DDS Record Documentation

Release ..

eProsima

Apr 11, 2023

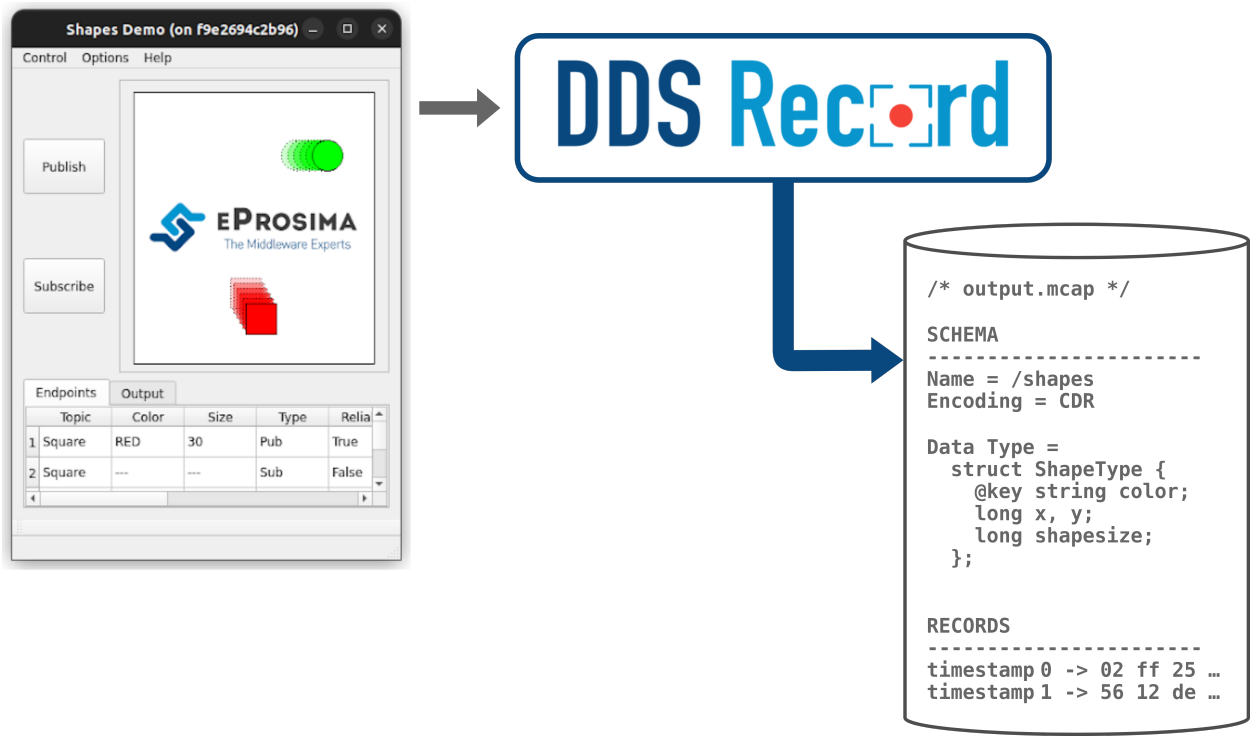
INTRODUCTION

eProsima DDS Record 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. At the moment, it is only possible to replay the data using external tools capable of interpreting the MCAP format, as the *eProsima DDS Record* does not provide a replay tool.

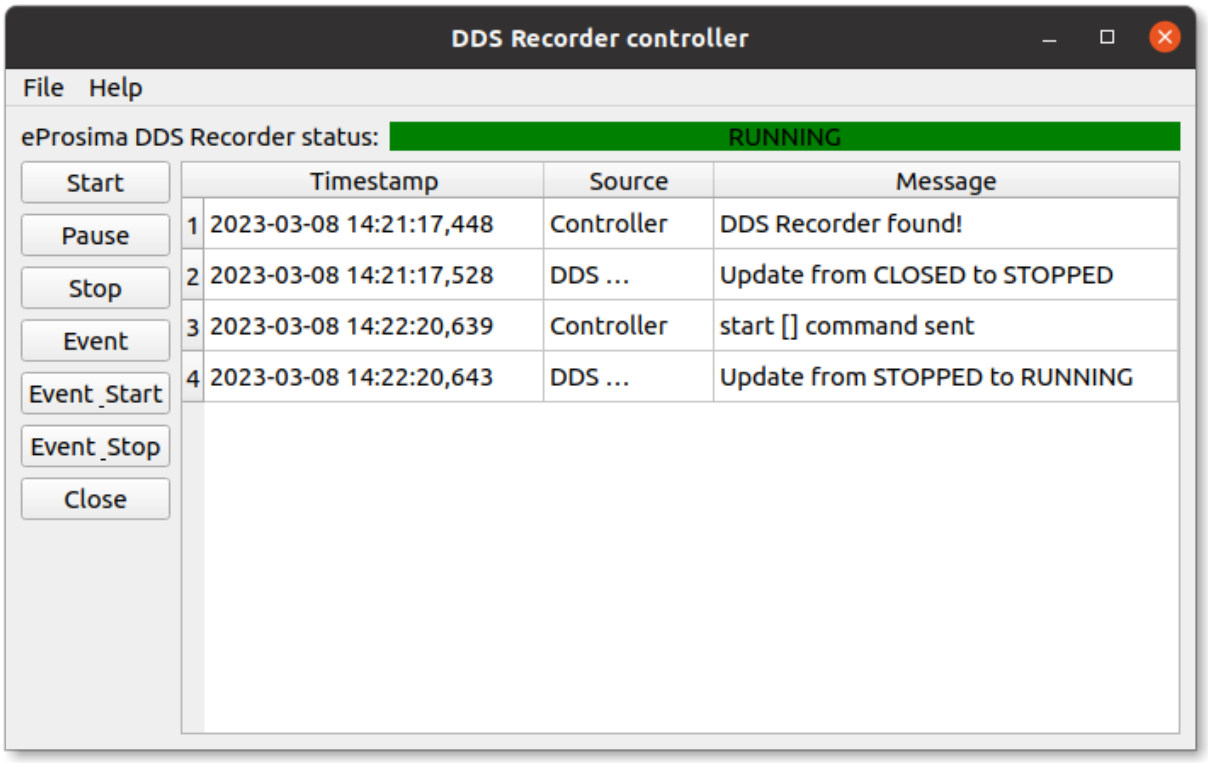
eProsima DDS Record is easily configurable and installed with a default setup, so that DDS topics, data types and entities are automatically discovered without the need to specify the types of data recorded. This is because the recording tool exploits the DynamicTypes functionality of eProsima Fast DDS,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

eProsima DDS Record 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 Recorder controller
File
Help
eProsima DDS Recorder status:
RUNNING
Start
Pause
Stop
Event
Event_Start
Event_Stop
Close
Timestamp
Source
Message
1 2023-03-08 14:21:17,448 Controller DDS Recorder found!
2 2023-03-08 14:21:17,528 DDS ... Update from CLOSED to STOPPED
3 2023-03-08 14:22:20,639 Controller start [] command sent
4 2023-03-08 14:22:20,643 DDS ... Update from STOPPED to RUNNING

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 Recorder 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*
- *Developer Manual*
- *Release Notes*

