DDS Record Replay Documentation

Release ..

eProsima

Mar 26, 2024

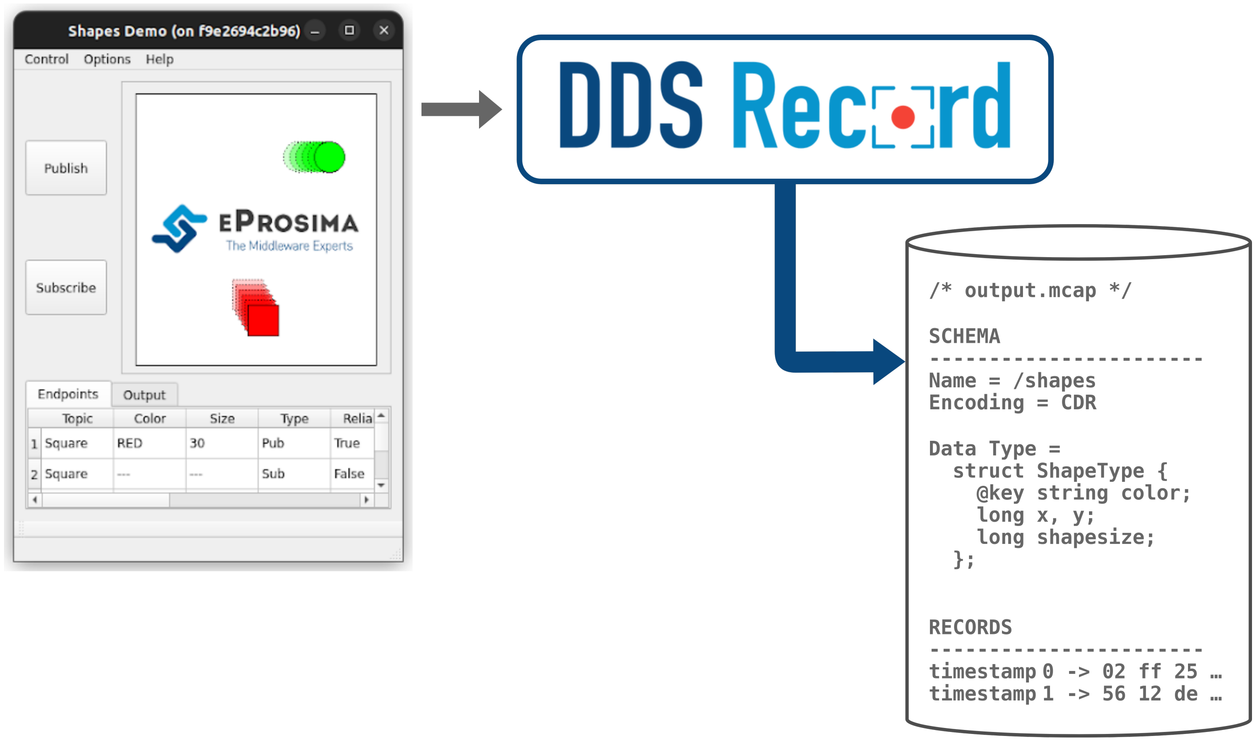
INTRODUCTION

eProsima DDS Record & Replay is an end-user software application that efficiently saves DDS data published into a DDS environment in a MCAP format database. Thus, the exact playback of the recorded network events is possible as the data is linked to the timestamp at which the original data was published.

eProsima DDS Record & Replay is easily configurable and installed with a default setup, so that DDS topics, data types and entities are automatically discovered without the need to specify the types of data recorded. This is because the recording tool exploits the DynamicTypes functionality of eProsima Fast DDS, the C++ implementation of the DDS (Data Distribution Service) Specification defined by the Object Management Group (OMG).

eProsima DDS Record & Replay includes the following tools:

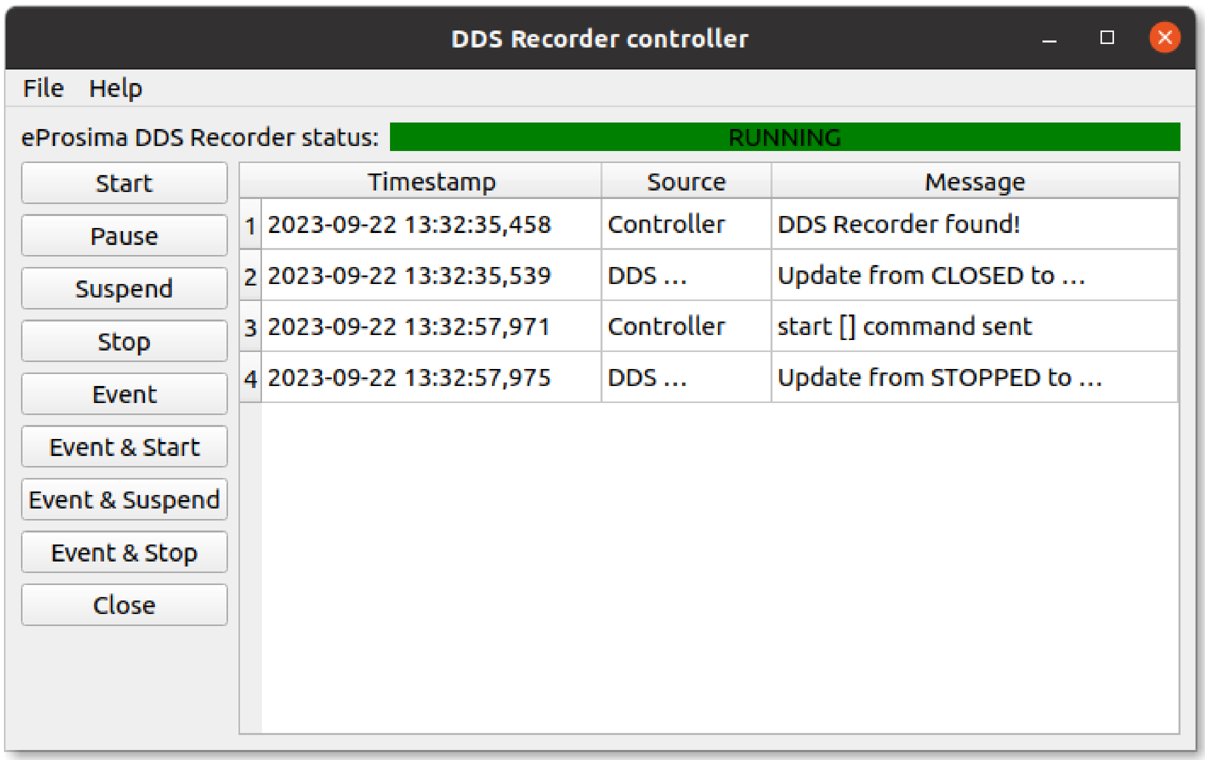
- **DDS Recorder tool**. The main functionality of this tool is to save the data in a MCAP database. The database contains the records of the publication timestamp of the data, the serialized data, and the definition of the data serialization type and format. The output MCAP file can be read with any user tool compatible with MCAP file reading since it contains all the necessary information for reading and reproducing the data.



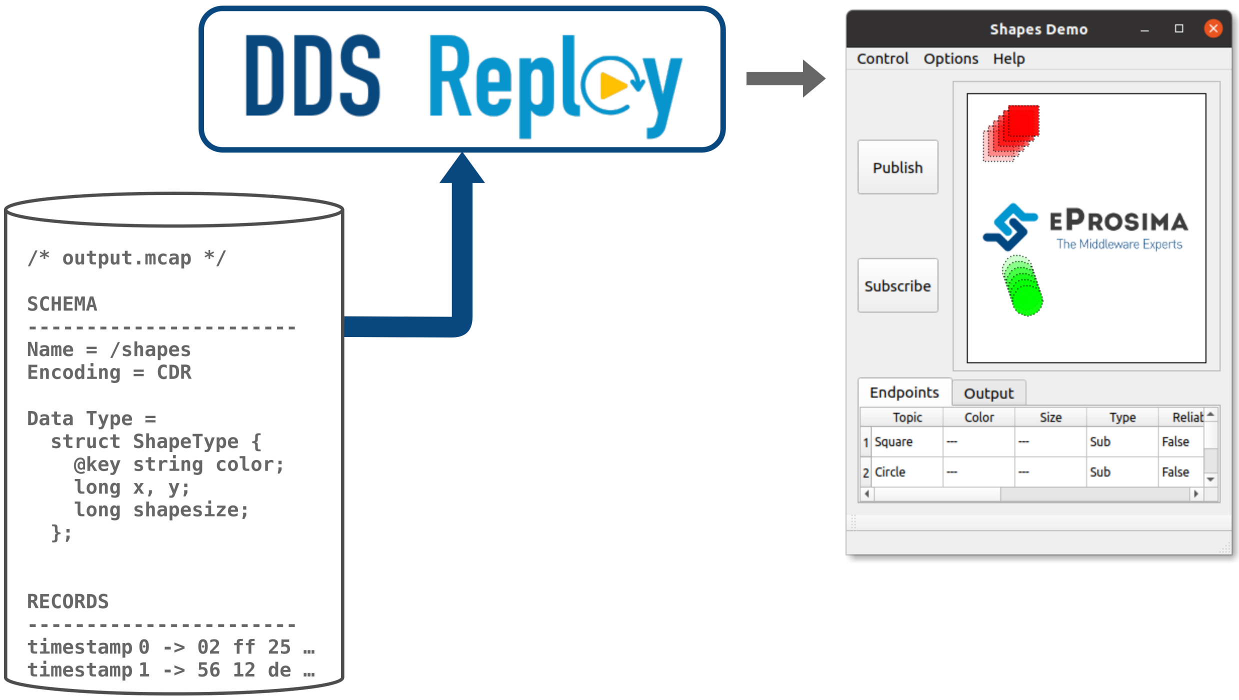
	Topic	Color	Size	Type	Relia
1	Square	RED	30	Pub	True
2	Square	---	---	Sub	False

```
/* output.mcap */

SCHEMA
------------------------
Name = /shapes
Encoding = CDR

Data Type =
  struct ShapeType {
    @key string color;
    long x, y;
    long shapesize;
  };


RECORDS
------------------------
timestamp 0 -> 02 ff 25 …
timestamp 1 -> 56 12 de …
```

- **DDS Remote Controller tool**. This application allows remote control of the recording tool. Thus, a user can have the recording tool on a device and from another device send commands to start, stop or pause data recording.
- **DDS Replay tool**. This application allows to reproduce DDS traffic recorded with a *DDS Recorder*. A user can specify which messages to replay by either setting a time range (begin/end times) out of which messages will be discarded, or directly by blocking/whitelisting a set of topics of interest. It is also possible to choose a different playback rate, as well as to use topic QoS different to the ones recorded.



	Timestamp	Source	Message
1	2023-09-22 13:32:35,458	Controller	DDS Recorder found!
2	2023-09-22 13:32:35,539	DDS ...	Update from CLOSED to ...
3	2023-09-22 13:32:57,971	Controller	start [] command sent
4	2023-09-22 13:32:57,975	DDS ...	Update from STOPPED to ...



```
/* output.mcap */

SCHEMA
-----------------------
Name = /shapes
Encoding = CDR

Data Type =
  struct ShapeType {
    @key string color;
    long x, y;
    long shapesize;
  };


RECORDS
-----------------------
timestamp 0 -> 02 ff 25 …
timestamp 1 -> 56 12 de …
```

	Topic	Color	Size	Type	Reliat
1	Square	—	—	Sub	False
2	Circle	—	—	Sub	False

CHAPTER

ONE

CONTACTS AND COMMERCIAL SUPPORT

Find more about us at eProsima's webpage.

Support available at:

- Email: support@eprosima.com
- Phone: +34 91 804 34 48

CHAPTER

TWO

CONTRIBUTING TO THE DOCUMENTATION

DDS Record & Replay Documentation is an open source project, and as such all contributions, both in the form of feedback and content generation, are most welcomed. To make such contributions, please refer to the Contribution Guidelines hosted in our GitHub repository.

CHAPTER

THREE

STRUCTURE OF THE DOCUMENTATION

This documentation is organized into the sections below.

- *Installation Manual*
- *Recording application*
- *Replay application*
- *Tutorials*
- *Developer Manual*
- *Release Notes*

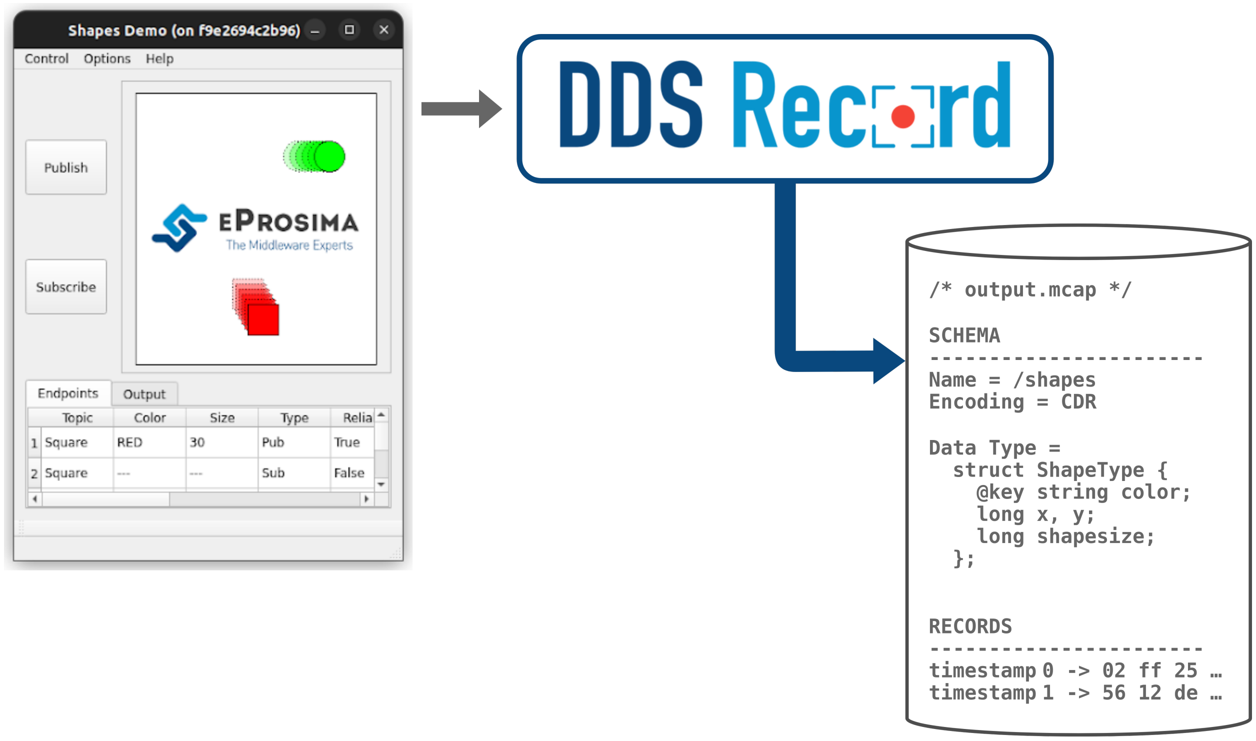
eProsima DDS Record & Replay is an end-user software application that efficiently saves DDS data published into a DDS environment in a MCAP format database. Thus, the exact playback of the recorded network events is possible as the data is linked to the timestamp at which the original data was published.

eProsima DDS Record & Replay is easily configurable and installed with a default setup, so that DDS topics, data types and entities are automatically discovered without the need to specify the types of data recorded. This is because the recording tool exploits the DynamicTypes functionality of eProsima Fast DDS, the C++ implementation of the DDS (Data Distribution Service) Specification defined by the Object Management Group (OMG).

3.1 Overview

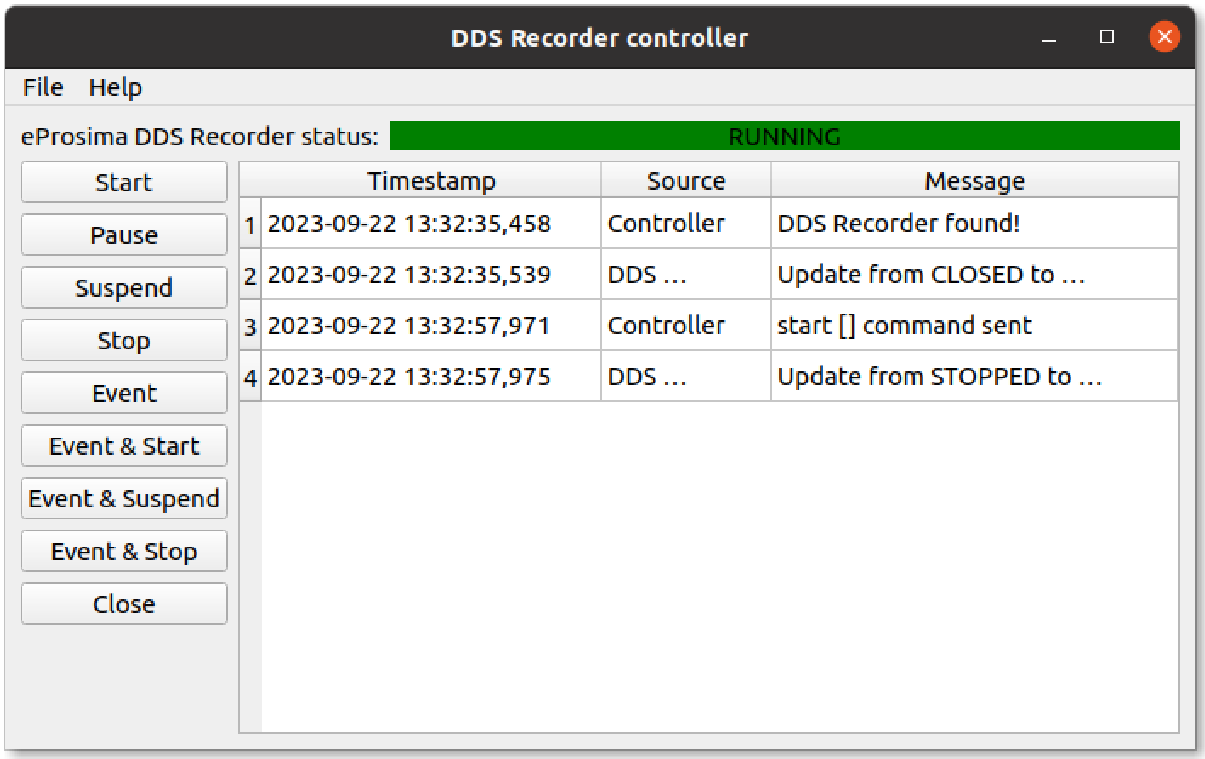
eProsima DDS Record & Replay includes the following tools:

- **DDS Recorder tool**. The main functionality of this tool is to save the data in a MCAP database. The database contains the records of the publication timestamp of the data, the serialized data, and the definition of the data serialization type and format. The output MCAP file can be read with any user tool compatible with MCAP file reading since it contains all the necessary information for reading and reproducing the data.
- **DDS Remote Controller tool**. This application allows remote control of the recording tool. Thus, a user can have the recording tool on a device and from another device send commands to start, stop or pause data recording.
- **DDS Replay tool**. This application allows to reproduce DDS traffic recorded with a *DDS Recorder*. A user can specify which messages to replay by either setting a time range (begin/end times) out of which messages will be discarded, or directly by blocking/whitelisting a set of topics of interest. It is also possible to choose a different playback rate, as well as to use topic QoS different to the ones recorded.



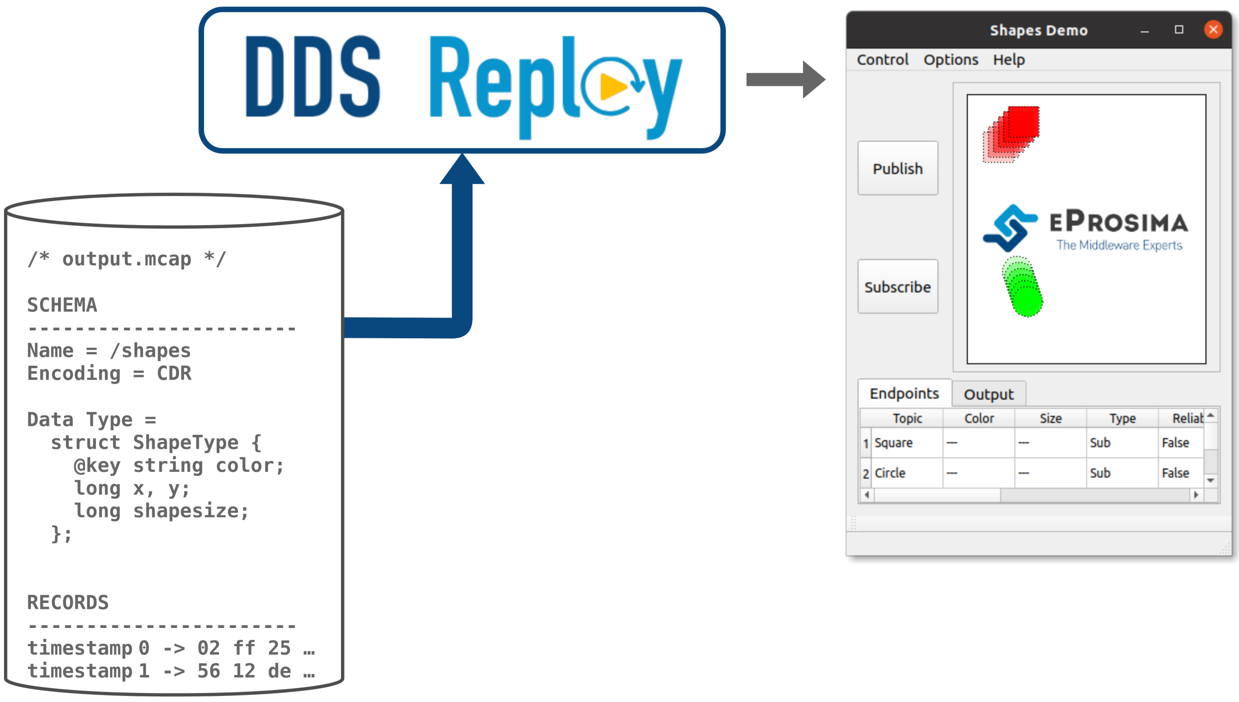
	Topic	Color	Size	Type	Relia
1	Square	RED	30	Pub	True
2	Square	---	---	Sub	False

```
/* output.mcap */

SCHEMA
-----------------------
Name = /shapes
Encoding = CDR

Data Type =
  struct ShapeType {
    @key string color;
    long x, y;
    long shapesize;
  };


RECORDS
-----------------------
timestamp 0 -> 02 ff 25 …
timestamp 1 -> 56 12 de …
```



eProsima DDS Recorder status: RUNNING

	Timestamp	Source	Message
1	2023-09-22 13:32:35,458	Controller	DDS Recorder found!
2	2023-09-22 13:32:35,539	DDS …	Update from CLOSED to …
3	2023-09-22 13:32:57,971	Controller	start [] command sent
4	2023-09-22 13:32:57,975	DDS …	Update from STOPPED to …



```
/* output.mcap */

SCHEMA
-----------------------
Name = /shapes
Encoding = CDR

Data Type =
  struct ShapeType {
    @key string color;
    long x, y;
    long shapesize;
  };


RECORDS
-----------------------
timestamp 0 -> 02 ff 25 …
timestamp 1 -> 56 12 de …
```

	Topic	Color	Size	Type	Reliat
1	Square	---	---	Sub	False
2	Circle	---	---	Sub	False

3.2 Contacts and Commercial support

Find more about us at eProsima's webpage.

