"data": f"string message {i}"}
        writer.write_message("/chatter", schema, msg)
    writer.finish()
```

Example: C++ Reader


```cpp
#define MCAP_IMPLEMENTATION
#include <mcap/reader.hpp>
#include <iostream>
#include <string>

int main(int argc, char* argv[]) {
  mcap::McapReader reader;
  reader.open(argv[1]);
  for (const auto& msgView : reader.readMessages()) {
    auto& msg = msgView.message;
    std::string str{reinterpret_cast<const char*>(msg), msg.dataSize};
    std::cout << msgView.channel->topic << " [" << msg.logTime << "] (" <<
                 msgView.schema->name << "): " << str << "\n";
  }
  reader.close();
}
```

Example: C++ Writer


```cpp
#define MCAP_IMPLEMENTATION
#include <mcap/writer.hpp>
#include <string>

int main(int argc, char** argv) {
  mcap::McapWriter writer;
  writer.open(argv[1], mcap::McapWriterOptions{""});
  mcap::Channel channel{"chatter", "text/plain", 0};
  writer.addChannel(channel);
  for (uint32_t i = 0; i < 10; i++) {
    auto str = std::string{"string message "} + std::to_string(i);
    writer.write(mcap::Message{.channelId = channel.id, .sequence = i,
      .logTime = i, .publishTime = i, .dataSize = str.size(),
      .data = reinterpret_cast<const std::byte*>(str.data())
    });
  }
  writer.close();
}
```

Use MCAP Today

ROS2

- Install: `sudo apt install ros- $ROS_DISTRO-rosbag2- storage- mcap`
- Record: `ros2 bag record -s mcap`
- Use `mcap-ros2-support` API to read and write ROS2 messages programmatically
- Convert existing .db3 files with `mcap convert` or `ros2 bag convert`

ROS1

- Use `mcap-ros1-support` API to read and write ROS1 messages programmatically
- Convert existing .bag files with `mcap convert`
- *Recording can be backported to ROS1 if there's demand*

mcap.dev

Appendix

Tuning For Different Use Cases

No CRCs, Fast Compression, Larger Blocks

```
FG! 0! ~  LJ M=
    ; @MFC1 AR=~  [ V[ [ YS[     µ  í [ +  ì
    ; GEHJ =KKAGF~  â 8 KL<â
    ; GEHJ =KKAGF*=N=D~  â $ 9 KL=KLâ
```

Minimal Overhead

```
FG! 0! ~  LJ M=
    FG! @MFCAF?~  LJ M=
    FG1 MEE9 J Q~  LJ M=
```