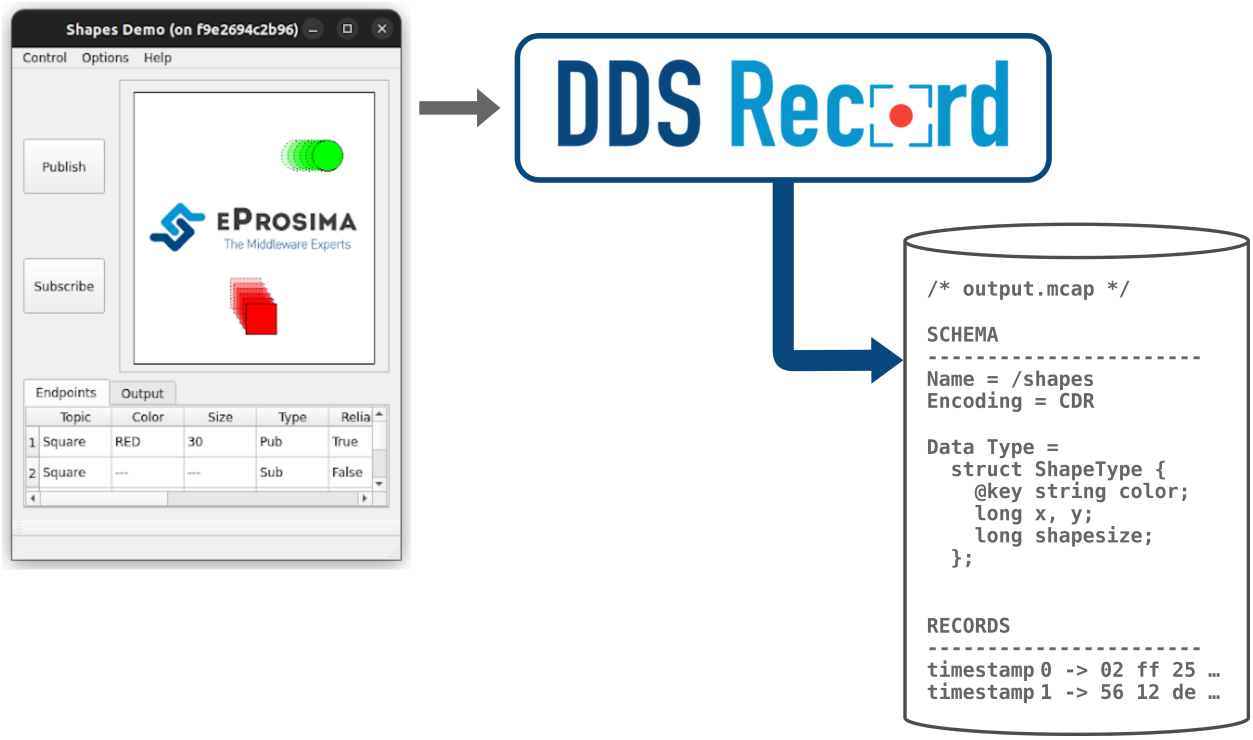
eProsima DDS Record is an end-user software application that efficiently saves DDS data published into a DDS environment in a MCAP format database. Thus, the exact playback of the recorded network events is possible as the data is linked to the timestamp at which the original data was published. At the moment, it is only possible to replay the data using external tools capable of interpreting the MCAP format, as the *eProsima DDS Record* does not provide a replay tool.

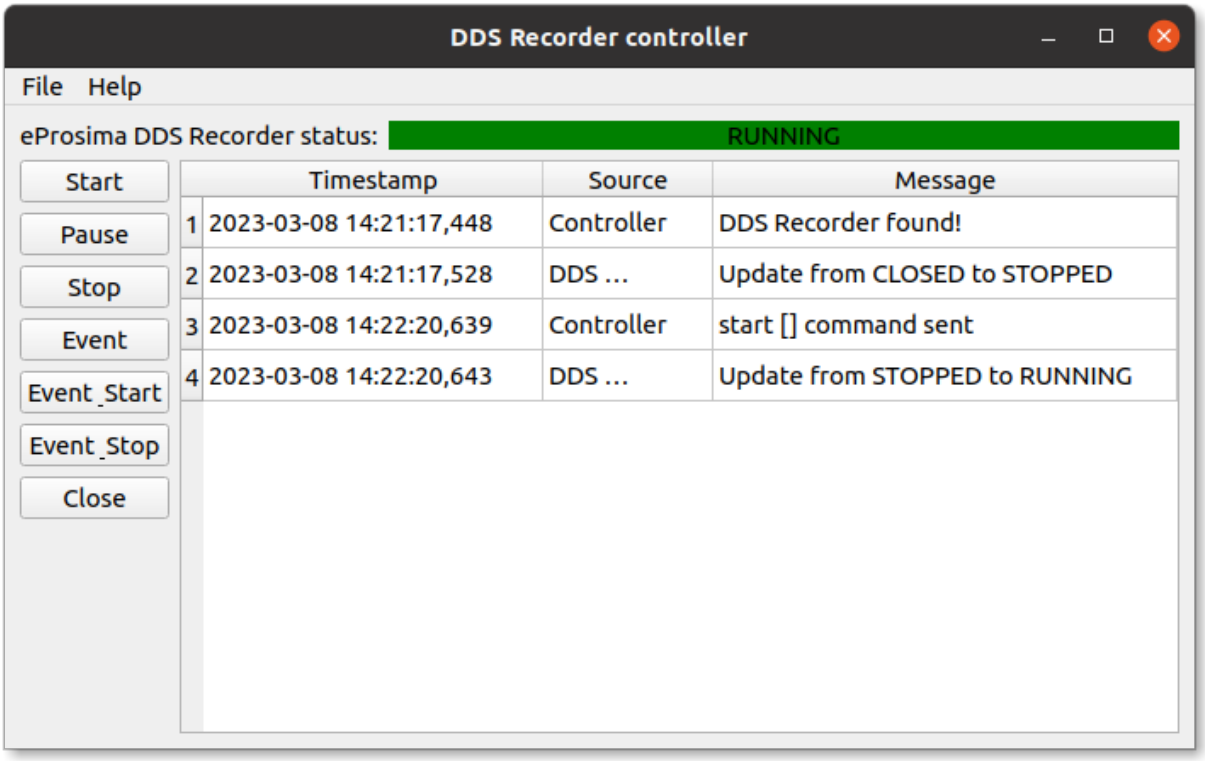
eProsima DDS Record is easily configurable and installed with a default setup, so that DDS topics, data types and entities are automatically discovered without the need to specify the types of data recorded. This is because the recording tool exploits the DynamicTypes functionality of eProsima Fast DDS,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

3.1 Overview

eProsima DDS Record 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.


Shapes Demo (on f9e2694c2b96)
Control Options Help
Publish
Subscribe
eProsima
The Middleware Experts
Endpoints Output
Topic Color Size Type Relia
1 Square RED 30 Pub True
2 Square --- --- Sub False
DDS Record
/* 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 Recorder controller
File Help
eProsima DDS Recorder status: RUNNING
Start
Pause
Stop
Event
Event_Start
Event_Stop
Close
Timestamp Source Message
1 2023-03-08 14:21:17,448 Controller DDS Recorder found!
2 2023-03-08 14:21:17,528 DDS … Update from CLOSED to STOPPED
3 2023-03-08 14:22:20,639 Controller start [] command sent
4 2023-03-08 14:22:20,643 DDS … Update from STOPPED to RUNNING

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 Recorder 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*
- *Developer Manual*
- *Release Notes*

3.5 DDS Recorder on Windows

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

3.6 DDS Recorder on Linux

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

3.7 Docker Image (recommended)

Warning: Docker image of *DDS Recorder* will be updated soon in eProsima Downloads website.

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

2. Extract the image by executing the following command:

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

   where `version` is the downloaded version of *DDS Recorder*.

3. Build a *DDS Recorder* configuration YAML file on the local machine. This will be the *DDS Recorder* configuration file that runs inside the Docker container. To continue this installation manual, let's use one of the configuration files provided in the *Tutorials* section. Open your preferred text editor and copy a full example from the *Tutorials* section 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 \
       -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 Recorder* running in the Docker container. If you want to terminate the application gracefully, just press `Ctrl+C` to stop the execution of *DDS Recorder*.

3.8 Getting Started

3.8.1 Project Overview

eProsima DDS Record 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.

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. The registration of the data type in the file allows the reading of the data (deserialization) when loading the file.

Since the *DDS Recorder* needs to record the data types of the data to be saved in the database, the user application must communicate the data types with which it is working by means of the type lookup service. 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`, `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* 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* recording application (*DDS Recorder* or `ddsrecorder`) has to offer.

Prerequisites

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

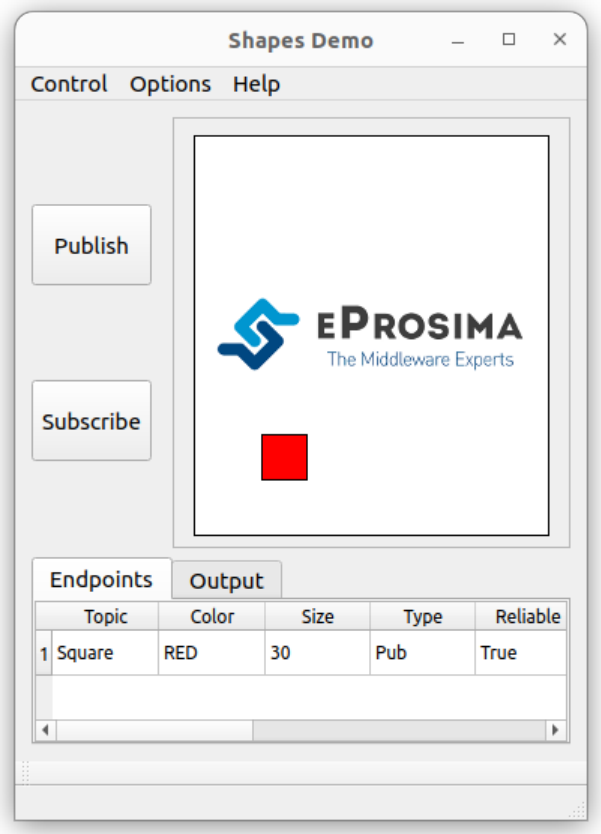
- *DDS Recorder on Windows*
- *DDS Recorder on Linux*
- *Docker Image (recommended)*

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*.