Support available at:

- Email: support@eprosima.com
- Phone: +34 91 804 34 48

3.3 Contributing to the documentation

DDS Record & Replay Documentation is an open source project, and as such all contributions, both in the form of feedback and content generation, are most welcomed. To make such contributions, please refer to the Contribution Guidelines hosted in our GitHub repository.

3.4 Structure of the documentation

This documentation is organized into the sections below.

- *Installation Manual*
- *Recording application*
- *Replay application*
- *Tutorials*
- *Developer Manual*
- *Release Notes*

3.5 DDS Record & Replay on Windows

Warning: The current version of *DDS Record & Replay* does not have installers for Windows platforms. Please refer to the *Windows installation from sources* section to learn how to build *DDS Record & Replay* on Windows from sources.

3.6 DDS Record & Replay on Linux

Warning: The current version of *DDS Record & Replay* does not have installers for Linux platforms. Please refer to the *Linux installation from sources* section to learn how to build *DDS Record & Replay* on Linux from sources.

3.7 Docker Image (recommended)

Warning: Currently, *DDS Record & Replay* Docker image only contains *DDS Recorder* tool, *DDS Replay tool* application will be added soon.

eProsima distributes a Docker image of *DDS Record & Replay* with Ubuntu 22.04 as base image. This image launches an instance of *DDS Record & Replay* that is configured using a *YAML* configuration file provided by the user and shared with the Docker container. The steps to run *DDS Record & Replay* in a Docker container are explained below.

1. Download the compressed Docker image in `.tar` format from the eProsima Downloads website. It is strongly recommended to download the image corresponding to the latest version of *DDS Record & Replay*.

2. Extract the image by executing the following command:

   ```
   load ubuntu-ddsrecorder:<version>.tar
   ```

   where `version` is the downloaded version of *DDS Record & Replay*.

3. Build a *DDS Record & Replay* configuration YAML file on the local machine. This will be the *DDS Record & Replay* configuration file that runs inside the Docker container. To continue this installation manual, let's use the configuration file provided in *this tutorial*. Open your preferred text editor and copy the configuration example from *here* into the `/<dds_recorder_ws>/DDS_RECORDER_CONFIGURATION.yaml` file, where `dds_recorder_ws` is the path of the configuration file. To make this accessible from the Docker container we will create a shared volume containing just this file. This is explained in next point.

4. Run the Docker container executing the following command:

   ```
   docker run -it \
       --net=host \
       --ipc=host \
       --privileged \
   ```

(continues on next page)

(continued from previous page)

```
    -v /<dds_recorder_ws>/DDS_RECORDER_CONFIGURATION.yaml:/root/DDS_RECORDER_
↪CONFIGURATION.yaml \
    ubuntu-ddsrecorder:v0.3.0
```

It is important to mention that both the path to the configuration file hosted in the local machine and the one created in the Docker container must be absolute paths in order to share just one single file as a shared volume.

After executing the previous command you should be able to see the initialization traces from the *DDS Record & Replay* running in the Docker container. If you want to terminate the application gracefully, just press `Ctrl+C` to stop the execution of *DDS Record & Replay*.

3.8 Getting Started

3.8.1 Project Overview

eProsima DDS Record & Replay is a cross-platform application developed by eProsima and powered by *Fast DDS* that contains a set of tools for debugging DDS networks. Among these tools is a recording application, called *DDS Recorder*, which allows a user to capture data published in a DDS environment for later analysis or playback.

The *DDS Recorder* application automatically discovers all topics in the DDS network and saves the data published in each topic with the publication timestamp of the data. Furthermore, by using the DynamicTypes feature of *Fast DDS*, it is possible to record the type of the data in the MCAP file. The benefit of this comes from the fact that the data is saved serialized according to the CDR format.

By default, *eProsima DDS Recorder* saves all DDS traffic encountered in the domain of choice, storing samples in the same form they are received (serialized) without the need to have received the (dynamic) type associated to these samples. When recorded samples with no associated type information are played back through a *DDS Replayer*, only DDS applications in possession of this type information will be able to receive and process these messages.

However, some applications might not have this information available out of the box, as it is the case of applications relying on *Dynamic Types*. Additionally, tools such as Foxglove Studio require this information to be stored in the resulting MCAP file so messages can be deserialized for visualization. In such scenarios, it is required that type information gets stored along with data samples, which is automatically done by a *DDS Recorder* instance as long as the publisher applications (whose messages are recorded) send this information. This can be easily achieved by applying the configuration described in *this section*.

Moreover, *DDS Recorder* is designed to ensure that internal communications are handled efficiently, from the reception of the data to its storage in the output database. This is achieved through the internal implementation of a zero-copy communication mechanism implemented in one of the *DDS Recorder* base libraries. It is also possible to configure the number of threads that execute these data reception and saving tasks, as well as the size of the internal buffers to avoid writing to disk with each received data.

Usage Description

DDS Recorder is a terminal (non-graphical) application that creates a recording service as long as it is running. Although most use cases are covered by the default configuration, the *DDS Recorder* can be configured via a YAML file, whose format is very intuitive and human-readable.

- **Run**: Only the command that launches the application (`ddsrecorder`) needs to be executed to run a *DDS Recorder*. Please, read this *section* to apply a specific configuration, and this *section* to see the supported arguments.
- **Interact**: Once the *DDS Recorder* application is running, the allowlist and blocklist topic lists could be changed in runtime by just changing the YAML configuration file. It is also possible to change the status of the recorder

(RUNNING, PAUSED, SUSPENDED, STOPPED or CLOSED) by remote control of the application. This remote control is done by sending commands via DDS or by using the graphical remote control application provided with the *eProsima DDS Record & Replay* software tool (see *Remote control*).

- **Close**: To close the *DDS Recorder* application just send a *Ctrl+C* signal to terminate the process gracefully (see *Closing Recording Application*) or close it remotely using the remote control application (see *Remote control*).

Common Use cases

To get started with *DDS Recorder*, please visit section *Example of usage*. In addition, this documentation provides several tutorials on how to set up a *DDS Recorder*, a comprehensive Fast DDS application using DynamicTypes and how to read the generated MCAP file.

3.8.2 Example of usage

This example will serve as a hands-on tutorial, aimed at introducing some of the key concepts and features that *eProsima DDS Record & Replay* recording application (*DDS Recorder* or `ddsrecorder`) has to offer.

Prerequisites

It is required to have *eProsima DDS Record & Replay* previously installed using one of the following installation methods:

- *DDS Record & Replay on Windows*
- *DDS Record & Replay on Linux*
- *Docker Image (recommended)*

Additionally, ShapesDemo is required to publish and subscribe shapes of different colors and sizes. ShapesDemo application is already prepared to use Fast DDS DynamicTypes, which is required when using the DDS Recorder. Install it by following any of the methods described in the given links:

- Windows installation from binaries
- Linux installation from sources
- Docker Image

Start ShapesDemo

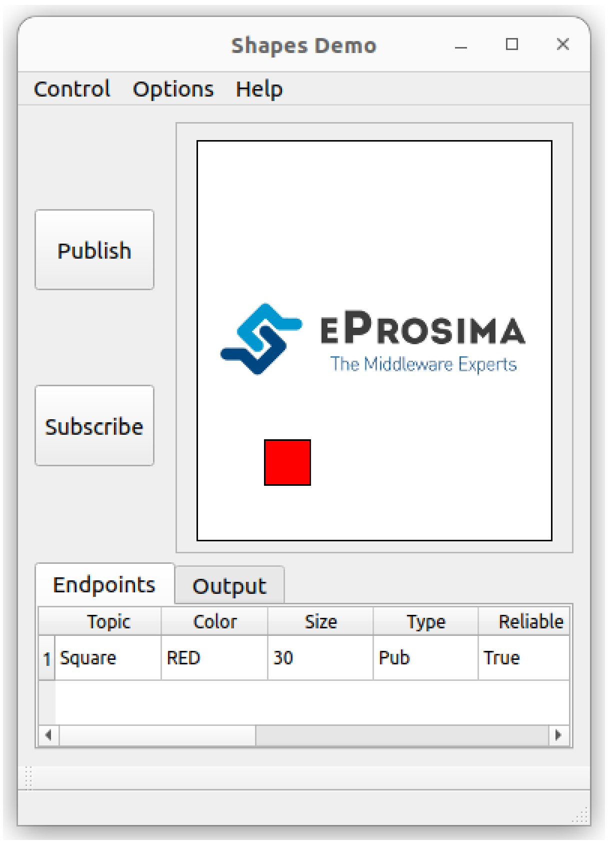
Let us launch a ShapesDemo instance and start publishing in topics `Square` with default settings.

Recorder configuration

DDS Recorder runs with default configuration settings. This default configuration records all messages of all DDS Topics found in DDS Domain `0` in the `output_YYYY-MM-DD-DD_hh-mm-ss.mcap` file.

Additionally, it is possible to change the default configuration parameters by means of a YAML configuration file.

Note: Please refer to *Configuration* for more information on how to configure a *DDS Recorder*.



	Topic	Color	Size	Type	Reliable
1	Square	RED	30	Pub	True

Recorder execution

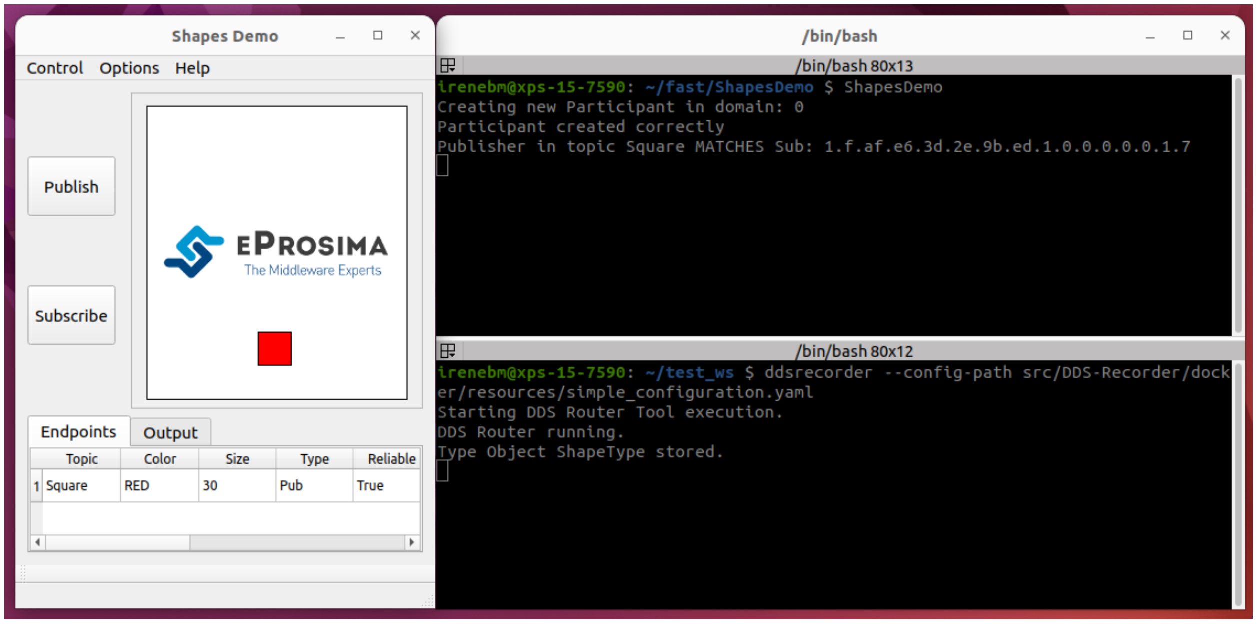
Launching a *DDS Recorder* instance is as easy as executing the following command:

```
ddsrecorder
```

In order to know all the possible arguments supported by this tool, use the command:

```
ddsrecorder --help
```

Stop the recorder with `Ctrl+C` and check that the MCAP file exists.



	Topic	Color	Size	Type	Reliable
1	Square	RED	30	Pub	True

```
irenebm@xps-15-7590: ~/fast/ShapesDemo $ ShapesDemo
Creating new Participant in domain: 0
Participant created correctly
Publisher in topic Square MATCHES Sub: 1.f.af.e6.3d.2e.9b.ed.1.0.0.0.0.0.1.7
```

```
irenebm@xps-15-7590: ~/test_ws $ ddsrecorder --config-path src/DDS-Recorder/dock
er/resources/simple_configuration.yaml
Starting DDS Router Tool execution.
DDS Router running.
Type Object ShapeType stored.
```

Next Steps

Explore section *Tutorials* for more information on how to configure and set up a recorder, as well as to discover multiple scenarios where *DDS Recorder* may serve as a useful tool. Also, feel free to check out *this* example, where a *DDS Replayer* is used to reproduce the traffic recorded following the steps in this tutorial.

3.9 Usage

eProsima DDS Recorder is a user application executed from command line.

3.9.1 Starting Recording Application

Docker Image

The recommended method to run the *DDS Recorder* is to instantiate a Docker container of the *DDS Record & Replay* image. *Here* are the instructions to download the compressed *DDS Record & Replay* Docker image and load it locally.

To run the *DDS Recorder* from a Docker container execute the following command:

```
docker run -it \
    --net=host \
    --ipc=host \
    -v /<dds_recorder_ws>/DDS_RECORDER_CONFIGURATION.yaml:/root/DDS_RECORDER_
↪CONFIGURATION.yaml \
    ubuntu-ddsrecorder:v<X.X.X> ddsrecorder
```

Installation from sources

eProsima DDS Record & Replay depends on `fastrtps`, `fastcdr` and `ddspipe` libraries. In order to correctly execute the recorder, make sure that `fastrtps`, `fastcdr` and `ddspipe` are properly sourced.

```
source <path-to-fastdds-installation>/install/setup.bash
source <path-to-ddspipe-installation>/install/setup.bash
source <path-to-ddsrecordreplay-installation>/install/setup.bash
```

Note: If Fast DDS, DDS Pipe and DDS Record & Replay have been installed in the system, these libraries would be sourced by default.

To start *eProsima DDS Recorder* with a default configuration, enter:

```
ddsrecorder
```

3.9.2 Closing Recording Application

SIGINT

To close *eProsima DDS Recorder*, press `Ctrl+C`. *DDS Recorder* will perform a clean shutdown.

SIGTERM

Write command `kill <pid>` in a different terminal, where `<pid>` is the id of the process running the *DDS Recorder*. Use `ps` or `top` programs to check the process ids.

TIMEOUT

Setting a maximum amount of seconds that the application will work using argument `--timeout` will close the application once the time has expired.

3.9.3 Recording Service Command-Line Parameters

The *DDS Recorder* application supports several input arguments:

Command	Description	Option	Possible Values	Default Value
Help	It shows the usage information of the application.	`-h` `--help`		
Version	It shows the current version of the *DDS Recorder* and the hash of the last commit of the compiled code.	`-v` `--version`		
Configuration File	Configuration file path.	`-c` `--config-path`		`./DDS_RECORDER_CONFIGURATION.yaml`
Reload Timer	The configuration file will be automatically reloaded according to the specified time period.	`-r` `--reload-time`	Unsigned Integer	`0`
Timeout	Set a maximum time while the application will be running. `0` means that the application will run forever (until kill via signal).	`-t` `--timeout`	Unsigned Integer	`0`
Debug	Enables the *DDS Recorder* logs so the execution can be followed by internal debugging information. Sets `Log Verbosity` to `info` and `Log Filter` to `DDSRECORDER`.	`-d` `--debug`		
Log Verbosity	Set the verbosity level so only log messages with equal or higher importance level are shown.	`--log-verbosity`	`info` `warning` `error`	`warning`
Log Filter	Set a regex string as filter.	`--log-filter`	String	`"DDSRECORDER"`

3.10 Configuration

3.10.1 DDS Recorder Configuration

A *DDS Recorder* is configured by a *.yaml* configuration file. This *.yaml* file contains all the information regarding the DDS interface configuration, recording parameters, and *DDS Recorder* specifications. Thus, this file has four major configuration groups:

- `dds`: configuration related to DDS communication.
- `recorder`: configuration of data writing in the database.
- `remote-controller`: configuration of the remote controller of the *DDS Recorder*.
- `specs`: configuration of the internal operation of the *DDS Recorder*.

DDS Configuration

Configuration related to DDS communication.

DDS Domain

Tag `domain` configures the *Domain Id*.

```yaml
domain: 101
```

Built-in Topics

The discovery phase can be accelerated by listing topics under the `builtin-topics` tag. The *DDS Recorder* will create the DataWriters and DataReaders for these topics in the *DDS Recorder* initialization. The *Topic QoS* for these topics can be manually configured with the *Manual Topic* and with the *Specs Topic QoS*; if a *Topic QoS* is not configured, it will take its default value.

The `builtin-topics` must specify a `name` and `type` without wildcard characters.

Example of usage:

```yaml
builtin-topics:
  - name: HelloWorldTopic
    type: HelloWorld
```

Topic Filtering

The *DDS Recorder* automatically detects the topics that are being used in a DDS Network. The *DDS Recorder* then creates internal DDS *Readers* to record the data published on each topic. The *DDS Recorder* allows filtering DDS *Topics* to allow users to configure the DDS *Topics* that must be recorded. These data filtering rules can be configured under the `allowlist` and `blocklist` tags. If the `allowlist` and `blocklist` are not configured, the *DDS Recorder* will recorded the data published on every topic it discovers. If both the `allowlist` and `blocklist` are configured and a topic appears in both of them, the `blocklist` has priority and the topic will be blocked.

Topics are determined by the tags `name` (required) and `type`, both of which accept wildcard characters.

Note: Placing quotation marks around values in a YAML file is generally optional, but values containing wildcard characters do require single or double quotation marks.

Consider the following example:

```yaml
allowlist:
  - name: AllowedTopic1
    type: Allowed

  - name: AllowedTopic2
    type: "*"

  - name: HelloWorldTopic
    type: HelloWorld

blocklist:
  - name: "*"
    type: HelloWorld
```

In this example, the data published in the topic `AllowedTopic1` with type `Allowed` and in the topic `AllowedTopic2` with any type will be recorded by the *DDS Recorder*. The data published in the topic `HelloWorldTopic` with type `HelloWorld` will be blocked, since the `blocklist` is blocking all topics with any name and with type `HelloWorld`.

Topic QoS

The following is the set of QoS that are configurable for a topic. For more information on topics, please read the Fast DDS Topic section.

Quality of Service	Yaml tag	Data type	Default value	QoS set
Reliability	`reliability`	*bool*	`false`	`RELIABLE` / `BEST_EFFORT`
Durability	`durability`	*bool*	`false`	`TRANSIENT_LOCAL` / `VOLATILE`
Ownership	`ownership`	*bool*	`false`	`EXCLUSIVE_OWNERSHIP_QOS` / `SHARED_OWNERSHIP_QOS`
Partitions	`partitions`	*bool*	`false`	Topic with / without partitions
Key	`keyed`	*bool*	`false`	Topic with / without key
History Depth	`history-depth`	*unsigned integer*	`5000`	*History Depth*
Max Reception Rate	`max-rx-rate`	*float*	`0` (unlimited)	*Max Reception Rate*
Downsampling	`downsampling`	*unsigned integer*	`1`	*Downsampling*

Warning: Manually configuring `TRANSIENT_LOCAL` durability may lead to incompatibility issues when the discovered reliability is `BEST_EFFORT`. Please ensure to always configure the `reliability` when configuring the `durability` to avoid the issue.

History Depth

The `history-depth` tag configures the history depth of the Fast DDS internal entities. By default, the depth of every RTPS History instance is `5000`, which sets a constraint on the maximum number of samples a *DDS Recorder* instance can deliver to late joiner Readers configured with `TRANSIENT_LOCAL` DurabilityQosPolicyKind. Its value should be decreased when the sample size and/or number of created endpoints (increasing with the number of topics) are big enough to cause memory exhaustion issues. If enough memory is available, however, the `history-depth` could be increased to deliver a greater number of samples to late joiners.

Max Reception Rate

The `max-rx-rate` tag limits the frequency [Hz] at which samples are processed by discarding messages received before `1/max-rx-rate` seconds have passed since the last processed message. It only accepts non-negative numbers. By default it is set to `0`; it processes samples at an unlimited reception rate.

Downsampling

The `downsampling` tag reduces the sampling rate of the received data by only keeping *1* out of every *n* samples received (per topic), where *n* is the value specified under the `downsampling` tag. When the `max-rx-rate` tag is also set, downsampling only applies to messages that have passed the `max-rx-rate` filter. It only accepts positive integers. By default it is set to 1; it accepts every message.

Manual Topics

A subset of *Topic QoS* can be manually configured for a specific topic under the tag `topics`. The tag `topics` has a required `name` tag that accepts wildcard characters. It also has two optional tags: a `type` tag that accepts wildcard characters, and a `qos` tag with the *Topic QoS* that the user wants to manually configure. If a `qos` is not manually configured, it will get its value by discovery.

```yaml
topics:
  - name: "temperature/*"
    type: "temperature/types/*"
    qos:
      max-rx-rate: 15
      downsampling: 2
```

Note: The *Topic QoS* configured in the Manual Topics take precedence over the *Specs Topic QoS*.

Ignore Participant Flags

A set of discovery traffic filters can be defined in order to add an extra level of isolation. This configuration option can be set through the `ignore-participant-flags` tag:

```yaml
ignore-participant-flags: no_filter                       # No filter (default)
# or
ignore-participant-flags: filter_different_host           # Discovery traffic from␣
↪another host is discarded
```

(continues on next page)

(continued from previous page)

```
# or
ignore-participant-flags: filter_different_process           # Discovery traffic from␣
↪another process on same host is discarded
# or
ignore-participant-flags: filter_same_process                # Discovery traffic from␣
↪own process is discarded
# or
ignore-participant-flags: filter_different_and_same_process  # Discovery traffic from␣
↪own host is discarded
```

See Ignore Participant Flags for more information.

Custom Transport Descriptors

By default, *DDS Recorder* internal participants are created with enabled UDP and Shared Memory transport descriptors. The use of one or the other for communication will depend on the specific scenario, and whenever both are viable candidates, the most efficient one (Shared Memory Transport) is automatically selected. However, a user may desire to force the use of one of the two, which can be accomplished via the `transport` configuration tag.

```
transport: builtin    # UDP & SHM (default)
# or
transport: udp        # UDP only
# or
transport: shm        # SHM only
```

Warning: When configured with `transport: shm`, *DDS Recorder* will only communicate with applications using Shared Memory Transport exclusively (with disabled UDP transport).

Interface Whitelist

Optional tag `whitelist-interfaces` allows to limit the network interfaces used by UDP and TCP transport. This may be useful to only allow communication within the host (note: same can be done with *Ignore Participant Flags*). Example:

```
whitelist-interfaces:
  - "127.0.0.1"    # Localhost only
```

See Interface Whitelist for more information.

Recorder Configuration

Configuration of data writing in the database.

Output File

The recorder output file does support the following configuration settings under the `output` tag:

Parameter	Tag	Description	Data type	Default value
File path	`path`	Configure the path to save the output file.	`string`	`.`
File name	`filename`	Configure the name of the output file.	`string`	`output`
Timestamp format	`timestamp-format`	Configure the format of the output file timestamp (as in `std::put_time`).	`string`	`%Y-%m-%d_%H-%M-%S_%Z`
Local timestamp	`local-timestamp`	Whether to use a local or global (GMT) timestamp.	`boolean`	`true`

When DDS Recorder application is launched (or when remotely controlled, every time a `start/pause` command is received while in `SUSPENDED/STOPPED` state), a temporary file with `filename` name (+timestamp prefix) and `.mcap.tmp~` extension is created in `path`. This file is not readable until the application is terminated (or a `suspend/stop/close` command is received). On such event, the temporal file is renamed to have `.mcap` extension in the same location, and is then ready to be processed.

Buffer size

`buffer-size` indicates the number of samples to be stored in the process memory before the dump to disk. This avoids disk access each time a sample is received. By default, its value is set to `100`.

Event Window

DDS Recorder can be configured to continue saving data when it is in paused mode. Thus, when an event is triggered from the remote controller, samples received in the last `event-window` seconds are stored in the database.

In other words, the `event-window` acts as a sliding time window that allows to save the collected samples in this time window only when the remote controller event is received. By default, its value is set to `20` seconds.

Log Publish Time

By default (`log-publish-time: false`) received messages are stored in the MCAP file with `logTime` value equals to the reception timestamp. Additionally, the timestamp corresponding to when messages were initially published (`publishTime`) is also included in the information dumped to MCAP files. In some applications, it may be required to use the `publishTime` as `logTime`, which can be achieved by providing the `log-publish-time: true` configuration option.

Only With Type

By default, all (allowed) received messages are recorded regardless of whether their associated type information has been received. However, a user can enforce that **only** samples whose type is received are recorded by setting `only-with-type: true`.

Compression

Compression settings for writing to an MCAP file can be specified under the `compression` configuration tag. The supported compression options are:

Parameter	Tag	Description	Data type	Default value	Possible values
Compression Algorithm	`algorithm`	Compression algorithm to use when writing Chunks.	`string`	`zstd`	`none` `lz4` `zstd`
Compression Level	`level`	Compression level to use when writing Chunks.	`string`	`default`	`fastest` `fast` `default` `slow` `slowest`
Force Compression	`force`	Force compression on all Chunks (even for those that do not benefit from compression).	`boolean`	`false`	`true` `false`

Record Types

By default, all type information received during execution is stored in a dedicated MCAP file section. This information is then leveraged by *DDS Replayer* on playback, publishing recorded types in addition to data samples, which may be required for receiver applications relying on *Dynamic Types* (see *Replay Types*). However, a user may choose to disable this feature by setting `record-types: false`.

Topic type format

The optional `ros2-types` tag enables specification of the format for storing schemas. When set to `true`, schemas are stored in ROS 2 message format (.msg). If set to `false`, schemas are stored in OMG IDL format (.idl). By default it is set to `false`.

Remote Controller

Configuration of the DDS remote control system. Please refer to *Remote Control* for further information on how to use *DDS Recorder* remotely. The supported configurations are:

Parameter	Tag	Description	Data type	Default value	Possible values
Enable	`enable`	Enable DDS remote control system.	`boolean`	`true`	`true` `false`
DDS Domain	`domain`	DDS Domain of the DDS remote control system.	`integer`	DDS domain being recorded	From `0` to `255`
Initial state	`initial-state`	Initial state of *DDS Recorder*.	`string`	`RUNNING`	`RUNNING` `PAUSED` `SUSPENDED` `STOPPED`
Command Topic Name	`command-topic-name`	Name of Controller Command DDS Topic.	`string`	`/ddsrecorder/command`	
Status Topic Name	`status-topic-name`	Name of Controller Status DDS Topic.	`string`	`/ddsrecorder/status`	

Specs Configuration

The internals of a *DDS Recorder* can be configured using the `specs` optional tag that contains certain options related with the overall configuration of the *DDS Recorder* instance to run. The values available to configure are:

Number of Threads

`specs` supports a `threads` optional value that allows the user to set a maximum number of threads for the internal `ThreadPool`. This ThreadPool allows to limit the number of threads spawned by the application. This improves the performance of the internal data communications.

This value should be set by each user depending on each system characteristics. In case this value is not set, the default number of threads used is `12`.

Maximum Number of Pending Samples

It is possible that a *DDS Recorder* starts receiving data from a topic that it has not yet registered, i.e. a topic for which it does not know the data type. In this case, messages are kept in an internal circular buffer until their associated type information is received, event on which they are written to disk.

However, the recorder execution might end before this event ever occurs. Depending on configuration (see *Only With Type*), messages kept in the pending samples buffer will be stored or not on closure. Hence, note that memory consumption would continuously grow whenever a sample with unknown type information is received.

To avoid the exhaustion of memory resources in such scenarios, a configuration option is provided which lets the user set a limit on memory usage. The `max-pending-samples` parameter allows to configure the size of the aforementioned circular buffers **for each topic** that is discovered. The default value is equal to `5000` samples, with `-1` meaning no limit, and `0` no pending samples.

Depending on the combination of this configuration option and the value of `only-with-type`, the following situations may arise when a message with unknown type is received:

- If `max-pending-samples` is `-1`, or if it is greater than `0` and the circular buffer is not full, the sample is added to the collection.
- If `max-pending-samples` is greater than `0` and the circular buffer reaches its maximum capacity, the oldest sample with same type as the received one is popped, and either written without type (`only-with-type:  false`) or discarded (`only-with-type:  true`).

- If `max-pending-samples` is `0`, the message is written without type if `only-with-type: false`, and discarded otherwise.

Cleanup Period

As explained in *Event Window*, a *DDS Recorder* in paused mode awaits for an event command to write in disk all samples received in the last `event-window` seconds. To accomplish this, received samples are stored in memory until the aforementioned event is triggered and, in order to limit memory consumption, outdated (received more than `event-window` seconds ago) samples are removed from this buffer every `cleanup-period` seconds. By default, its value is equal to twice the `event-window`.

QoS

`specs` supports a `qos` **optional** tag to configure the default values of the *Topic QoS*.

Note: The *Topic QoS* configured in `specs` can be overwritten by the *Manual Topics*.

Logging

`specs` supports a `logging` **optional** tag to configure the *DDS Recorder* logs. Under the `logging` tag, users can configure the type of logs to display and filter the logs based on their content and category. When configuring the verbosity to `info`, all types of logs, including informational messages, warnings, and errors, will be displayed. Conversely, setting it to `warning` will only show warnings and errors, while choosing `error` will exclusively display errors. By default, the filter allows all errors to be displayed, while selectively permitting warning and informational messages from `DDSRECORDER` category.

Note: Configuring the logs via the Command-Line is still active and takes precedence over YAML configuration when both methods are used simultaneously.

Logging	Yaml tag	Description	Data type	Default value	Possible values
Verbosity	`verbosity`	Show messages of equal or higher importance.	*enum*	`error`	`info` / `warning` / `error`
Filter	`filter`	Regex to filter the category or message of the logs.	*string*	info : DDSRECORDER warning : DDSRECORDER error : ""	Regex string

Note: For the logs to function properly, the `-DLOG_INFO=ON` compilation flag is required.

The *DDS Recorder* prints the logs by default (warnings and errors in the standard error and infos in the standard output). The *DDS Recorder*, however, can also publish the logs in a DDS topic. To publish the logs, under the tag `publish`, set `enable: true` and set a `domain` and a `topic-name`. The type of the logs published is defined as follows:

LogEntry.idl

```
const long UNDEFINED = 0x10000000;
const long SAMPLE_LOST = 0x10000001;
const long TOPIC_MISMATCH_TYPE = 0x10000002;
const long TOPIC_MISMATCH_QOS = 0x10000003;
const long FAIL_MCAP_CREATION = 0x12000001;
const long FAIL_MCAP_WRITE = 0x12000002;

enum Kind {
  Info,
  Warning,
  Error
};

struct LogEntry {
  @key long event;
  Kind kind;
  string category;
  string message;
  string timestamp;
};
```

Note: The type of the logs can be published by setting `publish-type: true`.

Example of usage

```yaml
logging:
  verbosity: info
  filter:
    error: "DDSPIPE|DDSRECORDER"
    warning: "DDSPIPE|DDSRECORDER"
    info: "DDSRECORDER"
  publish:
    enable: true
    domain: 84
    topic-name: "DdsRecorderLogs"
    publish-type: false
  stdout: true
```

Monitor

`specs` supports a `monitor` **optional** tag to publish internal data from the *DDS Recorder*. If the monitor is enabled, it publishes (and logs under the `MONITOR_DATA` *log filter*) the *DDS Recorder's* internal data on a `domain`, under a `topic-name`, once every `period` (in milliseconds). If the monitor is not enabled, the *DDS Recorder* will not collect or publish any data.

Note: The data published is relative to each period. The *DDS Recorder* will reset its tracked data after publishing it.

In particular, the *DDS Recorder* can monitor its internal status and its topics. When monitoring its internal status, the *DDS Recorder* will track different errors of the *DDS Recorder*. The type of the data published is defined as follows:

DdsRecorderMonitoringStatus.idl

```
struct MonitoringErrorStatus {
    boolean type_mismatch;
    boolean qos_mismatch;
};

struct MonitoringStatus {
    MonitoringErrorStatus error_status;
    boolean has_errors;
};

struct DdsRecorderMonitoringErrorStatus {
    boolean mcap_file_creation_failure;
    boolean disk_full;
};

struct DdsRecorderMonitoringStatus : MonitoringStatus {
    DdsRecorderMonitoringErrorStatus ddsrecorder_error_status;
};
```

When monitoring its topics, the *DDS Recorder* will track the number of messages lost, received, and the message reception rate [Hz] of each topic. It will also track if a topic's type is discovered, if there is a type mismatch, and if there is a QoS mismatch. The type of the data published is defined as follows:

MonitoringTopics.idl

```
struct DdsTopicData
{
    string participant_id;
    unsigned long msgs_lost;
    unsigned long msgs_received;
    double msg_rx_rate;
};

struct DdsTopic
{
    string name;
    string type_name;
    boolean type_discovered;
    boolean type_mismatch;
    boolean qos_mismatch;
    sequence<DdsTopicData> data;
};

struct MonitoringTopics
{
    sequence<DdsTopic> topics;
};
```

Example of usage

```
monitor:
  domain: 10
```

(continues on next page)

(continued from previous page)

```yaml
  status:
    enable: true
    domain: 11
    period: 2000
    topic-name: "DdsRecorderStatus"

  topics:
    enable: true
    domain: 12
    period: 1500
    topic-name: "DdsRecorderTopics"
```

General Example

A complete example of all the configurations described on this page can be found below.

Warning: This example can be used as a quick reference, but it may not be correct due to incompatibility or exclusive properties. **Do not take it as a working example**.

```yaml
dds:
  domain: 0

  allowlist:
    - name: "topic_name"
      type: "topic_type"

  blocklist:
    - name: "topic_name"
      type: "topic_type"

  builtin-topics:
    - name: "HelloWorldTopic"
      type: "HelloWorld"

  topics:
    - name: "temperature/*"
      type: "temperature/types/*"
      qos:
        max-rx-rate: 15
        downsampling: 2

  ignore-participant-flags: no_filter
  transport: builtin
  whitelist-interfaces:
    - "127.0.0.1"

recorder:
  output:
    filename: "output"
```

(continues on next page)

(continued from previous page)

```yaml
    path: "."
    timestamp-format: "%Y-%m-%d_%H-%M-%S_%Z"
    local-timestamp: false

  buffer-size: 50
  event-window: 60
  log-publish-time: false
  only-with-type: false
  compression:
    algorithm: lz4
    level: slowest
    force: true
  record-types: true
  ros2-types: false

remote-controller:
  enable: true
  domain: 10
  initial-state: "PAUSED"
  command-topic-name: "/ddsrecorder/command"
  status-topic-name: "/ddsrecorder/status"

specs:
  threads: 8
  max-pending-samples: 10
  cleanup-period: 90

  qos:
    max-rx-rate: 20
    downsampling: 3

  logging:
    verbosity: info
    filter:
      error: "DDSPIPE|DDSRECORDER"
      warning: "DDSPIPE|DDSRECORDER"
      info: "DDSRECORDER"
    publish:
      enable: true
      domain: 84
      topic-name: "DdsRecorderLogs"
      publish-type: false
    stdout: true

  monitor:
    domain: 10
    topics:
      enable: true
      domain: 11
      period: 1000
      topic-name: "DdsRecorderTopics"
```

(continues on next page)

(continued from previous page)

```yaml
  status:
    enable: true
    domain: 12
    period: 2000
    topic-name: "DdsRecorderStatus"
```

3.10.2 Fast DDS Configuration

As explained in *this section*, a *DDS Recorder* instance stores (by default) all data regardless of whether their associated data type is received or not. Some applications rely on this information being recorded and written in the resulting MCAP file, which requires that the user application is configured to send the necessary type information. However, *Fast DDS* does not send the data type information by default, it must be configured to do so.

First of all, when generating the topic types using *eProsima Fast DDS Gen*, the option `-typeobject` must be added in order to generate the needed code to fill the `TypeObject` data.

For native types (data types that does not rely in other data types) this is enough, as *Fast DDS* will send the `TypeObject` by default. However, for more complex types, it is required to use `TypeInformation` mechanism. In the *Fast DDS* `DomainParticipant` set the following QoS in order to send this information:

```cpp
DomainParticipantQos pqos;
pqos.wire_protocol().builtin.typelookup_config.use_server = true;
```

Feel free to review *this* section, where it is explained in detail how to configure a Fast DDS Publisher/Subscriber leveraging *Dynamic Types*.

3.11 Remote Control

The *DDS Recorder* application from *eProsima DDS Record & Replay* allows remote control and monitoring of the tool via DDS. Thus it is possible both to monitor the execution status of the *DDS Recorder* and to control the execution status of this application.

Moreover, eProsima provides a remote controlling tool that allows to visualize the status of a *DDS Recorder* and to send commands to it to change its current status.

This section explains the different execution states of a *DDS Recorder*, how to create your own tool using the DDS topics that the application defines to control its behavior, and the presentation of the eProsima user application for the remote control of the *DDS Recorder*.

3.11.1 DDS Recorder Statuses

The *DDS Recorder* application may have the following states:

- **CLOSED**: The application is not running. To start running the application it is required to launch it from the terminal by executing `ddsrecorder`. Once the `ddsrecorder` application is executed, it will automatically go into recording mode (`RUNNING` state), although this can be modified through the *.yaml* configuration file. Please refer to the *DDS Recorder* *remote controller configuration section* for more options on the initial state of the application.
- **RUNNING**: The application is running and recording data in the database.

- **PAUSED**: The application is running but not recording data in the database. In this state, the application stores the data it has received in a time window prior to the current time. The data will not be saved to the database until an event arrives from the remote controller.
- **SUSPENDED**: The application is running but not recording data. Internal entities are created and samples received but discarded (advantage: lower latency in transition to `RUNNING/PAUSED` states).
- **STOPPED**: The application is running but not recording data. Internal entities are not created and thus no samples are received.

To change from one state to another, commands can be sent to the application through the *Controller Command* DDS topic to be defined later. The commands that the application accepts are as follows:

- **start**: Changes to `RUNNING` state if it was not in it.
- **pause**: Changes to `PAUSED` state if it was not in it.
- **event**: Triggers a recording event to save the data of the time window prior to the event. This command can take the next state as an argument, so it is possible to trigger an event and change the state with the same command. This is useful when the recorder is in a paused state, the user wants to record all the data collected in the current time window and then immediately switch to `RUNNING` state to start recording data. It could also be the case that the user wants to capture the event, save the data and then stop the recorder to inspect the output file. The arguments are sent as a serialized *json* in string format.
- **suspend**: Changes to `SUSPENDED` state if it was not in it.
- **stop**: Changes to `STOPPED` state if it was not in it.
- **close**: Closes the *DDS Recorder* application.

The following is the state diagram of the *DDS Recorder* application with all the available commands and the state change effect they cause.

3.11.2 DDS Controller Data Types

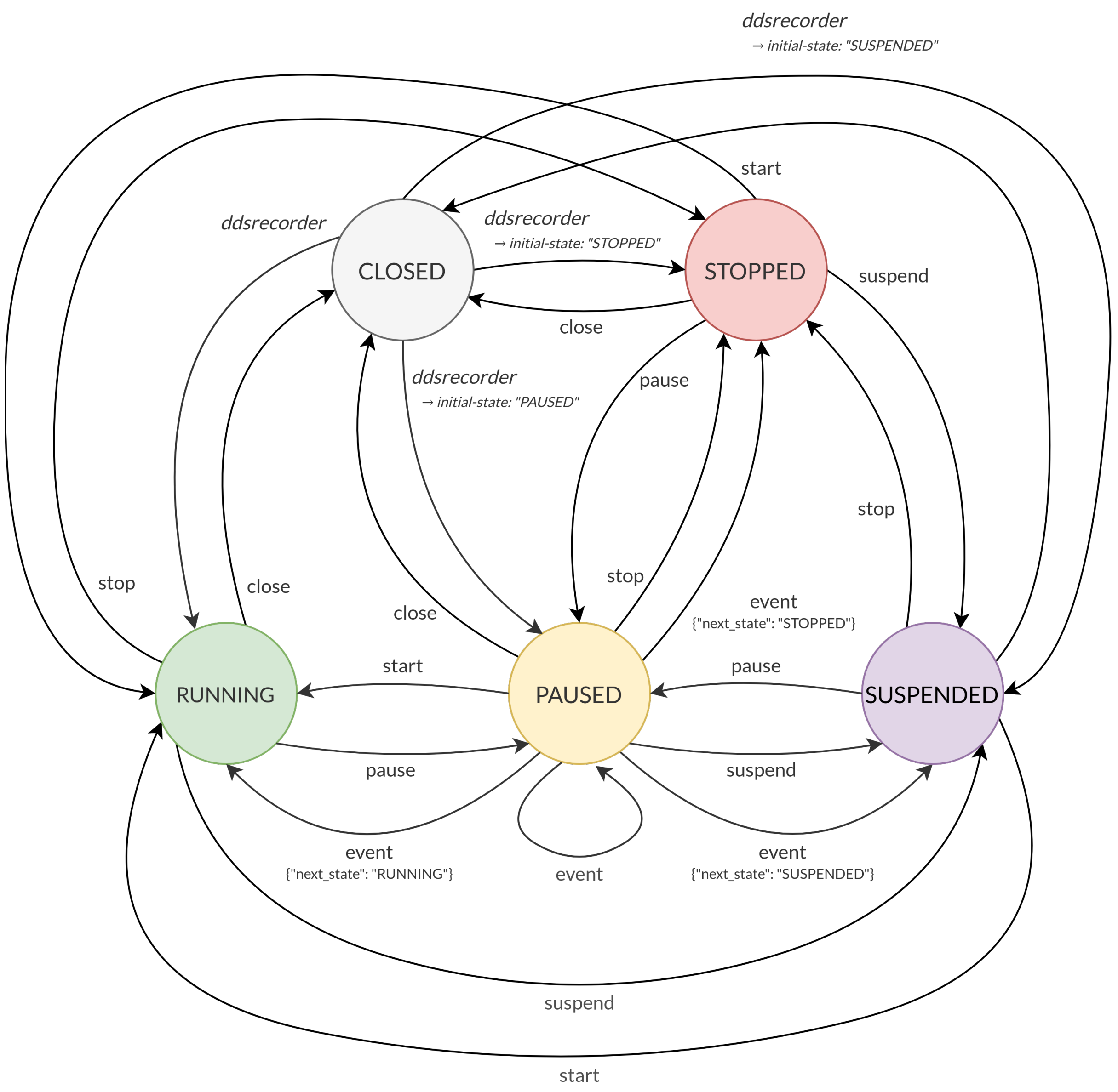
The *DDS Recorder* contains a DDS subscriber in the *Controller Command* topic and a DDS publisher in the *Controller Status* topic. These topics' names are by default `/ddsrecorder/command` and `/ddsrecorder/status`, respectively, but can also be specified by users via the `command-topic-name` and `status-topic-name` configuration tags. Therefore, any user can create his own application to control the *DDS Recorder* remotely by creating a publisher in the *Controller Command* topic, which sends commands to the recorder, and a subscriber in the *Controller Status* topic to monitor its status.

Note: Status and command topics are not blocked by default, i.e. messages on this topics will be recorded if listening on the same domain the controller is launched. If willing to avoid this, include these topics in the *blocklist*:

```
dds:
  blocklist:
    - type: DdsRecorderStatus
    - type: DdsRecorderCommand
```

The following is a description of the aforementioned control topics.