Recorder execution

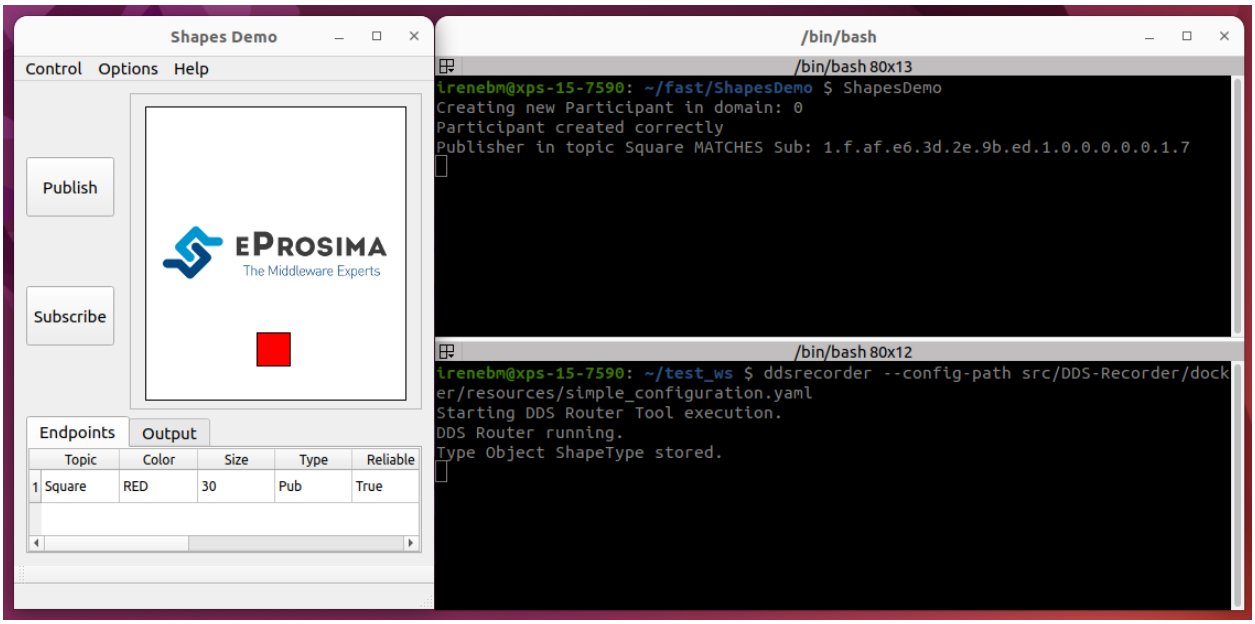
Launching a *DDS Recorder* instance is as easy as executing the following command:

```
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.



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.

3.9 Usage

eProsima DDS Record 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 Recorder* image. *Here* are the instructions to download the compressed *DDS Recorder* 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 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-ddsrecorder-installation>/install/setup.bash
```

Note: If Fast DDS, DDS Router and DDS Recorder have been installed in the system, these libraries would be sourced by default.

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

```
ddsrecorder
```

3.9.2 Closing Recording Application

SIGINT

To close *eProsima DDS Record*, press `Ctrl+C`. *eProsima DDS Record* 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.

Topic Filtering

DDS Recorder includes a mechanism to automatically detect which topics are being used in a DDS network. By automatically detecting these topics, a *DDS Recorder* creates internal DDS *Readers* for each topic in order to record the data published on each discovered topic.

Note: *DDS Recorder* entities are created with the QoS of the first Subscriber found in this Topic.

DDS Recorder allows filtering of DDS *Topics*, that is, it allows to define the DDS Topics' data that is going to be recorder by the application. This way, it is possible to define a set of rules in *DDS Recorder* to filter those data samples the user does not wish to save.

It is not mandatory to define such set of rules in the configuration file. In this case, a *DDS Recorder* will save all the data published under the topics that it automatically discovers within the DDS network to which it connects.

To define these data filtering rules based on the Topics to which they belong, the following lists are available:

- Allowed topics list (`allowlist`)
- Block topics list (`blocklist`)

These lists of topics stated above are defined by a tag in the *YAML* configuration file, which defines a *YAML* vector (`[]`). This vector contains the list of topics for each filtering rule. Each Topic is determined by its entries `name` and `type`, with only the first one being mandatory.

Topic entries	Data type	Default value
`name`	`string`	`-`
`type`	`string`	`"*"`

See *Topic* section for further information about the topic.

Note: Placing quotation marks around values in a YAML file is generally optional. However, values containing wildcard characters must be enclosed by single or double quotation marks.

Allow topic list

This is the list of topics that *DDS Recorder* will record, i.e. the data published under the topics matching the expressions in the `allowlist` will be saved by *DDS Recorder*.

Note: If no `allowlist` is provided, data will be recorded for all topics (unless filtered out in `blocklist`).

Block topic list

This is the list of topics that the *DDS Recorder* will block, that is, all data published under the topics matching the filters specified in the `blocklist` will be discarded by the *DDS Recorder* and therefore will not be recorded.

This list takes precedence over the `allowlist`. If a topic matches an expression both in the `allowlist` and in the `blocklist`, the `blocklist` takes precedence, causing the data under this topic to be discarded.

Example of usage - Allowlist and blocklist collision:

In the following example, the `HelloWorldTopic` topic is both in the `allowlist` and (implicitly) in the `blocklist`, so according to the `blocklist` preference rule this topic is blocked. Moreover, only the topics present in the allowlist are relayed, regardless of whether more topics are dynamically discovered in the DDS network. In this case the forwarded topics are `AllowedTopic1` with data type `Allowed` and `AllowedTopic2` regardless of its data type.

```yaml
allowlist:
  - name: AllowedTopic1
    type: Allowed
  - name: AllowedTopic2
    type: "*"
  - name: HelloWorldTopic
    type: HelloWorld

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

Built-in Topics

Apart from the dynamic DDS topics discovered in the network, the discovery phase can be accelerated by using the builtin topic list (`builtin-topics`). By defining topics in this list, the *DDS Recorder* will create the DataWriters and DataReaders in recorder initialization.

The builtin-topics list is defined in the same form as the `allowlist` and `blocklist`.

This feature also allows to manually force the QoS of a specific topic, so the entities created in such topic follows the specified QoS and not the one first discovered.

Topic Quality of Service

For every topic contained in this list, both `name` and `type` must be specified and contain no wildcard characters. Apart from these values, the tag `qos` under each topic allows to configure the following values:

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`
History Depth	`depth`	*integer*	*default value*	•
Partitions	`partitions`	*bool*	`false`	Topic with / without partitions
Ownership	`ownership`	*bool*	`false`	`EXCLUSIVE_OWNERSHIP_QOS` / `SHARED_OWNERSHIP_QOS`
Key	`keyed`	*bool*	`false`	Topic with / without key
Downsampling	`downsampling`	*integer*	*default value*	Downsampling factor
Max Reception Rate	`max-reception-rate`	*float*	*default value*	Maximum sample reception rate [Hz]

Example of usage:

```
builtin-topics:
  - name: HelloWorldTopic
    type: HelloWorld
    qos:
      reliability: true       # Use QoS RELIABLE
      durability: true        # Use QoS TRANSIENT_LOCAL
      depth: 100              # Use History Depth 100
      partitions: true        # Topic with partitions
      ownership: false        # Use QoS SHARED_OWNERSHIP_QOS
      keyed: true             # Topic with key
      downsampling: 4         # Keep 1 of every 4 samples
      max-reception-rate: 10  # Discard messages if less than 100ms elapsed␣
↪since the last sample was processed
```

DDS Domain

Tag `domain` configures the *Domain Id*.