- Command topic:
  - Topic name: Specified in `command-topic-name` configuration parameter (Default: `/ddsrecorder/command`)
  - Topic type name: `DdsRecorderCommand`


ddsrecorder
→ initial-state: "SUSPENDED"
start
ddsrecorder
ddsrecorder
→ initial-state: "STOPPED"
CLOSED
STOPPED
suspend
close
ddsrecorder
→ initial-state: "PAUSED"
pause
stop
stop
close
close
stop
event
{"next_state": "STOPPED"}
start
pause
RUNNING
PAUSED
SUSPENDED
pause
suspend
event
{"next_state": "RUNNING"}
event
event
{"next_state": "SUSPENDED"}
suspend
start

- Type description:
  - IDL definition

```
struct DdsRecorderCommand
{
    string command;
    string args;
};
```

  - DdsRecorderCommand type description:

Argument	Description	Data type	Possible values
`command`	Command to send.	`string`	`start` `pause` `event` `suspend` `stop` `close`
`args`	Arguments of the command. This arguments should contain a JSON serialized string.	`string`	· `event` command: `{"next_state": "RUNNING"}` `{"next_state": "SUSPENDED"}` `{"next_state": "STOPPED"}`

- Status topic:
  - Topic name: Specified in `status-topic-name` configuration parameter (Default: `/ddsrecorder/status`)
  - Topic type name: `DdsRecorderStatus`
  - Type description:
    - IDL definition

```
struct DdsRecorderStatus
{
    string previous;
    string current;
    string info;
};
```

  - DdsRecorderStatus type description:

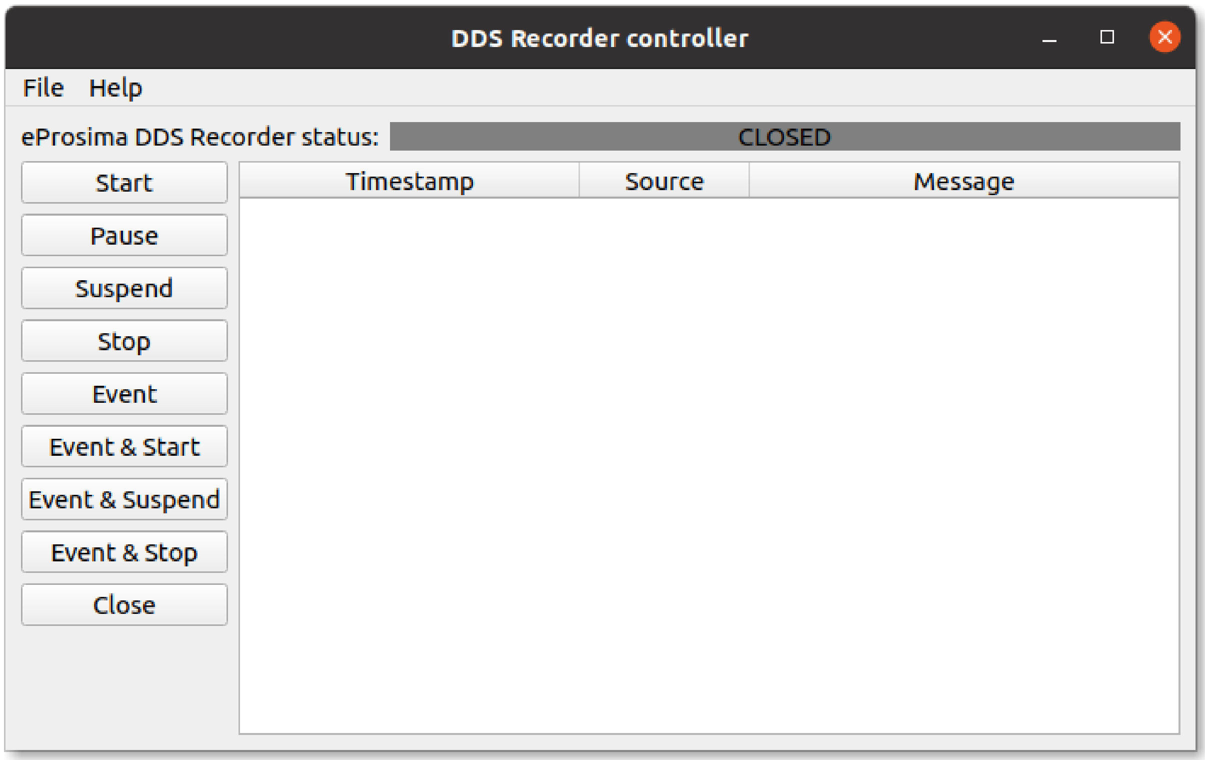
Argument	Description	Data type	Possible values
`previous`	Previous status of the *DDS Recorder*.	`string`	`RUNNING` `PAUSED` `SUSPENDED` `STOPPED`
`current`	Current status of the *DDS Recorder*.	`string`	`RUNNING` `PAUSED` `SUSPENDED` `STOPPED`
`info`	Additional information related to the state change. (Unused)	`string`	-

3.11.3 DDS Recorder remote controller application

eProsima DDS Record & Replay provides a graphical user application that implements a remote controller for the *DDS Recorder*. Thus the user can control a *DDS Recorder* instance using this application without having to implement their own.

Note: If installing *eProsima DDS Record & Replay* from sources, compilation flag `-DBUILD_DDSRECORDER_CONTROLLER=ON` is required to build this application.

Its interface is quite simple and intuitive. Once the application is launched, a layout as the following one should be visible:



If the controller should function in a domain different than the default one (`0`), change it by clicking `File->DDS Domain` and introducing the one desired:

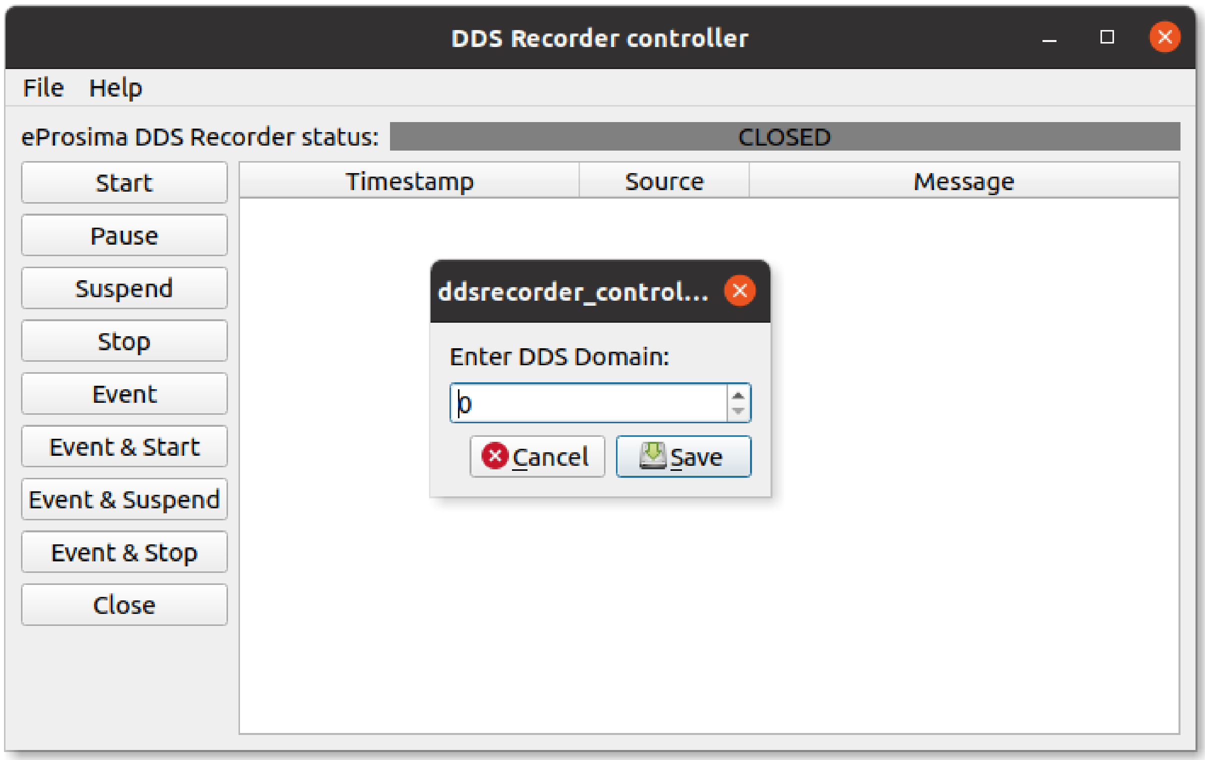
It is also possible to use non-default status and command topic names through the `File->DDS Topics` dialog.

When a *DDS Recorder* instance is found in the domain, a message is displayed in the logging panel:

From this point on, it is possible to interact with the recorder application by pushing any of the buttons appearing on the left. Every command sent is reflected in the logging panel and, additionally, the recorder application publishes its current status with every state transition undergone. This can be observed in the *eProsima DDS Recorder status* placeholder, located in the upper part of the layout:

By clicking on `Suspend` / `Stop` button, the recorder application ceases recording, but can be commanded to `Start` / `Pause` whenever wished. Once the user has finished all recording activity, it is possible to `Close` the recorder and free all resources used by the application:

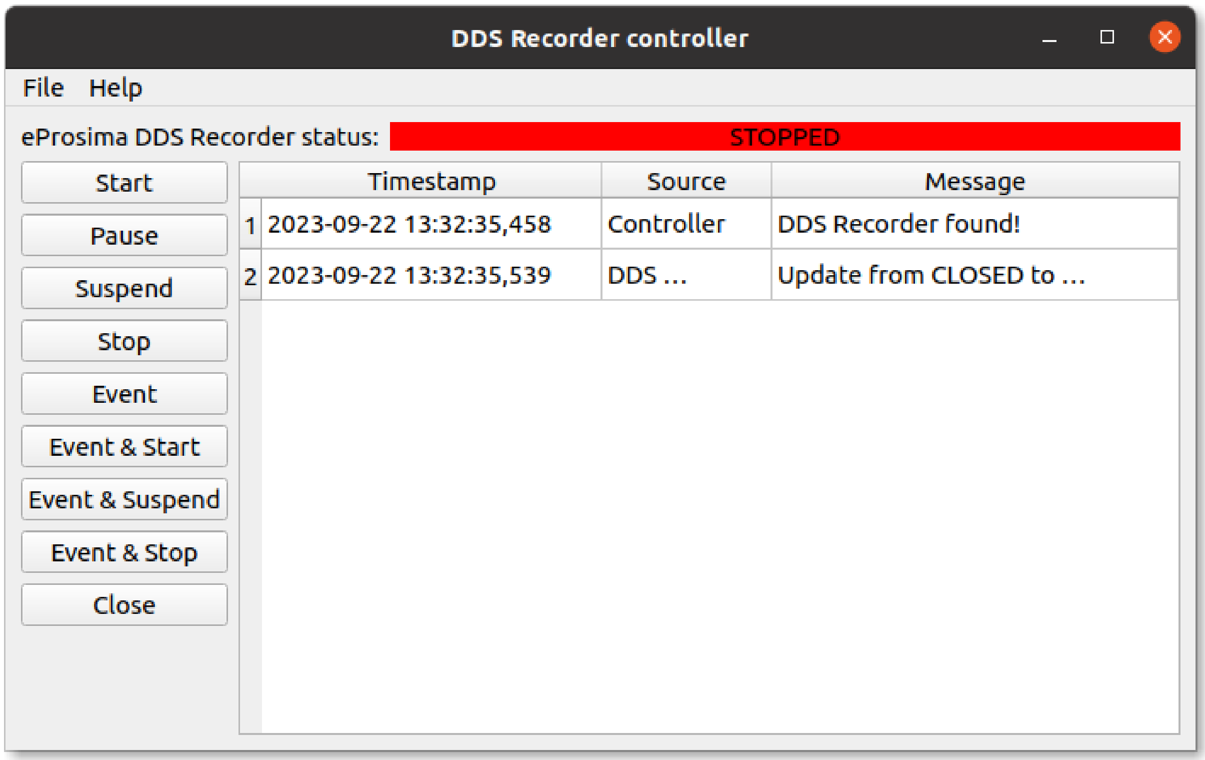
Note that once `CLOSED` state has been reached, commands will no longer have an effect on the recorder application as its process is terminated when a `close` command is received.



eProsima DDS Recorder status: CLOSED

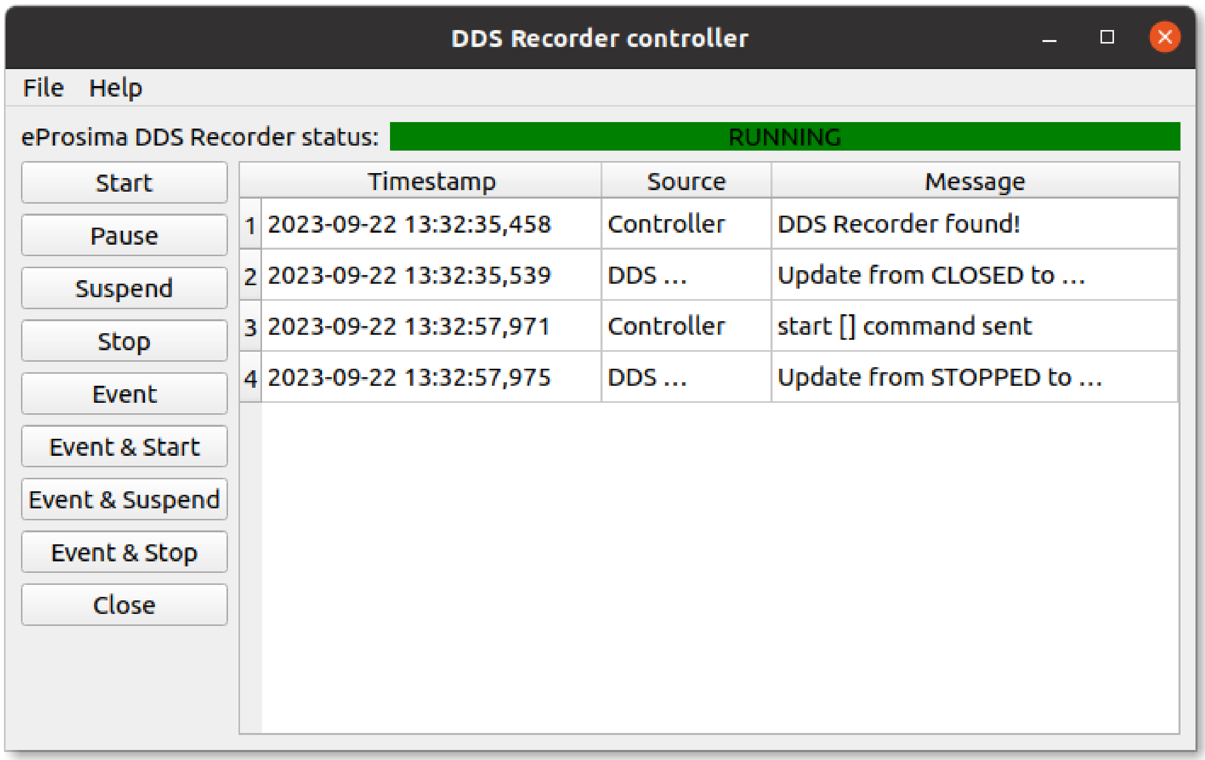
Timestamp	Source	Message

Enter DDS Domain: 0

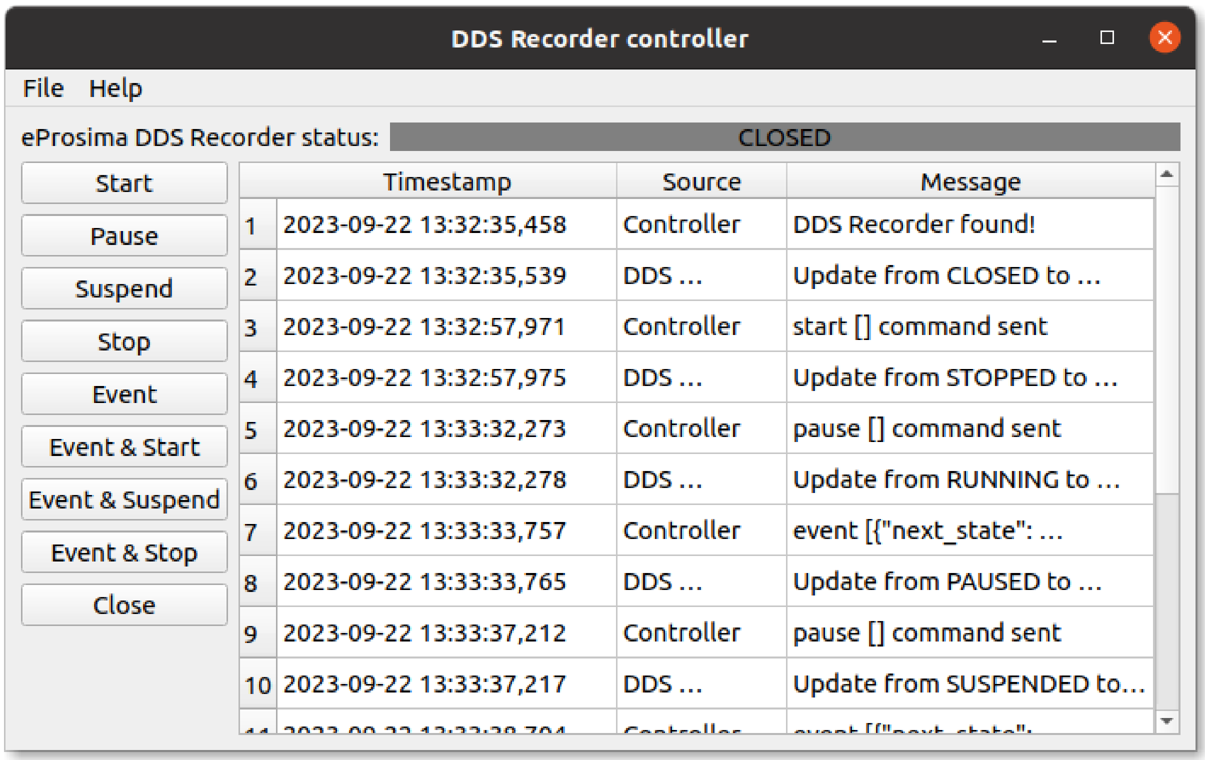


eProsima DDS Recorder status: STOPPED

	Timestamp	Source	Message
1	2023-09-22 13:32:35,458	Controller	DDS Recorder found!
2	2023-09-22 13:32:35,539	DDS ...	Update from CLOSED to ...



	Timestamp	Source	Message
1	2023-09-22 13:32:35,458	Controller	DDS Recorder found!
2	2023-09-22 13:32:35,539	DDS …	Update from CLOSED to …
3	2023-09-22 13:32:57,971	Controller	start [] command sent
4	2023-09-22 13:32:57,975	DDS …	Update from STOPPED to …



	Timestamp	Source	Message
1	2023-09-22 13:32:35,458	Controller	DDS Recorder found!
2	2023-09-22 13:32:35,539	DDS …	Update from CLOSED to …
3	2023-09-22 13:32:57,971	Controller	start [] command sent
4	2023-09-22 13:32:57,975	DDS …	Update from STOPPED to …
5	2023-09-22 13:33:32,273	Controller	pause [] command sent
6	2023-09-22 13:33:32,278	DDS …	Update from RUNNING to …
7	2023-09-22 13:33:33,757	Controller	event [{"next_state": …
8	2023-09-22 13:33:33,765	DDS …	Update from PAUSED to …
9	2023-09-22 13:33:37,212	Controller	pause [] command sent
10	2023-09-22 13:33:37,217	DDS …	Update from SUSPENDED to…
11	2023-09-22 13:33:38,704	Controller	event [{"next_state":

3.12 Getting Started

3.12.1 Project Overview

eProsima DDS Replayer is a cross-platform application that allows to play back messages recorded by a *DDS Recorder* instance.

A user can configure a *DDS Replayer* instance differently depending on the scenario and purpose, being able to tune parameters concerning the DDS layer as well as playback settings.

Among its many *configuration* options, the user is able to allow/block a set of topics, and/or define specific QoS (other than the recorded ones) to be applied to certain topics. It is also possible to publish samples at a rate different than the original one, filter messages according to its timestamp, or define a publication begin time, among others.

In addition, *eProsima DDS Replayer* is able to automatically send the type information recorded in a MCAP file, which might be required for applications relying on *Dynamic Types*.

Usage Description

DDS Replayer is a terminal (non-graphical) application that creates a replay service given an input data file. Although most use cases are covered by the default configuration, the *DDS Replayer* can be configured via a YAML file, whose format is very intuitive and human-readable.

- **Run**: Only the command that launches the application (`ddsreplayer`) needs to be executed to run a *DDS Replayer*. Please, read this *section* to apply a specific configuration, and this *section* to see the supported arguments.
- **Interact**: Once the *DDS Replayer* application is running, the allowlist and blocklist topic lists could be changed in runtime by just changing the YAML configuration file.
- **Close**: To close the *DDS Replayer* application just send a *Ctrl+C* signal to terminate the process gracefully (see *Closing Replay Application*).

Common Use cases

To get started with *DDS Replayer*, please visit section *Example of usage*.

3.12.2 Example of usage

This example will serve as a hands-on tutorial, aimed at introducing some of the key concepts and features that *eProsima DDS Record & Replay* replay application (*DDS Replayer* or `ddsreplayer`) has to offer.

Prerequisites

It is required to have *eProsima DDS Record & Replay* previously installed using one of the following installation methods:

- *DDS Record & Replay on Windows*
- *DDS Record & Replay on Linux*
- *Docker Image (recommended)*

Additionally, ShapesDemo is required to publish and subscribe shapes of different colors and sizes. Install it by following any of the methods described in the given links:

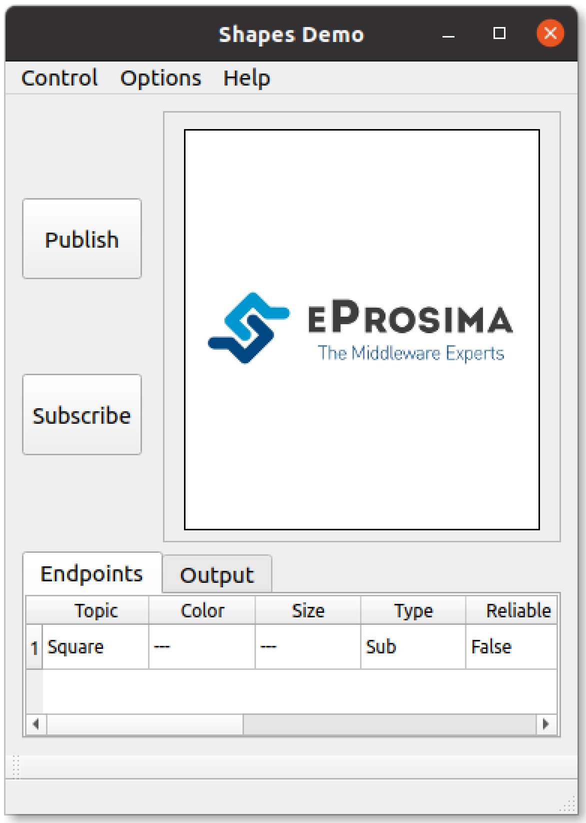
- Windows installation from binaries

- Linux installation from sources
- Docker Image

This is a follow-up tutorial, and assumes that *DDS Recorder* *Example of usage* has already been completed.

Start ShapesDemo

Let us launch a ShapesDemo instance and create a subscription in the `Square` topic with default settings.



	Topic	Color	Size	Type	Reliable
1	Square	---	---	Sub	False

Replayer configuration

The only configuration option required by a *DDS Replayer* is the path to an input MCAP file, which can be provided both as a CLI argument or via YAML configuration. By default, all messages stored in the provided input file are played back in DDS Domain `0`, starting at the very moment the application is launched.

It is also possible to change the default configuration parameters by means of a YAML configuration file.

Note: Please refer to *Configuration* for more information on how to configure a *DDS Replayer*.

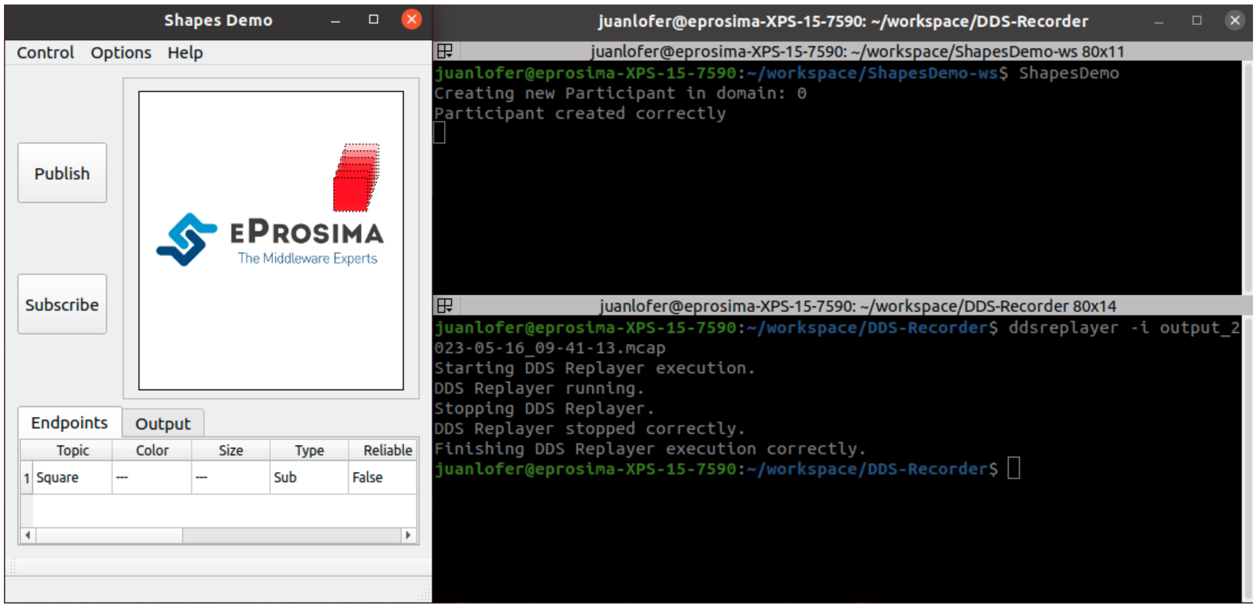
Replayer execution

Launching a *DDS Replayer* instance is as easy as executing the following command:

```
ddsreplayer -i output_YYYY-MM-DD-DD_hh-mm-ss.mcap
```

In order to know all the possible arguments supported by this tool, use the command:

```
ddsreplayer --help
```


Shapes Demo
Control
Options
Help
Publish
Subscribe
EPROSIMA
The Middleware Experts
Endpoints
Output
juanlofer@eprosima-XPS-15-7590: ~/workspace/DDS-Recorder
juanlofer@eprosima-XPS-15-7590: ~/workspace/ShapesDemo-ws 80x11
juanlofer@eprosima-XPS-15-7590: ~/workspace/DDS-Recorder 80x14

	Topic	Color	Size	Type	Reliable
1	Square	—	—	Sub	False

```
juanlofer@eprosima-XPS-15-7590:~/workspace/ShapesDemo-ws$ ShapesDemo
Creating new Participant in domain: 0
Participant created correctly
```

```
juanlofer@eprosima-XPS-15-7590:~/workspace/DDS-Recorder$ ddsreplayer -i output_2
023-05-16_09-41-13.mcap
Starting DDS Replayer execution.
DDS Replayer running.
Stopping DDS Replayer.
DDS Replayer stopped correctly.
Finishing DDS Replayer execution correctly.
juanlofer@eprosima-XPS-15-7590:~/workspace/DDS-Recorder$
```

Execution will end once every message found in the given input file is played back, although it can also be terminated with `Ctrl+C` at any point.

Next Steps

Feel free to experiment with the many *configuration* options available for a *DDS Replayer* instance. For example, you may try to modify the playback rate, block/allow the `Square` topic in the middle of execution, or set a different topic QoS configuration via the builtin-topics list.

3.13 Usage

eProsima DDS Replayer is a user application executed from command line.

3.13.1 Starting Replay Application

Docker Image

Warning: Currently, *DDS Record & Replay* Docker image only contains *DDS Recorder* tool, *DDS Replay tool* application will be added soon.

The recommended method to run the *DDS Replayer* is to instantiate a Docker container of the *DDS Record & Replay* image. *Here* are the instructions to download the compressed *DDS Record & Replay* Docker image and load it locally.

To run the *DDS Replayer* from a Docker container execute the following command:

```
docker run -it \
    --net=host \
    --ipc=host \
    -v /<dds_replayer_ws>/DDS_REPLAYER_CONFIGURATION.yaml:/root/DDS_REPLAYER_
↪CONFIGURATION.yaml \
    ubuntu-ddsrecorder:v<X.X.X> ddsreplayer
```

Installation from sources

eProsima DDS Record & Replay depends on `fastrtps`, `fastcdr` and `ddspipe` libraries. In order to correctly execute the replayer, make sure that `fastrtps`, `fastcdr` and `ddspipe` are properly sourced.

```
source <path-to-fastdds-installation>/install/setup.bash
source <path-to-ddspipe-installation>/install/setup.bash
source <path-to-ddsrecordreplay-installation>/install/setup.bash
```

Note: If Fast DDS, DDS Pipe and DDS Record & Replay have been installed in the system, these libraries would be sourced by default.

To start *eProsima DDS Replayer* with a default configuration, enter:

```
ddsreplayer -i input_file.mcap
```

3.13.2 Closing Replay Application

SIGINT

To close *eProsima DDS Replayer*, press `Ctrl+C`. *DDS Replayer* will perform a clean shutdown.

SIGTERM

Write command `kill <pid>` in a different terminal, where `<pid>` is the id of the process running the *DDS Replayer*. Use `ps` or `top` programs to check the process ids.

TIMEOUT

Setting a maximum amount of seconds that the application will work using argument `--timeout` will close the application once the time has expired.

3.13.3 Replay Service Command-Line Parameters

The *DDS Replayer* application supports several input arguments:

Command	Description	Option	Possible Values	Default Value
Help	It shows the usage information of the application.	`-h` `--help`		
Version	It shows the current version of the *DDS Replayer* and the hash of the last commit of the compiled code.	`-v` `--version`		
Input File	Input MCAP file path.	`-i` `--input-file`		
Configuration File	Configuration file path.	`-c` `--config-path`		`./DDS_REPLAYER_CONFIGURATION.yaml`
Reload Timer	The configuration file will be automatically reloaded according to the specified time period.	`-r` `--reload-time`	Unsigned Integer	`0`
Timeout	Set a maximum time while the application will be running. `0` means that the application will run forever (until kill via signal).	`-t` `--timeout`	Unsigned Integer	`0`
Debug	Enables the *DDS Replayer* logs so the execution can be followed by internal debugging information. Sets `Log Verbosity` to `info` and `Log Filter` to `DDSREPLAYER`.	`-d` `--debug`		
Log Verbosity	Set the verbosity level so only log messages with equal or higher importance level are shown.	`--log-verbosity`	`info` `warning` `error`	`warning`
Log Filter	Set a regex string as filter.	`--log-filter`	String	`"DDSREPLAYER"`

3.14 Configuration

3.14.1 DDS Replayer Configuration

A *DDS Replayer* is configured by a *.yaml* configuration file. This *.yaml* file contains all the information regarding the DDS interface configuration, playback parameters, and *DDS Replayer* specifications. Thus, this file has four major configuration groups:

- `dds`: configuration related to DDS communication.
- `replayer`: configuration with data playback parameters.
- `specs`: configuration of the internal operation of the *DDS Replayer*.

DDS Configuration

Configuration related to DDS communication.

DDS Domain

Tag `domain` configures the *Domain Id*.

```
domain: 101
```

Topic Filtering

The *DDS Replayer* automatically detects the topics that are being used in a DDS Network. The *DDS Replayer* then creates internal DDS *Writers* to replay the data published on each topic. The *DDS Replayer* allows filtering DDS *Topics* to allow users to configure the DDS *Topics* that must be replayed. These data filtering rules can be configured under the `allowlist` and `blocklist` tags. If the `allowlist` and `blocklist` are not configured, the *DDS Replayer* will replayed the data published on every topic it discovers. If both the `allowlist` and `blocklist` are configured and a topic appears in both of them, the `blocklist` has priority and the topic will be blocked.

Topics are determined by the tags `name` (required) and `type`, both of which accept wildcard characters.

Note: Placing quotation marks around values in a YAML file is generally optional, but values containing wildcard characters do require single or double quotation marks.

Consider the following example:

```yaml
allowlist:
  - name: AllowedTopic1
    type: Allowed

  - name: AllowedTopic2
    type: "*"

  - name: HelloWorldTopic
    type: HelloWorld

blocklist:
  - name: "*"
    type: HelloWorld
```

In this example, the data published in the topic `AllowedTopic1` with type `Allowed` and in the topic `AllowedTopic2` with any type will be replayed by the *DDS Replayer*. The data published in the topic `HelloWorldTopic` with type `HelloWorld` will be blocked, since the `blocklist` is blocking all topics with any name and with type `HelloWorld`.

Topic QoS

The following is the set of QoS that are configurable for a topic. For more information on topics, please read the Fast DDS Topic section.

Quality of Service	Yaml tag	Data type	Default value	QoS set
Reliability	`reliability`	*bool*	`false`	`RELIABLE` / `BEST_EFFORT`
Durability	`durability`	*bool*	`false`	`TRANSIENT_LOCAL` / `VOLATILE`
Ownership	`ownership`	*bool*	`false`	`EXCLUSIVE_OWNERSHIP_QOS` / `SHARED_OWNERSHIP_QOS`
Partitions	`partitions`	*bool*	`false`	Topic with / without partitions
Key	`keyed`	*bool*	`false`	Topic with / without key
History Depth	`history-depth`	*unsigned integer*	`5000`	*History Depth*
Max Transmission Rate	`max-tx-rate`	*float*	`0` (unlimited)	*Max Transmission Rate*

Warning: Manually configuring `TRANSIENT_LOCAL` durability may lead to incompatibility issues when the discovered reliability is `BEST_EFFORT`. Please ensure to always configure the `reliability` when configuring the `durability` to avoid the issue.

History Depth

The `history-depth` tag configures the history depth of the Fast DDS internal entities. By default, the depth of every RTPS History instance is `5000`, which sets a constraint on the maximum number of samples a *DDS Replayer* instance can deliver to late joiner Readers configured with `TRANSIENT_LOCAL` DurabilityQosPolicyKind. Its value should be decreased when the sample size and/or number of created endpoints (increasing with the number of topics) are big enough to cause memory exhaustion issues. If enough memory is available, however, the `history-depth` could be increased to deliver a greater number of samples to late joiners.

Max Transmission Rate

The `max-tx-rate` tag limits the frequency [Hz] at which samples are sent by discarding messages transmitted before `1/max-tx-rate` seconds have passed since the last sent message. It only accepts non-negative numbers. By default it is set to `0`; it sends samples at an unlimited transmission rate.

Manual Topics

A subset of QoS can be manually configured for a specific topic under the tag `topics`. The tag `topics` has a required `name` tag that accepts wildcard characters. It also has two optional tags: a `type` tag that accepts wildcard characters, and a `qos` tag with the QoS that the user wants to manually configure. If a `qos` is not manually configured, it will get its value by discovery.

Example of usage

```yaml
topics:
  - name: "temperature/*"
    type: "temperature/types/*"
    qos:
      max-tx-rate: 15
```

Note: The *Topic QoS* configured in the Manual Topics take precedence over the *Specs Topic QoS*.

Ignore Participant Flags

A set of discovery traffic filters can be defined in order to add an extra level of isolation. This configuration option can be set through the `ignore-participant-flags` tag:

```yaml
ignore-participant-flags: no_filter                          # No filter (default)
# or
ignore-participant-flags: filter_different_host              # Discovery traffic from␣
↪another host is discarded
# or
ignore-participant-flags: filter_different_process           # Discovery traffic from␣
↪another process on same host is discarded
# or
ignore-participant-flags: filter_same_process                # Discovery traffic from␣
↪own process is discarded
# or
ignore-participant-flags: filter_different_and_same_process  # Discovery traffic from␣
↪own host is discarded
```

(continues on next page)

(continued from previous page)

See Ignore Participant Flags for more information.

Custom Transport Descriptors

By default, *DDS Replayer* internal participants are created with enabled UDP and Shared Memory transport descriptors. The use of one or the other for communication will depend on the specific scenario, and whenever both are viable candidates, the most efficient one (Shared Memory Transport) is automatically selected. However, a user may desire to force the use of one of the two, which can be accomplished via the `transport` configuration tag.

```
transport: builtin    # UDP & SHM (default)
# or
transport: udp        # UDP only
# or
transport: shm        # SHM only
```

Warning: When configured with `transport: shm`, *DDS Replayer* will only communicate with applications using Shared Memory Transport exclusively (with disabled UDP transport).

Interface Whitelist

Optional tag `whitelist-interfaces` allows to limit the network interfaces used by UDP and TCP transport. This may be useful to only allow communication within the host (note: same can be done with *Ignore Participant Flags*). Example:

```
whitelist-interfaces:
  - "127.0.0.1"    # Localhost only
```

See Interface Whitelist for more information.

Replay Configuration

Configuration of data playback settings.

Input File

The path to the file, set through the `input-file` configuration tag. When the input file is specified both through CLI argument and YAML configuration file, the former takes precedence.

Begin Time

By default, all data stored in the provided MCAP file is played back. However, a user might be interested in only replaying data relative to a specific time frame. `begin-time` and `end-time` configuration options can be leveraged for this purpose, and their format is as follows:

Parameter	Tag	Description	Data type	Default value
Use local time zone	`local`	Whether to interpret the provided datetime as local (`true`) or as a Greenwich Mean Time (GMT/UTC +0) without Daylight Saving Time (DST) considerations (`false`).	`bool`	`true`
Datetime Format	`format`	Format followed by the provided datetime.	`string`	`"%Y-%m-%d_%H-%M-%S"`
Datetime	`datetime`	Datetime (seconds precision).	`string`	
Milliseconds	`milliseconds`	Milliseconds.	`integer`	`0`
Microseconds	`microseconds`	Microseconds.	`integer`	`0`
Nanoseconds	`nanoseconds`	Nanoseconds.	`integer`	`0`

Messages recorded/sent (see *Log Publish Time*) before `begin-time` will not be played back by a *DDS Replayer* instance.

End Time

As with `begin-time`, a user can discard messages recorded/sent after a specific timepoint set through the `end-time` tag, which follows the format described in *Begin Time*.

Start Replay Time

This configuration option (`start-replay-time`) allows to start replaying data at a certain timepoint following the format described in *Begin Time*. If the provided timepoint already expired, the replayer starts publishing messages right away.

Playback Rate

By default, data is replayed at the same rate it was published/received. However, a user might be interested in playing messages back at a rate different than the original one. This can be accomplished through the playback `rate` tag, which accepts positive float values (e.g. 0.5 <–> half speed || 2 <–> double speed).

Replay Types

By default, a *DDS Replayer* instance automatically sends all type information found in the provided MCAP file, which might be required for applications relying on *Dynamic Types*. Nonetheless, a user can choose to avoid this by setting `replay-types: false`, so only data samples are sent while their associated type information is disregarded.

Specs Configuration

The internals of a *DDS Replayer* can be configured using the `specs` optional tag that contains certain options related with the overall configuration of the *DDS Replayer* instance to run. The values available to configure are:

Number of Threads

`specs` supports a `threads` optional value that allows the user to set a maximum number of threads for the internal `ThreadPool`. This ThreadPool allows to limit the number of threads spawned by the application. This improves the performance of the internal data communications.

This value should be set by each user depending on each system characteristics. In case this value is not set, the default number of threads used is `12`.

Wait-for-acknowledgement Timeout

The execution of a *DDS Replayer* instance ends when the last message contained in the provided input file is published (or the user manually aborts the process, see *Closing Replay Application*). Note that this last message might be lost after publication, and if reliable Reliability QoS is being used, a mechanism should be established to avoid this problematic situation. For this purpose, the user can specify the maximum amount of milliseconds (`wait-all-acked-timeout`) to wait on closure until published messages are acknowledged by matched readers. Its value is set to `0` by default (no wait).

QoS

`specs` supports a `qos` **optional** tag to configure the default values of the *Topic QoS*.

Note: The *Topic QoS* configured in `specs` can be overwritten by the *Manual Topics*.

Logging

`specs` supports a `logging` **optional** tag to configure the *DDS Replayer* logs. Under the `logging` tag, users can configure the type of logs to display and filter the logs based on their content and category. When configuring the verbosity to `info`, all types of logs, including informational messages, warnings, and errors, will be displayed. Conversely, setting it to `warning` will only show warnings and errors, while choosing `error` will exclusively display errors. By default, the filter allows all errors to be displayed, while selectively permitting warning and informational messages from `DDSREPLAYER` category.

```yaml
logging:
  verbosity: info
  filter:
```

(continues on next page)

(continued from previous page)

```
    error: "DDSPIPE|DDSREPLAYER"
    warning: "DDSPIPE|DDSREPLAYER"
    info: "DDSREPLAYER"
```

Note: Configuring the logs via the Command-Line is still active and takes precedence over YAML configuration when both methods are used simultaneously.

Logging	Yaml tag	Description	Data type	Default value	Possible values
Verbosity	`verbosity`	Show messages of equal or higher importance.	*enum*	`error`	`info` / `warning` / `error`
Filter	`filter`	Regex to filter the category or message of the logs.	*string*	info : DDSREPLAYER warning : DDSREPLAYER error : ""	Regex string

Note: For the logs to function properly, the `-DLOG_INFO=ON` compilation flag is required.

The *DDS Replayer* prints the logs by default (warnings and errors in the standard error and infos in the standard output). The *DDS Replayer*, however, can also publish the logs in a DDS topic. To publish the logs, under the tag `publish`, set `enable: true` and set a `domain` and a `topic-name`. The type of the logs published is defined as follows:

LogEntry.idl

```
const long UNDEFINED = 0x10000000;
const long SAMPLE_LOST = 0x10000001;
const long TOPIC_MISMATCH_TYPE = 0x10000002;
const long TOPIC_MISMATCH_QOS = 0x10000003;

enum Kind {
  Info,
  Warning,
  Error
};

struct LogEntry {
  @key long event;
  Kind kind;
  string category;
  string message;
  string timestamp;
};
```

Note: The type of the logs can be published by setting `publish-type: true`.

Example of usage

```yaml
logging:
  verbosity: info
  filter:
    error: "DDSPIPE|FASTDDSSPY"
    warning: "DDSPIPE|FASTDDSSPY"
    info: "FASTDDSSPY"
  publish:
    enable: true
    domain: 84
    topic-name: "FastDdsSpyLogs"
    publish-type: false
  stdout: true
```

General Example

A complete example of all the configurations described on this page can be found below.

Warning: This example can be used as a quick reference, but it may not be correct due to incompatibility or exclusive properties. **Do not take it as a working example**.

```yaml
dds:
  domain: 0

  allowlist:
    - name: "topic_name"
      type: "topic_type"

  blocklist:
    - name: "topic_name"
      type: "topic_type"

  topics:
    - name: "temperature/*"
      type: "temperature/types/*"
      qos:
        max-tx-rate: 15

  ignore-participant-flags: no_filter
  transport: builtin
  whitelist-interfaces:
    - "127.0.0.1"

replayer:
  input-file: my_input.mcap

  begin-time:
    local: true
    datetime: 2023-04-10_10-37-50
    milliseconds: 100
    nanoseconds: 50
```

(continues on next page)

(continued from previous page)

```
  end-time:
    format: "%H-%M-%S_%Y-%m-%d"
    local: true
    datetime: 10-39-11_2023-04-10
    milliseconds: 200

  start-replay-time:
    local: true
    datetime: 2023-04-12_12-00-00
    milliseconds: 500

  rate: 1.4
  replay-types: true

specs:
  threads: 8
  wait-all-acked-timeout: 10

  qos:
    max-tx-rate: 20

  logging:
    verbosity: info
    filter:
      error: "DDSPIPE|DDSREPLAYER"
      warning: "DDSPIPE|DDSREPLAYER"
      info: "DDSREPLAYER"
    publish:
      enable: true
      domain: 84
      topic-name: "FastDdsSpyLogs"
      publish-type: false
    stdout: true
```

3.15 Configuring Fast DDS DynamicTypes

3.15.1 Background

As explained in *this section*, a *DDS Recorder* instance stores (by default) all data regardless of whether their associated data type is received or not. However, some applications require this information to be recorded and written in the resulting MCAP file, and for this to occur the publishing applications must send it via *Dynamic Types*.

This tutorial focuses on how to send the data type information using Fast DDS DynamicTypes and other relevant aspects of DynamicTypes. More specifically, this tutorial implements a DDS Publisher configured to send its data type, a DDS Subscriber that collects the data type and is able to read the incoming data, and a DDS Recorder is launched to save all the data published on the network. For more information about how to create the workspace with a basic DDS Publisher and a basic DDS Subscriber, please refer to Writing a simple C++ publisher and subscriber application .

The source code of this tutorial can be found in the public *eProsima DDS Record & Replay* GitHub repository with an explanation of how to build and run it.

Warning: This tutorial works with this branch of Fast DDS.

3.15.2 Prerequisites

Ensure that *eProsima DDS Record & Replay* is installed together with *eProsima* dependencies, i.e. *Fast DDS*, *Fast CDR* and *DDS Pipe*.

If *eProsima DDS Record & Replay* was installed using the recommended installation the environment is sourced by default, otherwise, just remember to source it in every terminal in this tutorial:

```
source <path-to-fastdds-installation>/install/setup.bash
source <path-to-ddspipe-installation>/install/setup.bash
source <path-to-ddsrecordreplay-installation>/install/setup.bash
```

3.15.3 Generating data types

eProsima Fast DDS-Gen is a Java application that generates *eProsima Fast DDS* source code using the data types defined in an IDL (Interface Definition Language) file. When generating the Types using *eProsima Fast DDS Gen*, the option `-typeobject` must be added in order to generate the needed code to fill the `TypeInformation` data.

The expected argument list of the application is:

```
fastddsgen -typeobject MyType.idl
```

3.15.4 DDS Publisher

The DDS publisher will be configured to act as a server of the data types of the data it publishes.

However, *Fast DDS* does not send the data type information by default, it must be configured to do so.

Data types

At the moment, there are two data types that can be used:

- HelloWorld.idl

```
struct HelloWorld
{
    unsigned long index;
    string message;
};
```

- Complete.idl

```
struct Timestamp
{
    long seconds;
    long milliseconds;
};

struct Point
{
    long x;
    long y;
    long z;
};

struct MessageDescriptor
{
    unsigned long id;
    string topic;
    Timestamp time;
};

struct Message
{
    MessageDescriptor descriptor;
    string message;
};

struct CompleteData
{
    unsigned long index;
    Point main_point;
    sequence<Point> internal_data;
    Message messages[2];
};
```

Examining the code

This section explains the C++ source code of the DDS Publisher, which can also be found here.