```
domain: 101
```

Recorder Configuration

Configuration of data writing in the database.

Output File

The recorder output file does support the following configurations:

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`

When DDS Recorder application is launched (or when remotely controlled, every time a `start` command is received), a temporary file with `filename` name and `.mcap.tmp~` extension is created in `path`. This file is not readable until the application is terminated (or a `stop` / `close` command is received). On such event, the temporal file is renamed to `filename` with `.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.

Max Reception Rate

Limits the frequency [Hz] at which samples are processed, by discarding messages received before `1/max-reception-rate` seconds have elapsed since the last processed message was received. When specified, `max-reception-rate` is set for all topics without distinction, but a different value can also set for a particular topic under the `qos` configuration tag within the builtin-topics list. This parameter only accepts integer values, and its default value is `0` (no limit).

Downsampling

Reduces the sampling rate of the received data by keeping *1* out of every *n* samples received (per topic), where *n* is the value specified in `downsampling`. If `max-reception-rate` is also set, downsampling applies to messages that already managed to pass this filter. When specified, this downsampling factor is set for all topics without distinction, but a different value can also set for a particular topic under the `qos` configuration tag within the builtin-topics list. This parameter only accepts positive integer values, and its default value is 1 (no downsampling).

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 topics.	`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` `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 order not to discard the samples received from this topic, it is possible to keep a limited number of samples in an internal circular buffer that stores those samples that do not yet have a known data type. The `max-pending-samples` parameter allows to configure the size of this circular buffer **for each topic** that is discovered. The default value is equal to `5000` samples.

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`.

General Example

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

```yaml
dds:
  domain: 0

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

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

  builtin-topics:
    - name: "HelloWorldTopic"
      type: "HelloWorld"
      qos:
        reliability: true
        durability: true
        keyed: false
        partitions: true
        ownership: false
        downsampling: 4
        max-reception-rate: 10

recorder:
  output:
    filename: "output"
    path: "."

  buffer-size: 50
  event-window: 60
  log-publish-time: false
```

(continues on next page)

(continued from previous page)