The private data members of the class defines the DDS Topic, `DataTypeKind`, DDS Topic type and DynamicType. The `DataTypeKind` defines the type to be used by the application (`HelloWorld` or `Complete`). For simplicity, this tutorial only covers the code related to the `HelloWorld` type.

```
    //! Name of the DDS Topic
    std::string topic_name_;
    //! The user can choose between HelloWorld and Complete types so this defines the␣
↪chosen type
    DataTypeKind data_type_kind_;
    //! Name of the DDS Topic type according to the DataTypeKind
    std::string data_type_name_;
    //! Actual DynamicType generated according to the DataTypeKind
    eprosima::fastrtps::types::DynamicType_ptr dynamic_type_;
```

The next lines show the constructor of the `TypeLookupServicePublisher` class that implements the publisher. The publisher is created with the topic and data type to use.

```
TypeLookupServicePublisher::TypeLookupServicePublisher(
        const std::string& topic_name,
        const uint32_t domain,
        DataTypeKind data_type_kind)
    : participant_(nullptr)
    , publisher_(nullptr)
    , topic_(nullptr)
    , datawriter_(nullptr)
    , topic_name_(topic_name)
    , data_type_kind_(data_type_kind)
```

Inside the `TypeLookupServicePublisher` constructor are defined the DomainParticipantQos. As the publisher act as a server of types, its QoS must be configured to send this information. Set `use_client` to `false` and `use_server` to `true`.

```
    DomainParticipantQos pqos;
    pqos.name("TypeLookupService_Participant_Publisher");

    pqos.wire_protocol().builtin.typelookup_config.use_client = false;
    pqos.wire_protocol().builtin.typelookup_config.use_server = true;
```

Next, we register the type in the participant:

1. Generate the dynamic type through `generate_helloworld_type_()` explained below.
2. Set the data type.
3. Create the `TypeSupport` with the dynamic type previously created.
4. Configure the `type` to fill automatically the `TypeInformation` and not `TypeObject` to be compliant with DDS-XTypes 1.2. standard.

```
    switch (data_type_kind_)
    {
        case DataTypeKind::HELLO_WORLD:
```

(continues on next page)

(continued from previous page)

```
            dynamic_type_ = generate_helloworld_type_();
            data_type_name_ = HELLO_WORLD_DATA_TYPE_NAME;
            break;
        case DataTypeKind::COMPLETE:
            dynamic_type_ = generate_complete_type_();
            data_type_name_ = COMPLETE_DATA_TYPE_NAME;
            break;
        default:
            throw std::runtime_error("Not recognized DynamicType kind");
            break;
    }

    TypeSupport type(new eprosima::fastrtps::types::DynamicPubSubType(dynamic_type_));

    // Send type information so the type can be discovered
    type->auto_fill_type_information(true);
    type->auto_fill_type_object(false);

    // Register the type in the Participant
    participant_->register_type(type);
```

The function `generate_helloworld_type_()` returns the dynamic type generated with the `TypeObject` and `TypeIdentifier` of the type.

```
eprosima::fastrtps::types::DynamicType_ptr
    TypeLookupServicePublisher::generate_helloworld_type_() const
{
    // Generate HelloWorld type using methods from Fast DDS Gen autogenerated code
    registerHelloWorldTypes();

    // Get the complete type object and type identifier of the dynamic type
    auto type_object = GetHelloWorldObject(true);
    auto type_id = GetHelloWorldIdentifier(true);

    // Use data type name, type identifier and type object to build the dynamic type
    return eprosima::fastrtps::types::TypeObjectFactory::get_instance()->build_dynamic_
↪type(
            HELLO_WORLD_DATA_TYPE_NAME,
            type_id,
            type_object);
}
```

Then we initialized the Publisher, DDS Topic and DDS DataWriter.

To make the publication, the public member function `publish()` is implemented:

1. It creates the variable that will contain the user data, `dynamic_data_`.
2. Fill that variable with the function `fill_helloworld_data_(msg)`, explained below.

```
void TypeLookupServicePublisher::publish(unsigned int msg_index)
{
    // Get the dynamic data depending on the data type
    eprosima::fastrtps::types::DynamicData_ptr dynamic_data_;
```

(continues on next page)

(continued from previous page)

```
    switch (data_type_kind_)
    {
    case DataTypeKind::HELLO_WORLD:
        dynamic_data_ = fill_helloworld_data_(msg_index);
        break;
    case DataTypeKind::COMPLETE:
        dynamic_data_ = fill_complete_data_(msg_index);
        break;

    default:
        throw std::runtime_error("Not recognized DynamicType kind");
        break;
    }

    // Publish data
    datawriter_->write(dynamic_data_.get());

    // Print the message published
    std::cout << "Message published: " << std::endl;
    eprosima::fastrtps::types::DynamicDataHelper::print(dynamic_data_);
    std::cout << "-----------------------------------------------------" << std::endl;
}
```

The function `fill_helloworld_data_()` returns the data to be sent with the information filled in.

First, the `Dynamic_ptr` that will be filled in and returned is created. Using the `DynamicDataFactory` we create the data that corresponds to our data type. Finally, data variables are assigned, in this case, `index` and `message`.

```
eprosima::fastrtps::types::DynamicData_ptr
    TypeLookupServicePublisher::fill_helloworld_data_(
        const unsigned int& index)
{
    // Create and initialize new dynamic data
    eprosima::fastrtps::types::DynamicData_ptr new_data;
    new_data = eprosima::fastrtps::types::DynamicDataFactory::get_instance()->create_
↪data(dynamic_type_);

    // Set index
    new_data->set_uint32_value(index, 0);
    // Set message
    new_data->set_string_value("Hello World", 1);

    return new_data;
}
```

3.15.5 DDS Subscriber

The DDS Subscriber is acting as a client of types, i.e. the subscriber will not know the types beforehand and it will discovery the data type via the type lookup service implemented on the publisher side.

Examining the code

This section explains the C++ source code of the DDS Subscriber, which can also be found here.

The private data members of the class defines the DDS Topic, DDS Topic type and DynamicType.

```
    //! Name of the DDS Topic
    std::string topic_name_;
    //! Name of the received DDS Topic type
    std::string type_name_;
    //! DynamicType generated with the received type information
    eprosima::fastrtps::types::DynamicType_ptr dynamic_type_;
```

The next lines show the constructor of the `TypeLookupServiceSubscriber` class that implements the subscriber setting the topic name as the one configured in the publisher side.

```
TypeLookupServiceSubscriber::TypeLookupServiceSubscriber(
        const std::string& topic_name,
        uint32_t domain)
    : participant_(nullptr)
    , subscriber_(nullptr)
    , topic_(nullptr)
    , datareader_(nullptr)
    , topic_name_(topic_name)
    , samples_(0)
```

The `DomainParticipantQos` are defined inside the `TypeLookupServiceSubscriber` constructor. As the subscriber act as a client of types, set the QoS in order to receive this information. Set `use_client` to `true` and `use_server` to `false`.

```
    DomainParticipantQos pqos;
    pqos.name("TypeLookupService_Participant_Subscriber");

    pqos.wire_protocol().builtin.typelookup_config.use_client = true;
    pqos.wire_protocol().builtin.typelookup_config.use_server = false;
```

Then, the Subscriber is initialized.

Inside `on_data_available()` callback function the `DynamicData_ptr` is created, which will be filled with the actual data received.

As in the subscriber, the `DynamicDataFactory` is used for the creation of the data that corresponds to our data type.

```
void TypeLookupServiceSubscriber::on_data_available(
        DataReader* reader)
{
    // Create a new DynamicData to read the sample
    eprosima::fastrtps::types::DynamicData_ptr new_dynamic_data;
    new_dynamic_data = eprosima::fastrtps::types::DynamicDataFactory::get_instance()->
↪create_data(dynamic_type_);
```

(continues on next page)

(continued from previous page)

```
    SampleInfo info;

    // Take next sample until we've read all samples or the application stopped
    while ((reader->take_next_sample(new_dynamic_data.get(), &info) == ReturnCode_
↪t::RETCODE_OK) && !is_stopped())
    {
        if (info.instance_state == ALIVE_INSTANCE_STATE)
        {
            samples_++;

            std::cout << "Message " << samples_ << " received:\n" << std::endl;
            eprosima::fastrtps::types::DynamicDataHelper::print(new_dynamic_data);
            std::cout << "-----------------------------------------------------" <<␣
↪std::endl;

            // Stop if all expecting messages has been received (max_messages number␣
↪reached)
            if (max_messages_ > 0 && (samples_ >= max_messages_))
            {
                stop();
            }
        }
    }
}
```

The function on_type_information_received() detects if new topic information has been received in order to proceed to register the topic in case it has the same name as the expected one. To register a remote topic, function register_remote_type_callback_() is used. Once the topic has been discovered and registered, it is created a DataReader on this topic.

```
void TypeLookupServiceSubscriber::on_type_information_received(
        eprosima::fastdds::dds::DomainParticipant*,
        const eprosima::fastrtps::string_255 topic_name,
        const eprosima::fastrtps::string_255 type_name,
        const eprosima::fastrtps::types::TypeInformation& type_information)
{
    // First check if the topic received is the one we are expecting
    if (topic_name.to_string() != topic_name_)
    {
        std::cout <<
            "Discovered type information from topic < " << topic_name.to_string() <<
            " > while expecting < " << topic_name_ << " >. Skipping..." << std::endl;
        return;
    }

    // Set the topic type as discovered
    bool already_discovered = type_discovered_.exchange(true);
    if (already_discovered)
    {
        return;
    }
```

(continues on next page)

(continued from previous page)

```
    std::cout <<
        "Found type in topic < " << topic_name_ <<
        " > with name < " << type_name.to_string() <<
        " > by lookup service. Registering..." << std::endl;

    // Create the callback to register the remote dynamic type
    std::function<void(const std::string&, const eprosima::fastrtps::types::DynamicType_
↪ptr)> callback(
            [this]
            (const std::string& name, const eprosima::fastrtps::types::DynamicType_ptr␣
↪type)
            {
                this->register_remote_type_callback_(name, type);
            });

    // Register the discovered type and create a DataReader on this topic
    participant_->register_remote_type(
        type_information,
        type_name.to_string(),
        callback);
}
```

The function `register_remote_type_callback_()`, which is in charge of register the topic received, is explained below. First, it creates a `TypeSupport` with the corresponding type and registers it into the participant. Then, it creates the DDS Topic with the topic name set in the creation of the Subscriber and the topic type previously registered. Finally, it creates the DataReader of that topic.

```
void TypeLookupServiceSubscriber::register_remote_type_callback_(
        const std::string&,
        const eprosima::fastrtps::types::DynamicType_ptr dynamic_type)
{
    ////////////////////
    // Register the type
    TypeSupport type(new eprosima::fastrtps::types::DynamicPubSubType(dynamic_type));
    type.register_type(participant_);

    ///////////////////////
    // Create the DDS Topic
    topic_ = participant_->create_topic(
            topic_name_,
            dynamic_type->get_name(),
            TOPIC_QOS_DEFAULT);

    if (topic_ == nullptr)
    {
        return;
    }

    ////////////////////////
    // Create the DataReader
    datareader_ = subscriber_->create_datareader(
```

(continues on next page)

(continued from previous page)

```
            topic_,
            DATAREADER_QOS_DEFAULT,
            this);
```

3.15.6 Running the application

Open two terminals:

- In the first terminal, run the DDS Publisher:

  ```
  source install/setup.bash
  cd DDS-Record-Replay/build/TypeLookupService
  ./TypeLookupService --entity publisher
  ```

- In the second terminal, run the DDS Subscriber:

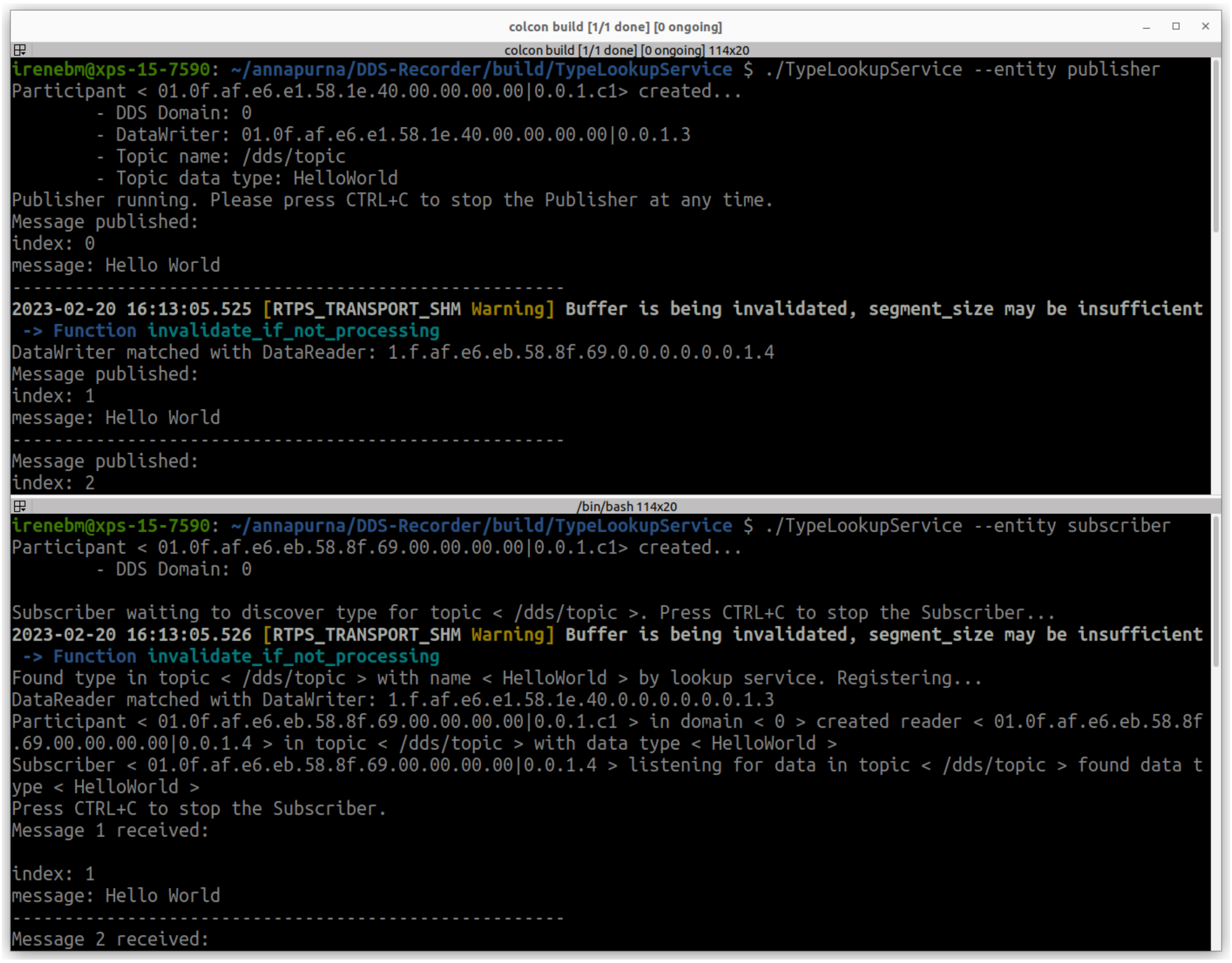
```
source install/setup.bash
cd DDS-Record-Replay/build/TypeLookupService
./TypeLookupService --entity subscriber
```

At this point, we observe that the data published reach the subscriber and it can access to the content of the sample received.

3.16 Visualize data with Foxglove

3.16.1 Background

This tutorial explains how to record data with *DDS Recorder* tool and visualize it with Foxglove Studio.


colcon build [1/1 done] [0 ongoing]
colcon build [1/1 done] [0 ongoing] 114x20
/bin/bash 114x20

```
irenebm@xps-15-7590: ~/annapurna/DDS-Recorder/build/TypeLookupService $ ./TypeLookupService --entity publisher
Participant < 01.0f.af.e6.e1.58.1e.40.00.00.00.00|0.0.1.c1> created...
        - DDS Domain: 0
        - DataWriter: 01.0f.af.e6.e1.58.1e.40.00.00.00.00|0.0.1.3
        - Topic name: /dds/topic
        - Topic data type: HelloWorld
Publisher running. Please press CTRL+C to stop the Publisher at any time.
Message published:
index: 0
message: Hello World
-----------------------------------------------------
2023-02-20 16:13:05.525 [RTPS_TRANSPORT_SHM Warning] Buffer is being invalidated, segment_size may be insufficient
 -> Function invalidate_if_not_processing
DataWriter matched with DataReader: 1.f.af.e6.eb.58.8f.69.0.0.0.0.0.0.1.4
Message published:
index: 1
message: Hello World
-----------------------------------------------------
Message published:
index: 2
```

```
irenebm@xps-15-7590: ~/annapurna/DDS-Recorder/build/TypeLookupService $ ./TypeLookupService --entity subscriber
Participant < 01.0f.af.e6.eb.58.8f.69.00.00.00.00|0.0.1.c1> created...
        - DDS Domain: 0

Subscriber waiting to discover type for topic < /dds/topic >. Press CTRL+C to stop the Subscriber...
2023-02-20 16:13:05.526 [RTPS_TRANSPORT_SHM Warning] Buffer is being invalidated, segment_size may be insufficient
 -> Function invalidate_if_not_processing
Found type in topic < /dds/topic > with name < HelloWorld > by lookup service. Registering...
DataReader matched with DataWriter: 1.f.af.e6.e1.58.1e.40.0.0.0.0.0.0.1.3
Participant < 01.0f.af.e6.eb.58.8f.69.00.00.00.00|0.0.1.c1 > in domain < 0 > created reader < 01.0f.af.e6.eb.58.8f
.69.00.00.00.00|0.0.1.4 > in topic < /dds/topic > with data type < HelloWorld >
Subscriber < 01.0f.af.e6.eb.58.8f.69.00.00.00.00|0.0.1.4 > listening for data in topic < /dds/topic > found data t
ype < HelloWorld >
Press CTRL+C to stop the Subscriber.
Message 1 received:

index: 1
message: Hello World
-----------------------------------------------------
Message 2 received:
```

3.16.2 Prerequisites

It is required to have *eProsima DDS Record & Replay* previously installed using one of the following installation methods:

- *DDS Record & Replay on Windows*
- *DDS Record & Replay on Linux*
- *Docker Image (recommended)*

Additionally, we will use ShapesDemo as a DDS Demo application to publish the data that will be recorded. This application is already prepared to use Fast DDS DynamicTypes, which is required when using the *DDS Recorder* tool. Download *eProsima Shapes Demo* from eProsima website or install it by following any of the methods described in the given links:

- Windows installation from binaries
- Linux installation from sources
- Docker Image

3.16.3 Configuring DDS Recorder

The DDS Recorder runs with default configuration parameters, but can also be configured via a YAML file. In this tutorial we will use a configuration file to change some default parameters and show how this file is loaded. The configuration file to be used is the following:

```yaml
dds:
  domain: 0

recorder:
  output:
    filename: "shapesdemo_data"
    path: "."
```

The previous configuration file configures a recorder in DDS Domain `0` and save the output file as `shapesdemo_data_<YYYY-MM-DD_hh-mm-ss>.mcap`, being `<YYYY-MM-DD_hh-mm-ss>` the timestamp of the time at which the *DDS Recorder* started recording.

Create a new file named `conf.yaml` and copy the above snippet into this file.

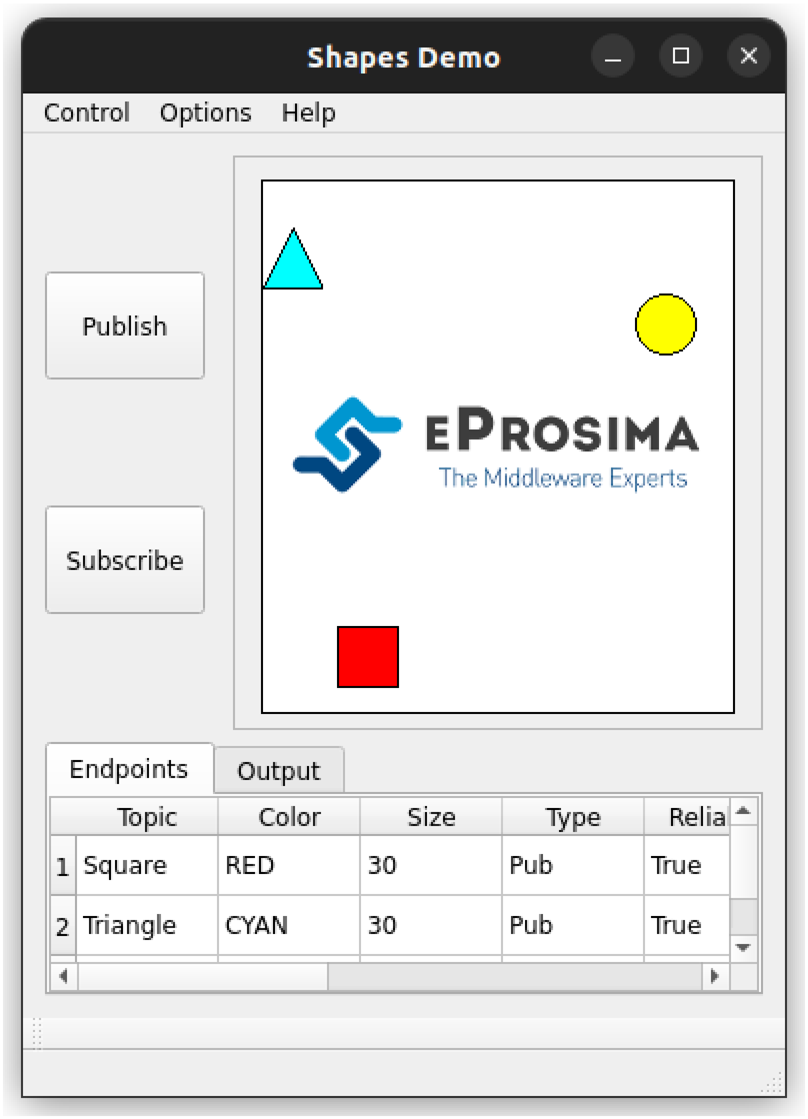
3.16.4 Running the application

Start ShapesDemo

Launch *eProsima Shapes Demo* application running the following command:

```
ShapesDemo
```

Start publishing in topics `Square`, `Triangle`, and `Circle` with default settings:



	Topic	Color	Size	Type	Relia
1	Square	RED	30	Pub	True
2	Triangle	CYAN	30	Pub	True

Recorder execution

Launch the *DDS Recorder* tool passing the configuration file as an argument:

```
ddsrecorder -c <path/to/config/file>/conf.yaml
```

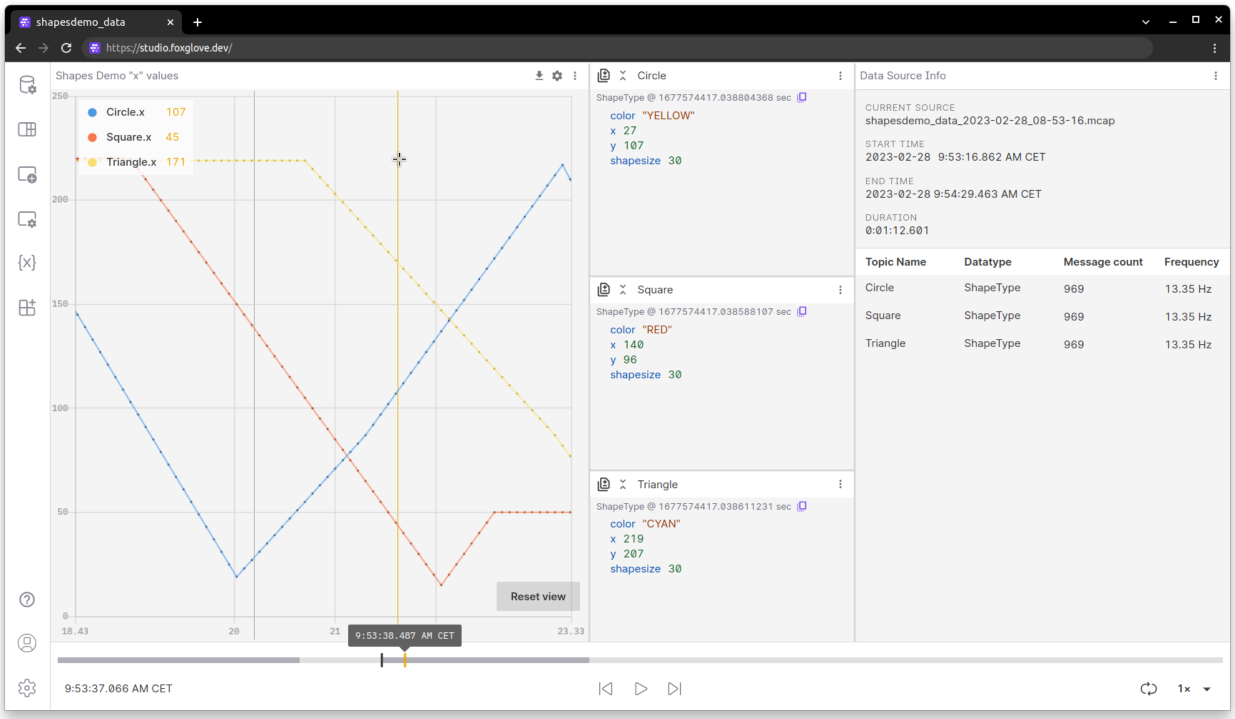
Once you have all the desired data, close the *DDS Recorder* application with `Ctrl+C`.