```
  downsampling: 3
  max-reception-rate: 20

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
```

3.10.2 Fast DDS Configuration

As mentioned before, the *DDS Recorder* requires the topic types in order to be able to record the data of such topics. This requires that the user application needs to be configured to send the required 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:

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

3.11 Remote Control

The *DDS Recorder* application from *eProsima DDS Record* 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.
- **STOPPED**: The application is running but not receiving data.

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.
- **stop**: Changes to `STOPPED` 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.
- **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.

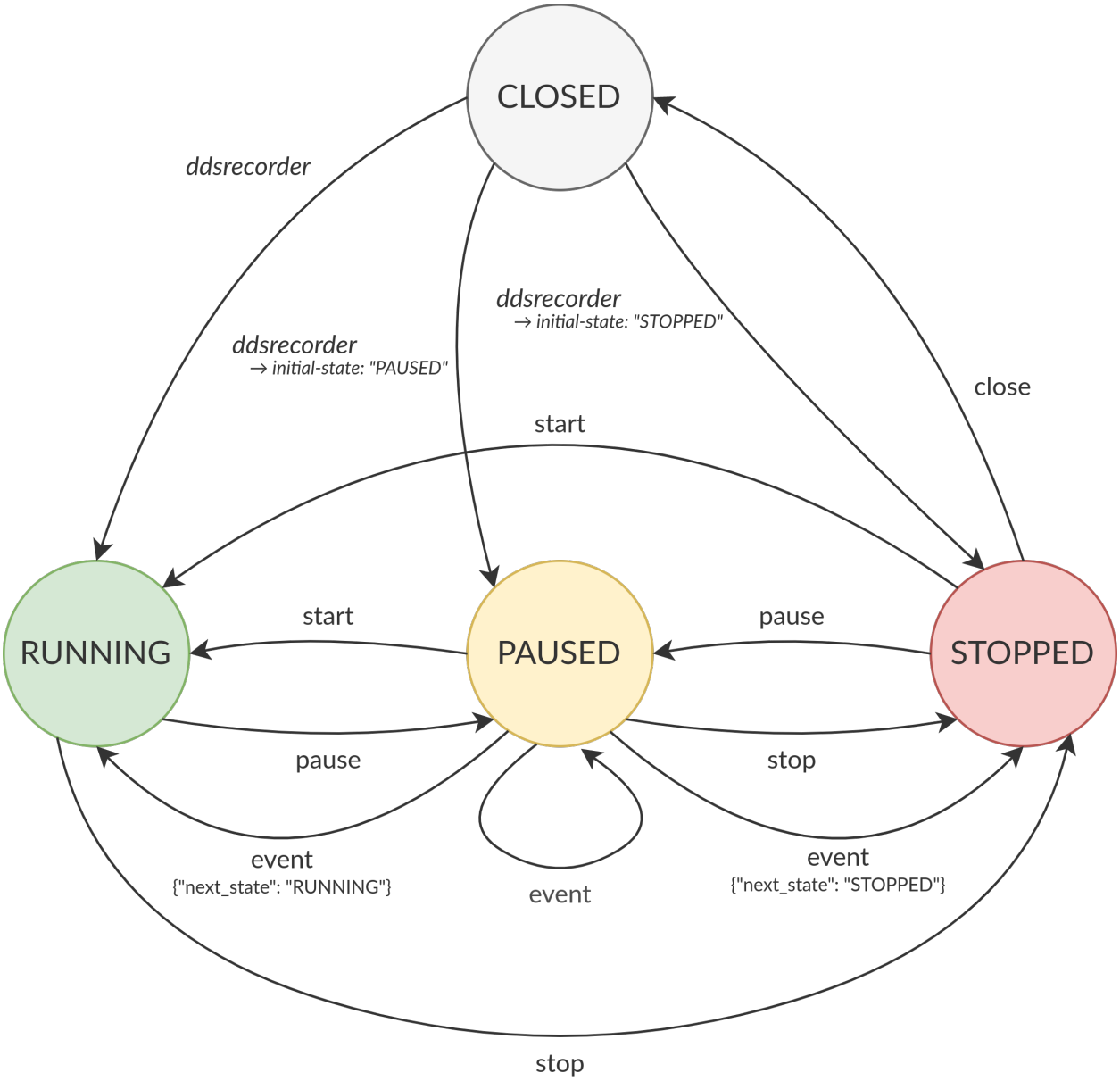
3.11.2 DDS Controller Data Types

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.


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

- Command topic:
  - Topic name: Specified in `command-topic-name` configuration parameter (Default: `/ddsrecorder/command`)
  - Topic type name: `DdsRecorderCommand`
  - 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` `stop` `event` `close`
`args`	Arguments of the command. This arguments should contain a JSON serialized string.	`string`	· `event` command: `{"next_state": "RUNNING"}` `{"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:

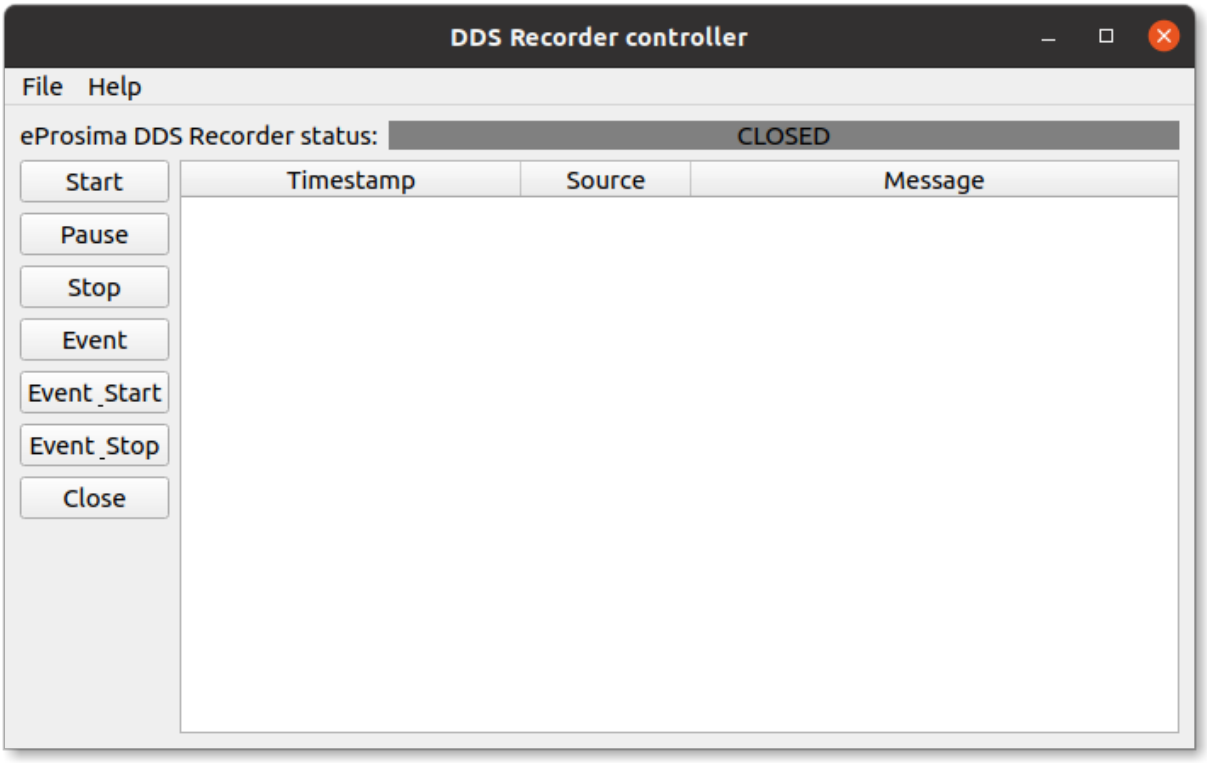
Argument	Description	Data type	Possible values
`previous`	Previous status of the *DDS Recorder*.	`string`	`RUNNING` `PAUSED` `STOPPED`
`current`	Current status of the *DDS Recorder*.	`string`	`RUNNING` `PAUSED` `STOPPED`
`info`	Additional information related to the state change. (Unused)	`string`	-

3.11.3 DDS Recorder remote controller application

eProsima DDS Record 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 *DDS Recorder* 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:



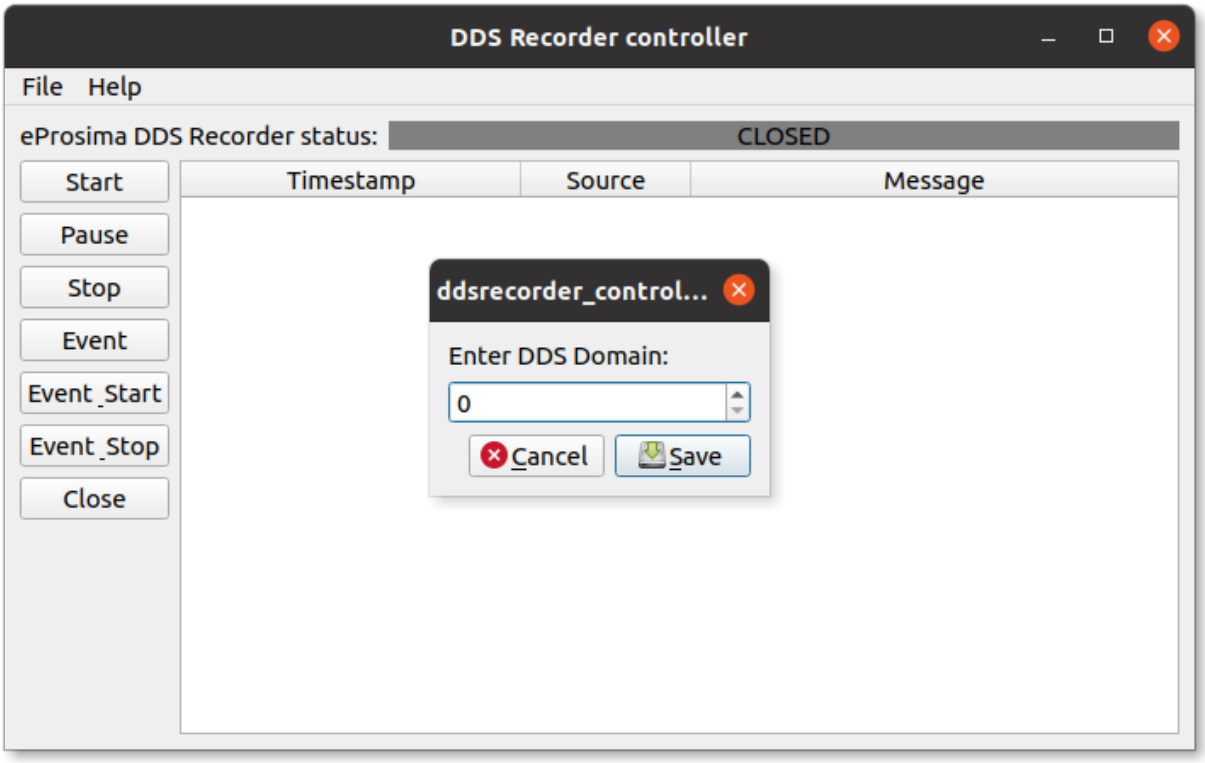
If the controller should function in a domain different than the default one (`0`), change it by clicking `File->DDS Domain` and introducing the one desired:

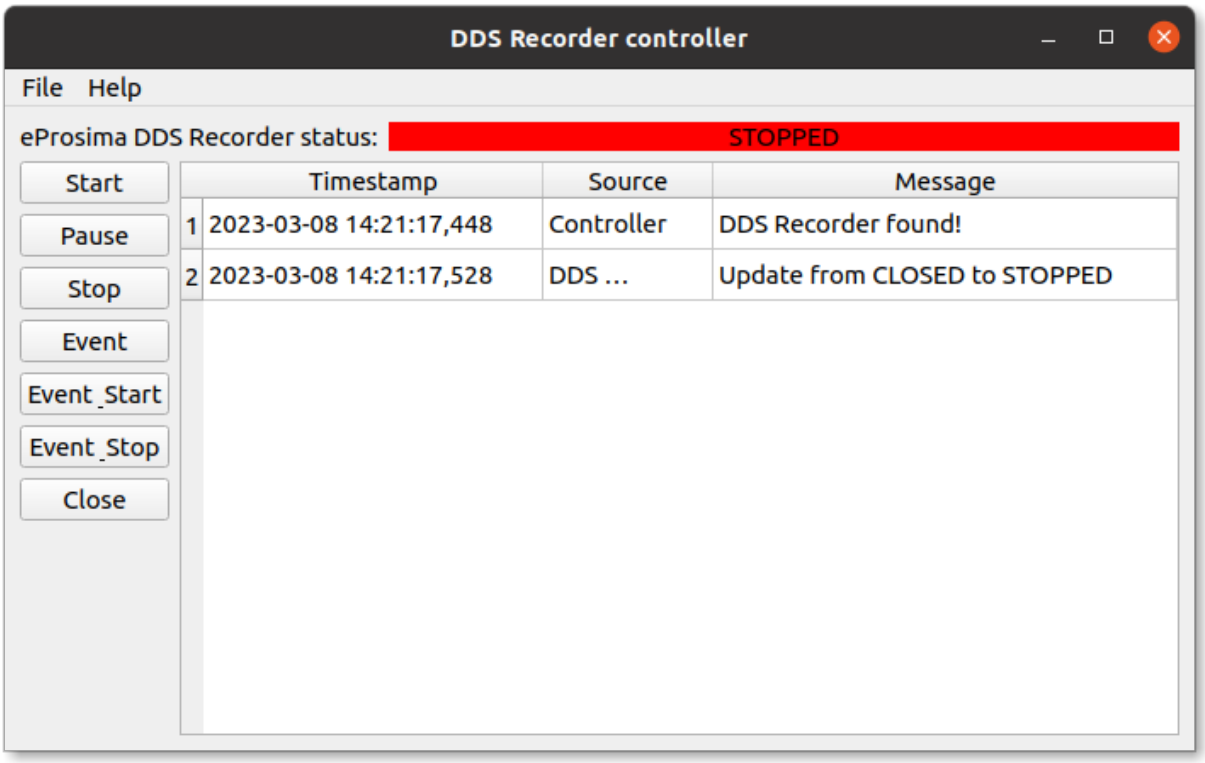
When a *DDS Recorder* instance is found in the domain, a message is displayed in the logging panel:

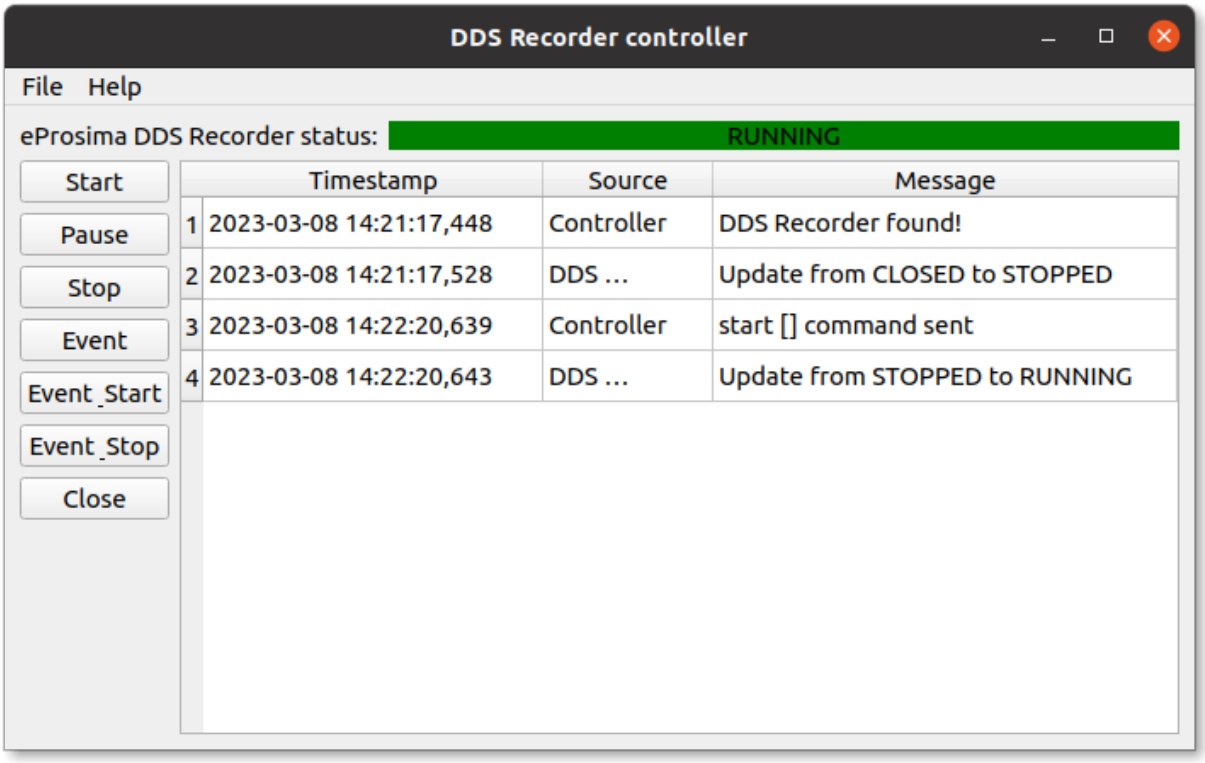
From this point on, it is possible to interact with the recorder application by pushing any of the buttons appearing on the left. Every command sent is reflected in the logging panel and, additionally, the recorder application publishes its current status with every state transition undergone. This can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

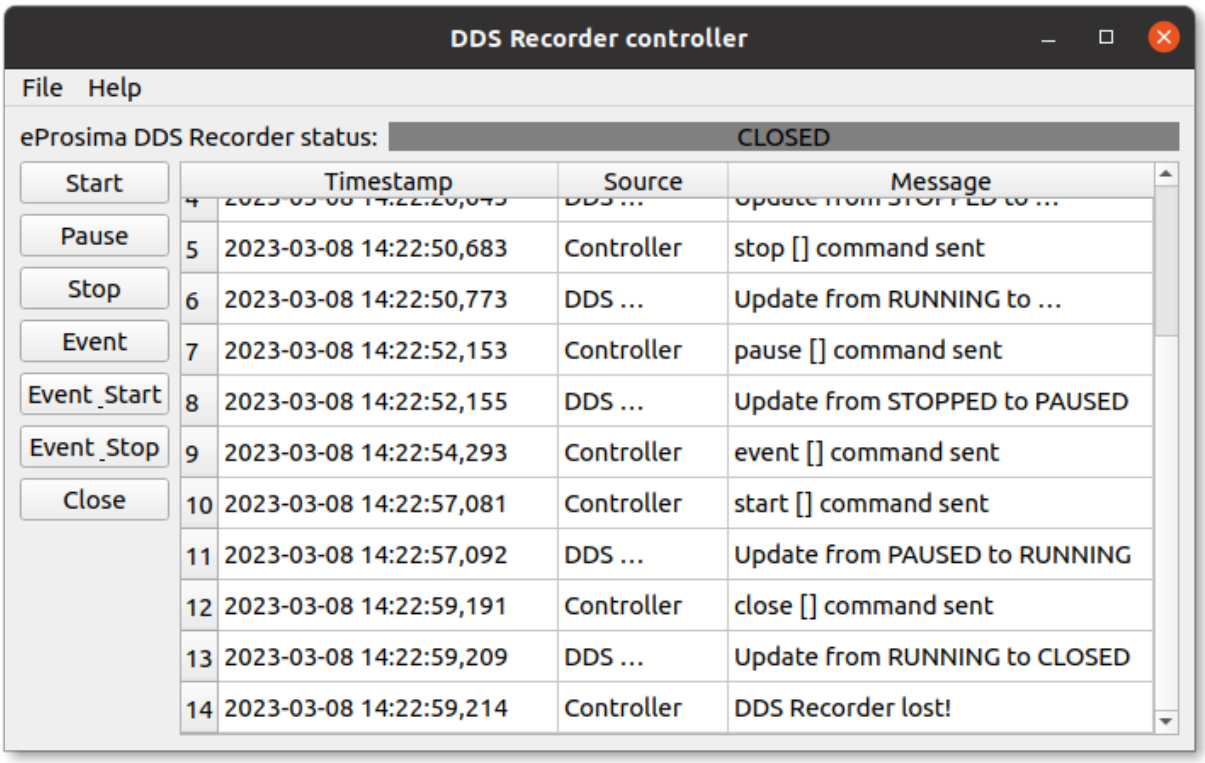
By clicking on `Stop` button, the recorder application ceases recording, but can be commanded to `Start` / `Pause` whenever wished. Once the user has finished all recording activity, it is possible to `Close` the recorder and free all resources used by the application:

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.


DDS Recorder controller
File
Help
eProsima DDS Recorder status:
CLOSED
Start
Pause
Stop
Event
Event_Start
Event_Stop
Close
Timestamp
Source
Message
ddsrecorder_control...
Enter DDS Domain:
0
Cancel
Save


DDS Recorder controller
File
Help
eProsima DDS Recorder status:
STOPPED
Start
Pause
Stop
Event
Event_Start
Event_Stop
Close
Timestamp
Source
Message
1
2023-03-08 14:21:17,448
Controller
DDS Recorder found!
2
2023-03-08 14:21:17,528
DDS ...
Update from CLOSED to STOPPED


DDS Recorder controller
File Help
eProsima DDS Recorder status: RUNNING
Start
Pause
Stop
Event
Event_Start
Event_Stop
Close
Timestamp Source Message
1 2023-03-08 14:21:17,448 Controller DDS Recorder found!
2 2023-03-08 14:21:17,528 DDS ... Update from CLOSED to STOPPED
3 2023-03-08 14:22:20,639 Controller start [] command sent
4 2023-03-08 14:22:20,643 DDS ... Update from STOPPED to RUNNING


DDS Recorder controller
File Help
eProsima DDS Recorder status: CLOSED
Start
Pause
Stop
Event
Event_Start
Event_Stop
Close
Timestamp Source Message
5 2023-03-08 14:22:50,683 Controller stop [] command sent
6 2023-03-08 14:22:50,773 DDS ... Update from RUNNING to ...
7 2023-03-08 14:22:52,153 Controller pause [] command sent
8 2023-03-08 14:22:52,155 DDS ... Update from STOPPED to PAUSED
9 2023-03-08 14:22:54,293 Controller event [] command sent
10 2023-03-08 14:22:57,081 Controller start [] command sent
11 2023-03-08 14:22:57,092 DDS ... Update from PAUSED to RUNNING
12 2023-03-08 14:22:59,191 Controller close [] command sent
13 2023-03-08 14:22:59,209 DDS ... Update from RUNNING to CLOSED
14 2023-03-08 14:22:59,214 Controller DDS Recorder lost!

3.12 Nomenclature

3.12.1 DDS Recorder nomenclature

MCAP Modular container file format for heterogeneous timestamped data.

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.

3.12.2 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.

3.13 Tutorials

This section will provide a collection of tutorials both on the use and application of the basic functionality, as well as on the exploitation of *DDS Recorder* for more advanced users.

3.13.1 Configuring Fast DDS DynamicTypes for data recording

Background

Currently, the DDS Recorder only stores DDS data whose data type is set as a DynamicType. The reason for this is that, without the need for the user to set the data type in the DDS Recorder, the DDS Recorder can access it via the type lookup service. Thus, the user will be able to record the published data using the *DDS Recorder* tool of the *eProsima DDS Record* software.

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

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 .

Prerequisites

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

If *eProsima DDS Record* was installed using the recommended installation the environment is source 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-ddsrecorder-installation>/install/setup.bash
```