Important: Please remember to close the *DDS Recorder* application before accessing the output file as the *.mcap* file needs to be properly closed.

Visualize data with Foxglove Studio

Finally, we will show how to load the generated MCAP file into Foxglove Studio in order to display the saved data.

1. Open Foxglove Studio web application using Google Chrome or download the desktop application from their Foxglove website. We recommend to use the web application as the it is usually up to date with the latest features.
2. Click `Open local file` and load the *.mcap* file previously created: `shapesdemo_data.mcap`.
3. Once the *.mcap* file is loaded, create your own layout with custom panels to visualize the recorded data. The image below shows an example of a dashboard with several panels for data introspection.



Circle
ShapeType @ 1677574417.038804368 sec
```
color "YELLOW"
x 27
y 107
shapesize 30
```

Square
ShapeType @ 1677574417.038588107 sec
```
color "RED"
x 140
y 96
shapesize 30
```

Triangle
ShapeType @ 1677574417.038611231 sec
```
color "CYAN"
x 219
y 207
shapesize 30
```

Data Source Info

CURRENT SOURCE
shapesdemo_data_2023-02-28_08-53-16.mcap

START TIME
2023-02-28 9:53:16.862 AM CET

END TIME
2023-02-28 9:54:29.463 AM CET

DURATION
0:01:12.601

Topic Name	Datatype	Message count	Frequency
Circle	ShapeType	969	13.35 Hz
Square	ShapeType	969	13.35 Hz
Triangle	ShapeType	969	13.35 Hz

Feel free to further explore the number of possibilities that *eProsima DDS Recorder* and *Foxglove Studio* together have to offer.

3.17 Linux installation from sources

The instructions for installing the *eProsima DDS Record & Replay* from sources and its required dependencies are provided in this page. It is organized as follows:

3.17.1 Dependencies installation

DDS Record & Replay depends on *eProsima Fast DDS* library and certain Debian packages. This section describes the instructions for installing *DDS Record & Replay* dependencies and requirements in a Linux environment from sources. The following packages will be installed:

- `foonathan_memory_vendor`, an STL compatible C++ memory allocation library.
- `fastcdr`, a C++ library that serializes according to the standard CDR serialization mechanism.
- `fastrtps`, the core library of eProsima Fast DDS library.
- `cmake_utils`, an eProsima utils library for CMake.
- `cpp_utils`, an eProsima utils library for C++.
- `ddspipe`, an eProsima internal library that enables the communication of DDS interfaces.

First of all, the *Requirements* and *Dependencies* detailed below need to be met. Afterwards, the user can choose whether to follow either the *colcon* or the *CMake* installation instructions.

Requirements

The installation of *eProsima DDS Record & Replay* in a Linux environment from sources requires the following tools to be installed in the system:

- *CMake, g++, pip, wget and git*
- *Colcon* [optional]
- *Fast DDS Python* [for remote controller only]
- *Gtest* [for test only]

CMake, g++, pip, wget and git

These packages provide the tools required to install *eProsima DDS Record & Replay* and its dependencies from command line. Install CMake, g++, pip, wget and git using the package manager of the appropriate Linux distribution. For example, on Ubuntu use the command:

```
sudo apt install cmake g++ pip wget git
```

Colcon

colcon is a command line tool based on CMake aimed at building sets of software packages. Install the ROS 2 development tools (colcon and vcstool) by executing the following command:

```
pip3 install -U colcon-common-extensions vcstool
```

Note: If this fails due to an Environment Error, add the `--user` flag to the `pip3` installation command.

Fast DDS Python

eProsima Fast DDS Python is a Python binding for the eProsima Fast DDS C++ library. It is only required for the *remote controller application*.

Clone the Github repository into the *eProsima DDS Record & Replay* workspace and compile it with colcon as a dependency package. Use the following command to download the code:

```
git clone https://github.com/eProsima/Fast-DDS-python.git src/Fast-DDS-python
```

Gtest

Gtest is a unit testing library for C++. By default, *eProsima DDS Record & Replay* does not compile tests. It is possible to activate them with the opportune CMake options when calling colcon or CMake. For more details, please refer to the *CMake options* section. For a detailed description of the Gtest installation process, please refer to the Gtest Installation Guide.

It is also possible to clone the Gtest Github repository into the *eProsima DDS Record & Replay* workspace and compile it with colcon as a dependency package. Use the following command to download the code:

```
git clone --branch release-1.11.0 https://github.com/google/googletest src/googletest-
↪distribution
```

Dependencies

eProsima DDS Record & Replay has the following dependencies, when installed from sources in a Linux environment:

- *Asio and TinyXML2 libraries*
- *OpenSSL*
- *yaml-cpp*
- *SWIG* [for remote controller only]
- *PyQt6* [for remote controller only]
- *MCAP dependencies*
- *eProsima dependencies*

Asio and TinyXML2 libraries

Asio is a cross-platform C++ library for network and low-level I/O programming, which provides a consistent asynchronous model. TinyXML2 is a simple, small and efficient C++ XML parser. Install these libraries using the package manager of the appropriate Linux distribution. For example, on Ubuntu use the command:

```
sudo apt install libasio-dev libtinyxml2-dev
```

OpenSSL

OpenSSL is a robust toolkit for the TLS and SSL protocols and a general-purpose cryptography library. Install OpenSSL using the package manager of the appropriate Linux distribution. For example, on Ubuntu use the command:

```
sudo apt install libssl-dev
```

yaml-cpp

yaml-cpp is a YAML parser and emitter in C++ matching the YAML 1.2 spec, and is used by *DDS Record & Replay* application to parse the provided configuration files. Install yaml-cpp using the package manager of the appropriate Linux distribution. For example, on Ubuntu use the command:

```
sudo apt install libyaml-cpp-dev
```

SWIG

SWIG is a software development tool that connects programs written in C and C++ with a variety of high-level programming languages. It is leveraged by *Fast DDS Python* to generate a Python wrapper over Fast DDS library. SWIG is only a requirement for the *remote controller application*. It can be installed using the package manager of the appropriate Linux distribution. For example, on Ubuntu use the command:

```
sudo apt install swig libpython3-dev
```

PyQt6

The *eProsima DDS Record & Replay* remote controller is a graphical user interface application implemented in Python using PyQt6. To install PyQt6 simply run:

```
pip3 install PyQt6
```

Note: To install PyQt6 on Ubuntu 20.04, update `pip` and `setuptools` packages first.

```
python3 -m pip install pip setuptools --upgrade
```

MCAP dependencies

MCAP is a modular container format and logging library for pub/sub messages with arbitrary message serialization. It is primarily intended for use in robotics applications, and works well under various workloads, resource constraints, and durability requirements. MCAP C++ library is packed within *DDS Record & Replay* as a header-only, but its dependencies need to be installed using the package manager of the appropriate Linux distribution. For example, on Ubuntu use the command:

```
sudo apt install liblz4-dev libzstd-dev
```

Note: To work with MCAP files via command line interface, you can use *MCAP CLI <https://mcap.dev/guides/cli>* _ to manage the data in MCAP files.

eProsima dependencies

If it already exists in the system an installation of *Fast DDS* and *DDS Pipe* libraries, just source this libraries when building *eProsima DDS Record & Replay* by running the following commands. In other case, just skip this step.

```
source <fastdds-installation-path>/install/setup.bash
source <ddspipe-installation-path>/install/setup.bash
```

3.17.2 Colcon installation (recommended)

1. Create a `DDS-Record-Replay` directory and download the `.repos` file that will be used to install *eProsima DDS Record & Replay* and its dependencies:

   ```
   mkdir -p ~/DDS-Record-Replay/src
   cd ~/DDS-Record-Replay
   wget https://raw.githubusercontent.com/eProsima/DDS-Record-Replay/v0.4.0/
   ↪ddsrecordreplay.repos
   vcs import src < ddsrecordreplay.repos
   ```

   Note: In case there is already a *Fast DDS* installation in the system it is not required to download and build every dependency in the `.repos` file. It is just needed to download and build the *eProsima DDS Record &*

Replay project having sourced its dependencies. Refer to section *eProsima dependencies* in order to check how to source *Fast DDS* library.

2. Build the packages:

```
colcon build
```

Note: To install *DDS Recorder remote controller application*, compilation flag `-DBUILD_DDSRECORDER_CONTROLLER=ON` is required.

Note: Being based on CMake, it is possible to pass the CMake configuration options to the `colcon build` command. For more information on the specific syntax, please refer to the CMake specific arguments page of the colcon manual.

3.17.3 CMake installation

This section explains how to compile *eProsima DDS Record & Replay* with CMake, either *locally* or *globally*.

Local installation

1. Create a `DDS-Record-Replay` directory where to download and build *DDS Record & Replay* and its dependencies:

```
mkdir -p ~/DDS-Record-Replay/src
mkdir -p ~/DDS-Record-Replay/build
cd ~/DDS-Record-Replay
wget https://raw.githubusercontent.com/eProsima/DDS-Record-Replay/v0.4.0/
↪ddsrecordreplay.repos
vcs import src < ddsrecordreplay.repos
```

2. Compile all dependencies using CMake.

   - Foonathan memory

```
cd ~/DDS-Record-Replay
mkdir build/foonathan_memory_vendor
cd build/foonathan_memory_vendor
cmake ~/DDS-Record-Replay/src/foonathan_memory_vendor -DCMAKE_INSTALL_
↪PREFIX=~/DDS-Record-Replay/install -DBUILD_SHARED_LIBS=ON
cmake --build . --target install
```

   - Fast CDR

```
cd ~/DDS-Record-Replay
mkdir build/fastcdr
cd build/fastcdr
cmake ~/DDS-Record-Replay/src/fastcdr -DCMAKE_INSTALL_PREFIX=~/DDS-
↪Record-Replay/install
cmake --build . --target install
```

   - Fast DDS

```
cd ~/DDS-Record-Replay
mkdir build/fastdds
cd build/fastdds
cmake ~/DDS-Record-Replay/src/fastdds -DCMAKE_INSTALL_PREFIX=~/DDS-
↪Record-Replay/install -DCMAKE_PREFIX_PATH=~/DDS-Record-Replay/install
cmake --build . --target install
```

- Dev Utils

```
# CMake Utils
cd ~/DDS-Record-Replay
mkdir build/cmake_utils
cd build/cmake_utils
cmake ~/DDS-Record-Replay/src/dev-utils/cmake_utils -DCMAKE_INSTALL_
↪PREFIX=~/DDS-Record-Replay/install -DCMAKE_PREFIX_PATH=~/DDS-Record-
↪Replay/install
cmake --build . --target install

# C++ Utils
cd ~/DDS-Record-Replay
mkdir build/cpp_utils
cd build/cpp_utils
cmake ~/DDS-Record-Replay/src/dev-utils/cpp_utils -DCMAKE_INSTALL_
↪PREFIX=~/DDS-Record-Replay/install -DCMAKE_PREFIX_PATH=~/DDS-Record-
↪Replay/install
cmake --build . --target install
```

- DDS Pipe

```
# ddspipe_core
cd ~/DDS-Record-Replay
mkdir build/ddspipe_core
cd build/ddspipe_core
cmake ~/DDS-Record-Replay/src/ddspipe/ddspipe_core -DCMAKE_INSTALL_
↪PREFIX=~/DDS-Record-Replay/install -DCMAKE_PREFIX_PATH=~/DDS-Record-
↪Replay/install
cmake --build . --target install

# ddspipe_participants
cd ~/DDS-Record-Replay
mkdir build/ddspipe_participants
cd build/ddspipe_participants
cmake ~/DDS-Record-Replay/src/ddspipe/ddspipe_participants -DCMAKE_
↪INSTALL_PREFIX=~/DDS-Record-Replay/install -DCMAKE_PREFIX_PATH=~/DDS-
↪Record-Replay/install
cmake --build . --target install

# ddspipe_yaml
cd ~/DDS-Record-Replay
mkdir build/ddspipe_yaml
cd build/ddspipe_yaml
cmake ~/DDS-Record-Replay/src/ddspipe/ddspipe_yaml -DCMAKE_INSTALL_
↪PREFIX=~/DDS-Record-Replay/install -DCMAKE_PREFIX_PATH=~/DDS-Record-
↪Replay/install
```

(continues on next page)

(continued from previous page)

```
cmake --build . --target install
```

3. Once all dependencies are installed, install *eProsima DDS Record & Replay*:

```
# ddsrecorder_participants
cd ~/DDS-Record-Replay
mkdir build/ddsrecorder_participants
cd build/ddsrecorder_participants
cmake ~/DDS-Record-Replay/src/ddsrecorder/ddsrecorder_participants -DCMAKE_INSTALL_
↪PREFIX=~/DDS-Record-Replay/install -DCMAKE_PREFIX_PATH=~/DDS-Record-Replay/install
cmake --build . --target install

# ddsrecorder_yaml
cd ~/DDS-Record-Replay
mkdir build/ddsrecorder_yaml
cd build/ddsrecorder_yaml
cmake ~/DDS-Record-Replay/src/ddsrecorder/ddsrecorder_yaml -DCMAKE_INSTALL_PREFIX=~/
↪DDS-Record-Replay/install -DCMAKE_PREFIX_PATH=~/DDS-Record-Replay/install
cmake --build . --target install

# ddsrecorder
cd ~/DDS-Record-Replay
mkdir build/ddsrecorder_tool
cd build/ddsrecorder_tool
cmake ~/DDS-Record-Replay/src/ddsrecorder/ddsrecorder -DCMAKE_INSTALL_PREFIX=~/DDS-
↪Record-Replay/install -DCMAKE_PREFIX_PATH=~/DDS-Record-Replay/install
cmake --build . --target install

# ddsreplayer
cd ~/DDS-Record-Replay
mkdir build/ddsreplayer_tool
cd build/ddsreplayer_tool
cmake ~/DDS-Record-Replay/src/ddsrecorder/ddsreplayer -DCMAKE_INSTALL_PREFIX=~/DDS-
↪Record-Replay/install -DCMAKE_PREFIX_PATH=~/DDS-Record-Replay/install
cmake --build . --target install
```

Note: By default, *eProsima DDS Record & Replay* does not compile tests. However, they can be activated by downloading and installing Gtest and building with CMake option `-DBUILD_TESTS=ON`.

4. Optionally, install the *remote controller application* along with its dependency *Fast DDS Python*:

```
# Fast DDS Python
cd ~/DDS-Record-Replay
mkdir build/fastdds_python
cd build/fastdds_python
cmake ~/DDS-Record-Replay/src/Fast-DDS-python/fastdds_python -DCMAKE_INSTALL_
↪PREFIX=~/DDS-Record-Replay/install -DCMAKE_PREFIX_PATH=~/DDS-Record-Replay/install
cmake --build . --target install

# Remote Controller Application
cd ~/DDS-Record-Replay
```

(continues on next page)

(continued from previous page)

```
mkdir build/controller_tool
cd build/controller_tool
cmake ~/DDS-Record-Replay/src/ddsrecorder/controller/controller_tool -DCMAKE_
↪INSTALL_PREFIX=~/DDS-Record-Replay/install -DCMAKE_PREFIX_PATH=~/DDS-Record-
↪Replay/install -DBUILD_DDSRECORDER_CONTROLLER=ON
cmake --build . --target install
```

Global installation

To install *eProsima DDS Record & Replay* system-wide instead of locally, remove all the flags that appear in the configuration steps of `Fast-CDR`, `Fast-DDS`, `Dev-Utils`, `DDS-Pipe`, and `DDS-Record-Replay`, and change the first in the configuration step of `foonathan_memory_vendor` to the following:

```
-DCMAKE_INSTALL_PREFIX=/usr/local/ -DBUILD_SHARED_LIBS=ON
```

3.17.4 Run an application

To run the *DDS Recorder* tool, source the installation path and execute the executable file that has been installed in `<install-path>/ddsrecorder_tool/bin/ddsrecorder`:

```
# If built has been done using colcon, all projects could be sourced as follows
source install/setup.bash
./<install-path>/ddsrecorder_tool/bin/ddsrecorder
```

Likewise, to run the *DDS Replay tool*, source the installation path and execute the executable file that has been installed in `<install-path>/ddsreplayer_tool/bin/ddsreplayer`:

```
# If built has been done using colcon, all projects could be sourced as follows
source install/setup.bash
./<install-path>/ddsreplayer_tool/bin/ddsreplayer
```

Be sure that these executables have execution permissions.

3.18 Windows installation from sources

The instructions for installing the *eProsima DDS Record & Replay* application from sources and its required dependencies are provided in this page. It is organized as follows:

3.18.1 Dependencies installation

eProsima DDS Record & Replay depends on *eProsima Fast DDS* library and certain Debian packages. This section describes the instructions for installing *eProsima DDS Record & Replay* dependencies and requirements in a Windows environment from sources. The following packages will be installed:

- `foonathan_memory_vendor`, an STL compatible C++ memory allocation library.
- `fastcdr`, a C++ library that serializes according to the standard CDR serialization mechanism.
- `fastrtps`, the core library of eProsima Fast DDS library.
- `cmake_utils`, an eProsima utils library for CMake.
- `cpp_utils`, an eProsima utils library for C++.
- `ddspipe`, an eProsima internal library that enables the communication of DDS interfaces.

First of all, the *Requirements* and *Dependencies* detailed below need to be met. Afterwards, the user can choose whether to follow either the *colcon* or the *CMake* installation instructions.

Requirements

The installation of *eProsima Fast DDS* in a Windows environment from sources requires the following tools to be installed in the system:

- *Visual Studio*
- *Chocolatey*
- *CMake, pip3, wget and git*
- *Colcon* [optional]
- *Fast DDS Python* [for remote controller only]
- *Gtest* [for test only]

Visual Studio

Visual Studio is required to have a C++ compiler in the system. For this purpose, make sure to check the `Desktop development with C++` option during the Visual Studio installation process.

If Visual Studio is already installed but the Visual C++ Redistributable packages are not, open Visual Studio and go to `Tools` -> `Get Tools and Features` and in the `Workloads` tab enable `Desktop development with C++`. Finally, click `Modify` at the bottom right.

Chocolatey

Chocolatey is a Windows package manager. It is needed to install some of *eProsima Fast DDS*'s dependencies. Download and install it directly from the website.

CMake, pip3, wget and git

These packages provide the tools required to install *eProsima Fast DDS* and its dependencies from command line. Download and install CMake, pip3, wget and git by following the instructions detailed in the respective websites. Once installed, add the path to the executables to the PATH from the *Edit the system environment variables* control panel.

Colcon

colcon is a command line tool based on CMake aimed at building sets of software packages. Install the ROS 2 development tools (colcon and vcstool) by executing the following command:

```
pip3 install -U colcon-common-extensions vcstool
```

Note: If this fails due to an Environment Error, add the `--user` flag to the `pip3` installation command.

Fast DDS Python

eProsima Fast DDS Python is a Python binding for the eProsima Fast DDS C++ library. It is only required for the *remote controller application*.

Clone the Github repository into the *eProsima DDS Record & Replay* workspace and compile it with colcon as a dependency package. Use the following command to download the code:

```
git clone https://github.com/eProsima/Fast-DDS-python.git src/Fast-DDS-python
```

Gtest

Gtest is a unit testing library for C++. By default, *eProsima DDS Record & Replay* does not compile tests. It is possible to activate them with the opportune CMake options when calling colcon or CMake. For more details, please refer to the *CMake options* section.

Run the following commands on your workspace to install Gtest.

```
git clone https://github.com/google/googletest.git
cmake -DCMAKE_INSTALL_PREFIX='C:\Program Files\gtest' -Dgtest_force_shared_crt=ON -
↪DBUILD_GMOCK=ON ^
    -B build\gtest -A x64 -T host=x64 googletest
cmake --build build\gtest --config Release --target install
```

or refer to the Gtest Installation Guide for a detailed description of the Gtest installation process.

Dependencies

eProsima DDS Record & Replay has the following dependencies, when installed from sources in a Windows environment:

- *Asio and TinyXML2 libraries*
- *OpenSSL*
- *yaml-cpp*
- *SWIG* [for remote controller only]
- *PyQt6* [for remote controller only]
- *MCAP dependencies*
- *eProsima dependencies*

Asio and TinyXML2 libraries

Asio is a cross-platform C++ library for network and low-level I/O programming, which provides a consistent asynchronous model. TinyXML2 is a simple, small and efficient C++ XML parser. They can be downloaded directly from the links below:

- Asio
- TinyXML2

After downloading these packages, open an administrative shell with *PowerShell* and execute the following command:

```
choco install -y -s <PATH_TO_DOWNLOADS> asio tinyxml2
```

where `<PATH_TO_DOWNLOADS>` is the folder into which the packages have been downloaded.

OpenSSL

OpenSSL is a robust toolkit for the TLS and SSL protocols and a general-purpose cryptography library. Download and install the latest OpenSSL version for Windows at this link. After installing, add the environment variable `OPENSSL_ROOT_DIR` pointing to the installation root directory.

For example:

```
OPENSSL_ROOT_DIR=C:\Program Files\OpenSSL-Win64
```

yaml-cpp

`yaml-cpp` is a YAML parser and emitter in C++ matching the YAML 1.2 spec, and is used by *DDS Record & Replay* application to parse the provided configuration files. From an administrative shell with *PowerShell*, execute the following commands in order to download and install `yaml-cpp` for Windows:

```
git clone --branch yaml-cpp-0.7.0 https://github.com/jbeder/yaml-cpp
cmake -DCMAKE_INSTALL_PREFIX='C:\Program Files\yamlcpp' -B build\yamlcpp yaml-cpp
cmake --build build\yamlcpp --target install    # If building in Debug mode, add --
↪config Debug
```

MCAP dependencies

MCAP is a modular container format and logging library for pub/sub messages with arbitrary message serialization. It is primarily intended for use in robotics applications, and works well under various workloads, resource constraints, and durability requirements. MCAP C++ library is packed within *DDS Record & Replay* as a header-only, but its dependencies need to be installed using the appropriate Windows package manager.

It is recommended to use vcpkg dependency manager to install LZ4 and zstd dependencies. Once both dependencies are installed, add the directory where the binaries are located to the `PATH`. The installed binaries are usually located under `<path\to\vcpkg>/\installed\x64-windows\bin` directory.

Note: To work with MCAP files via command line interface, you can use *MCAP CLI <https://mcap.dev/guides/cli>* _ to manage the data in MCAP files.

SWIG

SWIG is a software development tool that connects programs written in C and C++ with a variety of high-level programming languages. It is leveraged by *Fast DDS Python* to generate a Python wrapper over Fast DDS library. SWIG is only a requirement for the *remote controller application*. Download and install SWIG for Windows, choosing one of the releases available at their website.

PyQt6

The *eProsima DDS Record & Replay* remote controller is a graphical user interface application implemented in Python using PyQt6. To install PyQt6 simply run:

```
pip3 install PyQt6
```

eProsima dependencies

If it already exists in the system an installation of *Fast DDS* and *DDS Pipe* libraries, just source this libraries when building the *eProsima DDS Record & Replay* application by using the command:

```
source <fastdds-installation-path>/install/setup.bash
source <ddspipe-installation-path>/install/setup.bash
```

In other case, just skip this step.

3.18.2 Colcon installation (recommended)

Important: Run colcon within a Visual Studio prompt. To do so, launch a *Developer Command Prompt* from the search engine.

1. Create a `DDS-Record-Replay` directory and download the `.repos` file that will be used to install *eProsima DDS Record & Replay* and its dependencies:

```
mkdir <path\to\user\workspace>\DDS-Record-Replay
cd <path\to\user\workspace>\DDS-Record-Replay
mkdir src
wget https://raw.githubusercontent.com/eProsima/DDS-Record-Replay/v0.4.0/
↪ddsrecordreplay.repos ddsrecordreplay.repos
vcs import src < ddsrecordreplay.repos
```

Note: In case there is already a *Fast DDS* installation in the system it is not required to download and build every dependency in the `.repos` file. It is just needed to download and build the *eProsima DDS Record & Replay* project having sourced its dependencies. Refer to section *eProsima dependencies* in order to check how to source *Fast DDS* library.

2. Build the packages:

```
colcon build
```

Note: To install *DDS Recorder remote controller application*, compilation flag `-DBUILD_DDSRECORDER_CONTROLLER=ON` is required.

Note: Being based on CMake, it is possible to pass the CMake configuration options to the `colcon build` command. For more information on the specific syntax, please refer to the CMake specific arguments page of the colcon manual.

3.18.3 CMake installation

This section explains how to compile *eProsima DDS Record & Replay* with CMake, either *locally* or *globally*.

Local installation

1. Open a command prompt, and create a `DDS-Record-Replay` directory where to download and build *eProsima DDS Record & Replay* and its dependencies:

```
mkdir <path\to\user\workspace>\DDS-Record-Replay
mkdir <path\to\user\workspace>\DDS-Record-Replay\src
mkdir <path\to\user\workspace>\DDS-Record-Replay\build
cd <path\to\user\workspace>\DDS-Record-Replay
wget https://raw.githubusercontent.com/eProsima/DDS-Record-Replay/v0.4.0/
↪ddsrecordreplay.repos ddsrecordreplay.repos
vcs import src < ddsrecordreplay.repos
```

2. Compile all dependencies using CMake.

   - Foonathan memory

```
cd <path\to\user\workspace>\DDS-Record-Replay
mkdir build\foonathan_memory_vendor
cd build\foonathan_memory_vendor
cmake <path\to\user\workspace>\DDS-Record-Replay\src\foonathan_memory_
↪vendor -DCMAKE_INSTALL_PREFIX=<path\to\user\workspace>\DDS-Record-
↪Replay\install ^
```

(continues on next page)

(continued from previous page)

```
    -DBUILD_SHARED_LIBS=ON
cmake --build . --config Release --target install
```

- Fast CDR

```
cd <path\to\user\workspace>\DDS-Record-Replay
mkdir build\fastcdr
cd build\fastcdr
cmake <path\to\user\workspace>\DDS-Record-Replay\src\fastcdr -DCMAKE_
↪INSTALL_PREFIX=<path\to\user\workspace>\DDS-Record-Replay\install
cmake --build . --config Release --target install
```

- Fast DDS

```
cd <path\to\user\workspace>\DDS-Record-Replay
mkdir build\fastdds
cd build\fastdds
cmake <path\to\user\workspace>\DDS-Record-Replay\src\fastdds -DCMAKE_
↪INSTALL_PREFIX=<path\to\user\workspace>\DDS-Record-Replay\install ^
    -DCMAKE_PREFIX_PATH=<path\to\user\workspace>\DDS-Record-Replay\
↪install
cmake --build . --config Release --target install
```

- Dev Utils

```
# CMake Utils
cd <path\to\user\workspace>\DDS-Record-Replay
mkdir build\cmake_utils
cd build\cmake_utils
cmake <path\to\user\workspace>\DDS-Record-Replay\src\dev-utils\cmake_
↪utils -DCMAKE_INSTALL_PREFIX=<path\to\user\workspace>\DDS-Record-
↪Replay\install ^
    -DCMAKE_PREFIX_PATH=<path\to\user\workspace>\DDS-Record-Replay\
↪install
cmake --build . --config Release --target install

# C++ Utils
cd <path\to\user\workspace>\DDS-Record-Replay
mkdir build\cpp_utils
cd build\cpp_utils
cmake <path\to\user\workspace>\DDS-Record-Replay\src\dev-utils\cpp_utils␣
↪-DCMAKE_INSTALL_PREFIX=<path\to\user\workspace>\DDS-Record-Replay\
↪install ^
    -DCMAKE_PREFIX_PATH=<path\to\user\workspace>\DDS-Record-Replay\
↪install
cmake --build . --config Release --target install
```

- DDS Pipe

```
# ddspipe_core
cd <path\to\user\workspace>\DDS-Record-Replay
mkdir build\ddspipe_core
cd build\ddspipe_core
```

(continues on next page)

(continued from previous page)

```
cmake cd <path\to\user\workspace>\DDS-Record-Replay\src\ddspipe\ddspipe_
↪core -DCMAKE_INSTALL_PREFIX=<path\to\user\workspace>\DDS-Record-Replay\
↪install -DCMAKE_PREFIX_PATH=<path\to\user\workspace>\DDS-Record-Replay\
↪install
cmake --build . --target install

# ddspipe_yaml
cd <path\to\user\workspace>\DDS-Record-Replay
mkdir build\ddspipe_yaml
cd build\ddspipe_yaml
cmake <path\to\user\workspace>\DDS-Record-Replay\src\ddspipe\ddspipe_
↪yaml -DCMAKE_INSTALL_PREFIX=<path\to\user\workspace>\DDS-Record-Replay\
↪install -DCMAKE_PREFIX_PATH=<path\to\user\workspace>\DDS-Record-Replay\
↪install
cmake --build . --target install

# ddspipe_participants
cd <path\to\user\workspace>\DDS-Record-Replay
mkdir build\ddspipe_participants
cd build\ddspipe_participants
cmake <path\to\user\workspace>\DDS-Record-Replay\src\ddspipe\ddspipe_
↪participants -DCMAKE_INSTALL_PREFIX=<path\to\user\workspace>\DDS-
↪Record-Replay\install -DCMAKE_PREFIX_PATH=<path\to\user\workspace>\DDS-
↪Record-Replay\install
cmake --build . --target install
```

3. Once all dependencies are installed, install *eProsima DDS Record & Replay*:

```
# ddsrecorder_participants
cd <path\to\user\workspace>\DDS-Record-Replay
mkdir build\ddsrecorder_participants
cd build\ddsrecorder_participants
cmake <path\to\user\workspace>\DDS-Record-Replay\src\ddsrecorder\ddsrecorder_
↪participants ^
    -DCMAKE_INSTALL_PREFIX=<path\to\user\workspace>\DDS-Record-Replay\install -
↪DCMAKE_PREFIX_PATH=<path\to\user\workspace>\DDS-Record-Replay\install
cmake --build . --config Release --target install

# ddsrecorder_yaml
cd <path\to\user\workspace>\DDS-Record-Replay
mkdir build\ddsrecorder_yaml
cd build\ddsrecorder_yaml
cmake <path\to\user\workspace>\DDS-Record-Replay\src\ddsrecorder\ddsrecorder_yaml -
↪DCMAKE_INSTALL_PREFIX=<path\to\user\workspace>\DDS-Record-Replay\install ^
    -DCMAKE_PREFIX_PATH=<path\to\user\workspace>\DDS-Record-Replay\install
cmake --build . --config Release --target install

# ddsrecorder
cd <path\to\user\workspace>\DDS-Record-Replay
mkdir build\ddsrecorder_tool
cd build\ddsrecorder_tool
cmake <path\to\user\workspace>\DDS-Record-Replay\src\ddsrecorder\ddsrecorder -
↪DCMAKE_INSTALL_PREFIX=<path\to\user\workspace>\DDS-Record-Replay\install ^
```

(continues on next page)

(continued from previous page)

```
    -DCMAKE_PREFIX_PATH=<path\to\user\workspace>\DDS-Record-Replay\install
cmake --build . --config Release --target install

# ddsreplayer
cd <path\to\user\workspace>\DDS-Record-Replay
mkdir build\ddsreplayer_tool
cd build\ddsreplayer_tool
cmake <path\to\user\workspace>\DDS-Record-Replay\src\ddsrecorder\ddsreplayer -
↪DCMAKE_INSTALL_PREFIX=<path\to\user\workspace>\DDS-Record-Replay\install ^
    -DCMAKE_PREFIX_PATH=<path\to\user\workspace>\DDS-Record-Replay\install
cmake --build . --config Release --target install
```

Note: By default, *eProsima DDS Record & Replay* does not compile tests. However, they can be activated by downloading and installing Gtest and building with CMake option -DBUILD_TESTS=ON.

4. Optionally, install the *remote controller application* along with its dependency *Fast DDS Python*:

```
# Fast DDS Python
cd <path\to\user\workspace>\DDS-Record-Replay
mkdir build\fastdds_python
cd build\fastdds_python
cmake <path\to\user\workspace>\DDS-Record-Replay\src\Fast-DDS-python\fastdds_python␣
↪-DCMAKE_INSTALL_PREFIX=<path\to\user\workspace>\DDS-Record-Replay\install ^
    -DCMAKE_PREFIX_PATH=<path\to\user\workspace>\DDS-Record-Replay\install
cmake --build . --config Release --target install

# Remote Controller Application
cd <path\to\user\workspace>\DDS-Record-Replay
mkdir build\controller_tool
cd build\controller_tool
cmake <path\to\user\workspace>\DDS-Record-Replay\src\controller\controller_tool -
↪DCMAKE_INSTALL_PREFIX=<path\to\user\workspace>\DDS-Record-Replay\install ^
    -DCMAKE_PREFIX_PATH=<path\to\user\workspace>\DDS-Record-Replay\install -DBUILD_
↪DDSRECORDER_CONTROLLER=ON
cmake --build . --config Release --target install
```

Global installation

To install *eProsima DDS Record & Replay* system-wide instead of locally, remove all the flags that appear in the configuration steps of Fast-CDR, Fast-DDS, Dev-Utils, DDS-Pipe, and DDS-Record-Replay

3.18.4 Run an application

If *eProsima DDS Record & Replay* was compiled using colcon, when running an instance of *DDS Recorder* or *DDS Replayer*, the colcon overlay built in the dedicated `DDS-Record-Replay` directory must be sourced. There are two possibilities:

- Every time a new shell is opened, prepare the environment locally by typing the command:

  ```
  setup.bat
  ```

- Add the sourcing of the colcon overlay permanently, by opening the *Edit the system environment variables* control panel, and adding the installation path to the `PATH`.

However, when running a *DDS Recorder* or *DDS Replayer* compiled using CMake, it must be linked with its dependencies where the packages have been installed. This can be done by opening the *Edit system environment variables* control panel and adding to the `PATH` *eProsima DDS Record & Replay* installation directory: `<path\to\user\workspace>\DDS-Record-Replay\install`.

3.19 CMake options

eProsima DDS Record & Replay provides numerous CMake options for changing the behavior and configuration of *eProsima DDS Record & Replay*. These options allow the developer to enable/disable certain *eProsima DDS Record & Replay* settings by defining these options to `ON`/`OFF` at the CMake execution, or set the required path to certain dependencies.

Warning: These options are only for developers who installed *eProsima DDS Record & Replay* following the compilation steps described in *Linux installation from sources*.

Option	Description	Possible values	Default
`CMAKE_BUILD_TYPE`	CMake optimization build type.	`Release` `Debug`	`Release`
`BUILD_DDSRECORDER_CONTROLLER`	Build the *DDS Recorder* remote controller application.	`OFF` `ON`	`OFF`
`BUILD_DOCS`	Build the *eProsima DDS Record & Replay* documentation.	`OFF` `ON`	`OFF`
`BUILD_TESTS`	Build the *eProsima DDS Record & Replay* tools and documentation tests.	`OFF` `ON`	`OFF`
`LOG_INFO`	Activate *eProsima DDS Record & Replay* logs. It is set to `ON` if `CMAKE_BUILD_TYPE` is set to `Debug`.	`OFF` `ON`	`ON` if `Debug` `OFF` otherwise
`ASAN_BUILD`	Activate address sanitizer build.	`OFF` `ON`	`OFF`
`TSAN_BUILD`	Activate thread sanitizer build.	`OFF` `ON`	`OFF`

3.20 Version v0.4.0

This release includes the following **Recording features**:

- Publish the *Logs* in a DDS topic.
- New *Monitor* module.

This release includes the following **DDS Recorder tool configuration features**:

- New configuration option `logging` to configure the *Logs*.

This release includes the following **DDS Replayer tool configuration features**:

- New configuration option `logging` to configure the *Logs*.

This release includes the following **Dependencies Update**:

	Repository	Old Version	New Version
Foonathan Memory Vendor	eProsima/foonathan_memory_vendor	v1.3.1	v1.3.1
Fast CDR	eProsima/Fast-CDR	v2.1.2	v2.2.0
Fast DDS	eProsima/Fast-DDS	v2.13.1	v2.14.0
Dev Utils	eProsima/dev-utils	v0.5.0	v0.6.0
DDS Pipe	eProsima/DDS-Pipe	v0.3.0	v0.4.0

3.21 Previous Versions

3.21.1 Version v0.3.0

This release includes the following **Recording features**:

- New DDS Recorder suspended (active stopped) state (see *remote controller* for more details).

This release includes the following **DDS Recorder & Replay internal adjustments**:

- Store *DDS Record & Replay* version in metadata record of the generated MCAP files.
- Move dynamic types storage from metadata to attachments MCAP section.
- Set *app_id* and *app_metadata* attributes on *DDS Record & Replay* participants.
- Store schemas in OMG IDL and ROS 2 msg format.

Warning: Types recorded with previous versions of *DDS Record & Replay* are no longer "replayable" after this update.

This release includes the following **DDS Recorder tool configuration features**:

- Support *Compression Settings*.
- Allow disabling the storage of received types (see *Record Types*).
- New configuration options (`timestamp-format` and `local-timestamp`) available for *output file* settings.
- New configuration option (`topics`) to configure the *Manual Topics*.
- Rename `max-reception-rate` to `max-rx-rate`.
- Record data in either ROS 2 format or the raw DDS format (see *Topic Type Format*).

This release includes the following **DDS Replayer tool configuration features**:

- New configuration option (`topics`) to configure the *Manual Topics*.
- New configuration option (`max-tx-rate`) to configure the *Max transmission rate*.
- Remove the support for Built-in Topics.
- Read data in either ROS 2 format or the raw DDS format.

This release includes the following **Dependencies Update**:

	Repository	Old Version	New Version
Foonathan Memory Vendor	eProsima/foonathan_memory_vendor	v1.3.1	v1.3.1
Fast CDR	eProsima/Fast-CDR	v1.1.0	v2.1.3
Fast DDS	eProsima/Fast-DDS	v2.11.0	v2.13.1
Dev Utils	eProsima/dev-utils	v0.4.0	v0.5.0
DDS Pipe	eProsima/DDS-Pipe	v0.2.0	v0.3.0

3.21.2 Version v0.2.0

This release includes *DDS Replay tool*, supporting the following **Replay features**:

- Supports setting *begin* and *end* times (`begin-time` / `end-time`).
- Supports setting a replay *start* time (`start-replay-time`).
- Supports playing stored data at a specific playback *rate* (`rate`).
- Supports sending dynamic types stored in input MCAP file.

This release includes the following **User Interface features**:

- *Replay Service Command-Line Parameters*.

This release includes the following (*DDS Replay tool*) **Configuration features**:

- Support YAML *configuration file*.
- Support for allow and block topic filters at execution time and in run-time.
- Support configuration related to DDS communication.
- Support configuration of playback settings.
- Support configuration of the internal operation of the DDS Replayer.
- Support enabling/disabling dynamic types dispatch (see *Only With Type*).
- Support *Interface Whitelisting*.
- Support *Custom Transport Descriptors* (UDP or Shared Memory only).
- Support *Ignore Participant Flags*.

This release includes the following **Recording features**:

- Supports recording messages with unknown (dynamic) data type, and to only record data whose type is known (see *Only With Type*).

This release includes the following (*DDS Recorder tool*) **Configuration features**:

- Support record only data whose (dynamic) type is known: `only-with-type:  true` (see *Only With Type*).
- Support *Interface Whitelisting*.

- Support *Custom Transport Descriptors* (UDP or Shared Memory only).
- Support *Ignore Participant Flags*.

This release includes the following **Documentation features**:

- Updated documentation with Replay service configuration and usage instructions.

This release includes the following **Dependencies Update**:

	Repository	Old Version	New Version
Foonathan Memory Vendor	eProsima/foonathan_memory_vendor	v1.3.0	v1.3.1
Fast CDR	eProsima/Fast-CDR	v1.0.27	v1.1.0
Fast DDS	eProsima/Fast-DDS	v2.10.1	v2.11.0
Dev Utils	eProsima/dev-utils	v0.3.0	v0.4.0
DDS Pipe	eProsima/DDS-Pipe	v0.1.0	v0.2.0

3.21.3 Version v0.1.0

This is the first release of *eProsima DDS Record & Replay*.

This release includes several **features** regarding the recording of DDS data, configuration and user interaction.

This release includes the following **Recording features**:

- Supports DynamicTypes.
- Supports saves the data in a MCAP database.
- Supports for `downsampling` that reduces the sampling rate of the received data.
- Supports for `buffer-size` that indicates the number of samples to be stored in the process memory before the dump to disk.

This release includes the following **User Interface features**:

- *Recording Service Command-Line Parameters*.
- *Remote Control*.

This release includes the following **Configuration features**:

- Support YAML *configuration file*.
- Support for allow and block topic filters at execution time and in run-time.
- Support configuration related to DDS communication.
- Support configuration of data writing in the database.
- Support configuration of the remote controller of the DDS Recorder.
- Support configuration of the internal operation of the DDS Recorder.

This release includes the following **Tutorials**:

- *Configuring Fast DDS DynamicTypes for data recording*.
- *Visualize recorded data with Foxglove*.

This release includes the following **Documentation features**:

- This same documentation.

3.22 Glossary

3.22.1 Networking nomenclature

LAN **Local Area Network**

3.22.2 DDS Record & Replay nomenclature

MCAP Modular container file format for heterogeneous timestamped data.

3.22.3 DDS nomenclature

DataReader DDS element that subscribes to a specific Topic. It belong to one and only one Participant, and it is uniquely identified by a Guid.

See Fast DDS documentation for further information.

DataWriter DDS entity that publish data in a specific Topic. It belong to one and only one Participant, and it is uniquely identified by a Guid.

See Fast DDS documentation for further information.

Domain Id The Domain Id is a virtual partition for DDS networks. Only DomainParticipants with the same Domain Id would be able to communicate to each other. DomainParticipants in different Domains will not even discover each other.

See Fast DDS documentation for further information.

DomainParticipant A DomainParticipant is the entry point of the application to a DDS Domain. Every DomainParticipant is linked to a single domain from its creation, and cannot change such domain. It also acts as a factory for Publisher, Subscriber and Topic.

See Fast DDS documentation for further information.

Endpoint DDS element that publish or subscribes in a specific Topic. Endpoint kinds are *DataWriter* or *DataReader*.

Guid Global Unique Identifier. It contains a GuidPrefix and an EntityId. The EntityId uniquely identifies sub-entities inside a Participant. Identifies uniquely a DDS entity (DomainParticipant, DataWriter or DataReader).

GuidPrefix Global Unique Identifier shared by a Participant and all its sub-entities. Identifies uniquely a DDS Participant.

Topic DDS isolation abstraction to encapsulate subscriptions and publications. Each Topic is uniquely identified by a topic name and a topic type name (name of the data type it transmits).

See Fast DDS documentation for further information.

DynamicTypes The dynamic topic types offer the possibility to work over DDS without the restrictions related to the IDLs. Using them, the users can declare the different types that they need and manage the information directly, avoiding the additional step of updating the IDL file and the generation of C++ classes.

See Fast DDS documentation for further information.

INDEX