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
```

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;
```

(continues on next page)

(continued from previous page)

```
    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:
            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_;
    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;
}
```

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);
```

Running the application

Recording samples with DDS Recorder

Open two terminals:

- In the first terminal, run an instance of the described publisher:

```
source install/setup.bash
cd DDS-Recorder/build/TypeLookupService
./TypeLookupService
```

- In the second terminal, run the ddsrecorder:

```
source install/setup.bash
ddsrecorder
```

Stop the *DDS Recorder* at any time to save the output MCAP file containing the recorded data.

Publisher <-> Subscriber

Open two terminals:

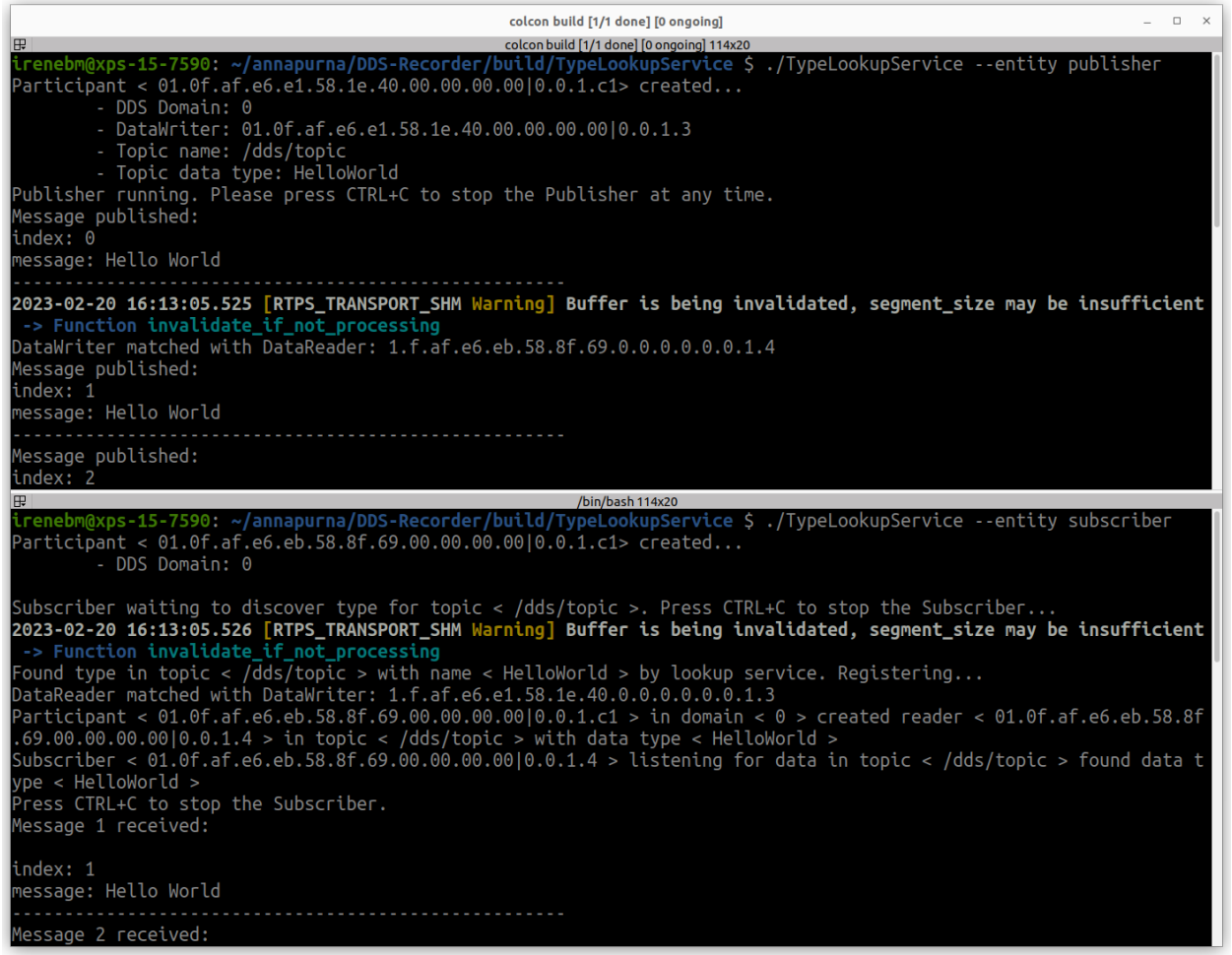
- In the first terminal, run the DDS Publisher:

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

- In the second terminal, run the DDS Subscriber:

```
source install/setup.bash
cd DDS-Recorder/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.


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

3.13.2 Visualize recorded data with Foxglove

Background

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

Prerequisites

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

- *DDS Recorder on Windows*
- *DDS Recorder 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

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:

```
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.

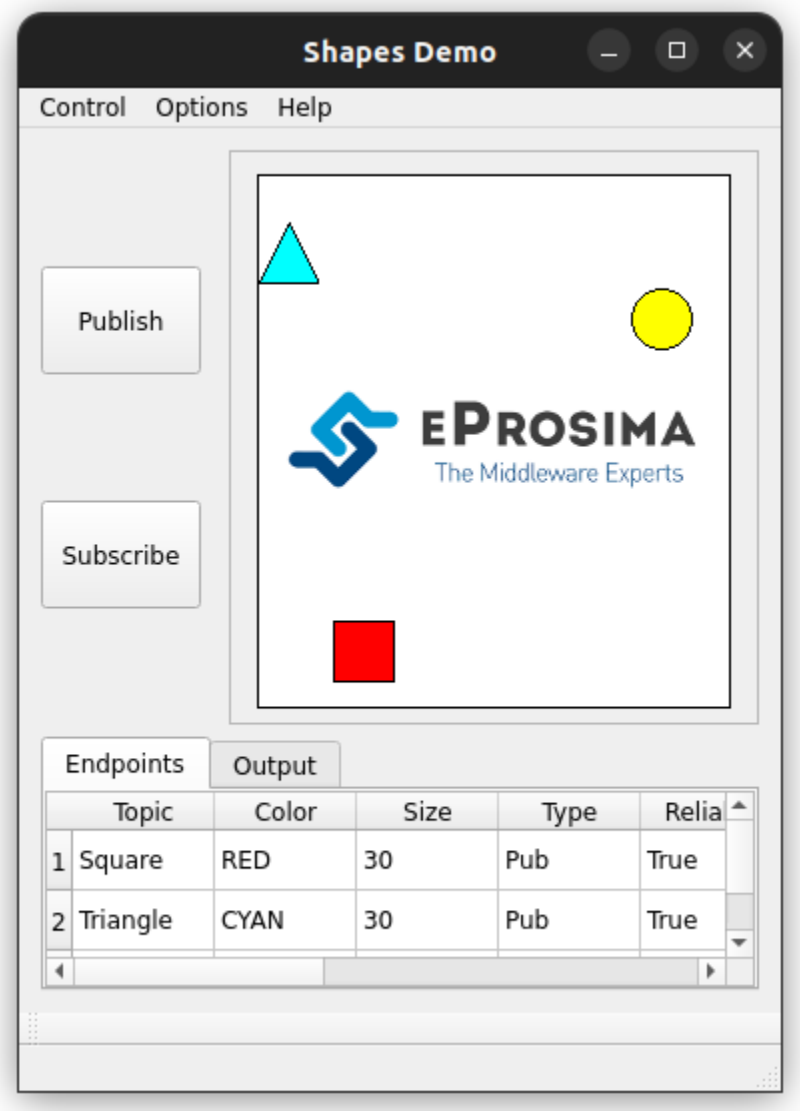
Running the application

Start ShapesDemo

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

```
ShapesDemo
```

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



Recorder execution

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

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

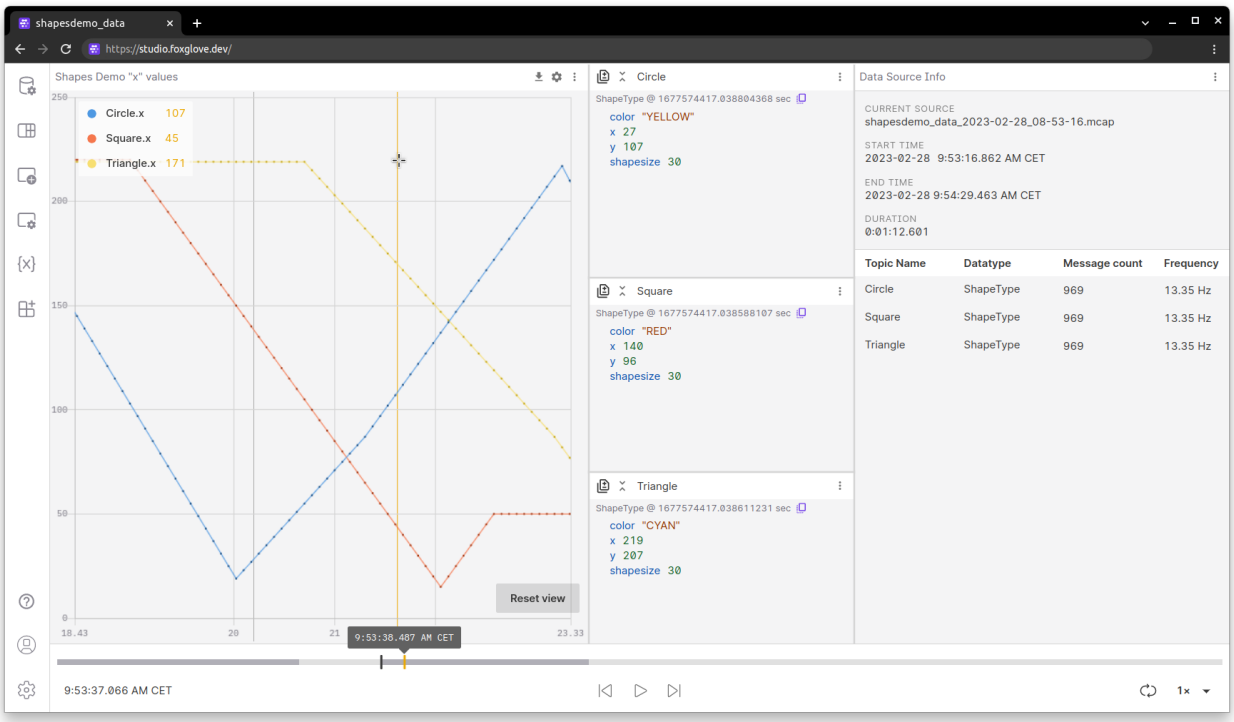
Once you have all the desired data, close the *DDS Recorder* application with `Ctrl`+C.

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.



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

3.14 Linux installation from sources

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

3.14.1 Dependencies installation

DDS Recorder depends on *eProsima Fast DDS* library and certain Debian packages. This section describes the instructions for installing *DDS Recorder* 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* 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* 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* 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* 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* 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 has the following dependencies, when installed from sources in a Linux environment:

- *Asio and TinyXML2 libraries*
- *OpenSSL*
- *yaml-cpp*
- *SWIG* [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 Recorder* 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
```

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 Recorder* 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
```

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* 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
```

3.14.2 Colcon installation (recommended)

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

   ```
   mkdir -p ~/DDS-Record/src
   cd ~/DDS-Record
   wget https://raw.githubusercontent.com/eProsima/DDS-Recorder/main/ddsrecorder.repos
   vcs import src < ddsrecorder.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* 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 both *DDS Recorder* and its *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.

3.14.3 CMake installation

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

Local installation

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

```
mkdir -p ~/DDS-Record/src
mkdir -p ~/DDS-Record/build
cd ~/DDS-Record
wget https://raw.githubusercontent.com/eProsima/DDS-Recorder/main/ddsrecorder.repos
vcs import src < ddsrecorder.repos
```

2. Compile all dependencies using CMake.

   - Foonathan memory

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

   - Fast CDR

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

   - Fast DDS

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

   - Dev Utils

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

# C++ Utils
```

(continues on next page)

(continued from previous page)

```
cd ~/DDS-Record
mkdir build/cpp_utils
cd build/cpp_utils
cmake ~/DDS-Record/src/dev-utils/cpp_utils -DCMAKE_INSTALL_PREFIX=~/DDS-
↪Record/install -DCMAKE_PREFIX_PATH=~/DDS-Record/install
cmake --build . --target install
```

- DDS Pipe

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

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

# ddspipe_yaml
cd ~/DDS-Record
mkdir build/ddspipe_yaml
cd build/ddspipe_yaml
cmake ~/DDS-Record/src/ddspipe/ddspipe_yaml -DCMAKE_INSTALL_PREFIX=~/DDS-
↪Record/install -DCMAKE_PREFIX_PATH=~/DDS-Record/install
cmake --build . --target install
```

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

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

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

# ddsrecorder
```

(continues on next page)

(continued from previous page)

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

Note: By default, *eProsima DDS Record* 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
mkdir build/fastdds_python
cd build/fastdds_python
cmake ~/DDS-Record/src/Fast-DDS-python/fastdds_python -DCMAKE_INSTALL_PREFIX=~/DDS-
↪Record/install -DCMAKE_PREFIX_PATH=~/DDS-Record/install
cmake --build . --target install

# Remote Controller Application
cd ~/DDS-Record
mkdir build/controller_tool
cd build/controller_tool
cmake ~/DDS-Record/src/ddsrecorder/controller/controller_tool -DCMAKE_INSTALL_
↪PREFIX=~/DDS-Record/install -DCMAKE_PREFIX_PATH=~/DDS-Record/install
cmake --build . --target install
```

Global installation

To install *eProsima DDS Record* 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`, and change the first in the configuration step of `foonathan_memory_vendor` to the following:

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

3.14.4 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
```

Be sure that this executable has execution permissions.

3.15 Windows installation from sources

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

3.15.1 Dependencies installation

eProsima DDS Record depends on *eProsima Fast DDS* library and certain Debian packages. This section describes the instructions for installing *eProsima DDS Record* 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]
- *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.

Gtest

Gtest is a unit testing library for C++. By default, *eProsima DDS Record* 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 has the following dependencies, when installed from sources in a Windows environment:

- *Asio and TinyXML2 libraries*
- *OpenSSL*
- *yaml-cpp*
- *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 Recorder* 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
```

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* 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.

3.15.2 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-Recorder` directory and download the `.repos` file that will be used to install *eProsima DDS Record* and its dependencies:

   ```
   mkdir <path\to\user\workspace>\DDS-Recorder
   cd <path\to\user\workspace>\DDS-Recorder
   mkdir src
   wget https://raw.githubusercontent.com/eProsima/DDS-Recorder/main/ddsrecorder.repos
   vcs import src < ddsrecorder.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* 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: 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.

3.15.3 CMake installation

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

Local installation

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

```
mkdir <path\to\user\workspace>\DDS-Record
mkdir <path\to\user\workspace>\DDS-Record\src
mkdir <path\to\user\workspace>\DDS-Record\build
cd <path\to\user\workspace>\DDS-Record
wget https://raw.githubusercontent.com/eProsima/DDS-Recorder/main/ddsrecorder.repos
vcs import src < ddsrecorder.repos
```

2. Compile all dependencies using CMake.

   - Foonathan memory

```
cd <path\to\user\workspace>\DDS-Record
mkdir build\foonathan_memory_vendor
cd build\foonathan_memory_vendor
cmake <path\to\user\workspace>\DDS-Record\src\foonathan_memory_vendor -
↪DCMAKE_INSTALL_PREFIX=<path\to\user\workspace>\DDS-Record\install ^
    -DBUILD_SHARED_LIBS=ON
cmake --build . --config Release --target install
```

   - Fast CDR

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

   - Fast DDS

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

   - Dev Utils

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

# C++ Utils
```

(continues on next page)

(continued from previous page)

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

- DDS Pipe

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

Global installation

To install *eProsima DDS Record* 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`

3.15.4 Run an application

If the *eProsima DDS Record* was compiled using colcon, when running an instance of a *DDS Recorder*, the colcon overlay built in the dedicated `DDS-Record` 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 an instance of a *DDS Recorder* 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` the *eProsima DDS Record*, *Fast DDS*, *Fast CDR*, *DDS Pipe* installation directories:

- *Fast DDS*: `C:\\Program Files\\fastrtps`
- *Fast CDR*: `C:\\Program Files\\fastcdr`
- *DDS Pipe*: `C:\\Program Files\\ddspipe`
- *eProsima DDS Record*: `C:\\Program Files\\ddsrecord`

3.16 CMake options

eProsima DDS Record provides numerous CMake options for changing the behavior and configuration of *eProsima DDS Record*. These options allow the developer to enable/disable certain *eProsima DDS Record* 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* 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 *eProsima DDS Record* remote controller application.	`OFF` `ON`	`OFF`
`BUILD_DOCS`	Build the *eProsima DDS Record* documentation.	`OFF` `ON`	`OFF`
`BUILD_TESTS`	Build the *eProsima DDS Record* tools and documentation tests.	`OFF` `ON`	`OFF`
`LOG_INFO`	Activate *eProsima DDS Record* 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`

3.17 Notes

3.17.1 Version v0.1.0

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

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.

3.18 Glossary

LAN Local Area Network

INDEX