rmw_connextdds

Release 0.4.0

Andrea Sorbini <asorbini@rti.com>

Mar 30, 2021

TABLE OF CONTENTS

CHAPTER

ONE

USER MANUAL

1.1 Installation

1.1.1 Installation Requirements

ROS 2 Releases

`rmw_connextdds` and `rmw_connextddsmicro` both support multiple versions of ROS 2.

The following table summarizes the installation methods available for each release:

ROS 2 Release	From Source	Binary
Rolling	Yes	Yes
Foxy	Yes	No
Eloquent	Yes	No
Dashing	Yes	No

RTI Connext DDS Requirements

Both RMW packages require the appropriate version of RTI Connext DDS to be available on the build and target systems.

`rmw_connextdds` requires RTI Connext DDS Professional (version 5.3.1 or later), while `rmw_connextddsmicro` requires RTI Connext DDS Micro (version 3.0.3 or later).

The installations must be made available via environment variables. The RMW packages will be skipped, and no library will be generated if the required product is not installed.

Variables can also be specified as arguments to `cmake` (e.g. `--cmake-args -DVAR=VALUE`), in which case they will take precedence over values in the environment.

RTI Connext DDS Professional Requirements

RTI Connext DDS Professional 5.3.1 or later must be installed in order to build and use `rmw_connextdds`.

The installation path must be specified using either `CONNEXTDDS_DIR`, or `NDDSHOME`. `CONNEXTDDS_DIR` takes precendence over `NDDSHOME`.

The build system will try to guess the correct target architecture based on the host build environment. Use `CONNEXTDDS_ARCH` to manually specify the target build architecture. This value must match one of the set of libraries installed under `${CONNEXTDDS_DIR}/lib`.

Note: The `rtisetenv_<architecture>` scripts shipped with RTI Connext DDS can be used to automatically set `NDDSHOME` (in addition to configuring `PATH` and `LD_LIBRARY_PATH` to load the Connext libraries and tools).

Even though they are "architecture-specific", these scripts will **not** set `CONNEXTDDS_ARCH`.

Note: A warning will be generated by package `rti_connext_dds_cmake_module` if no Connext installation is specified.

RTI Connext DDS Micro Requirements

RTI Connext DDS Micro 3.0.3 or later must be installed in order to build and use `rmw_connextddsmicro`.

Specify the installation path using `RTIMEHOME` (optional). If unspecified, the build system will inspect the installation of RTI Connext DDS Professional (if one is available) for a directory whose name starts with `rti_connext_dds_micro-`. The first result found will be used.

Similarly to RTI Connext DDS Professional, the build system will try to guess the correct target architecture based on the host build environment. Use `RTIME_TARGET_NAME` to manually specify the target build architecture. This value must match one of the set of libraries installed under `${RTIMEHOME}/lib`.

If Micro has never been built from source and no binary libraries are available under `${RTIMEHOME}/lib`, the build system will automatically try to build Micro from source for the detected target build environment. In this case, `RTIME_TARGET_NAME` will be automatically set to one of the following values:

Host System	RTIME_TARGET_NAME
Linux	`Linux`
macOS	`Darwin`
Windows	`Windows`

If you encounter any issue building Micro this way, please generate libraries by hand using the included `rtime-make` script (see the official documentation).

Note: The automatic detection of RTI Connext DDS Micro can be disabled by setting `RTIMEHOME` to an invalid (i.e. non-existing) path.

Warning: The automatic detection of RTI Connext DDS Micro is only available when RTI Connext DDS Professional 6.0.0 or later is installed.

1.1.2 Binary installation

Binary quick-start

These instructions are for Ubuntu 20.04 running on an x86 64-bit host.

1. Install your desired ROS 2 distributions, e.g. to install Rolling:

```
# Download and trust the ROS packaging master key.
sudo apt update && sudo apt install curl gnupg2 lsb-release
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-
↪key add -

# Add the ROS 2 repository to your apt repositories.
sudo sh -c 'echo "deb [arch=$(dpkg --print-architecture)] http://packages.ros.org/
↪ros2/ubuntu $(lsb_release -cs) main" > /etc/apt/sources.list.d/ros2-latest.list'

# Update package database.
sudo apt update

# Use package ros-rolling-ros-base for a more minimal installation.
sudo apt install ros-rolling-desktop
```

Note: Refer to the ROS 2 documentation for more installation options.

2. Install `rmw_connextdds`, e.g. for Rolling:

```
sudo apt install ros-rolling-rmw-connextdds
```

3. Run ROS 2 applications with RTI Connext DDS Professional 5.3.1, e.g.:

```
RMW_IMPLEMENTATION=rmw_connextdds ros2 run demo_nodes_cpp talker
```

Supported ROS 2 Releases

This method of installation is only available for `rmw_connextdds`, and only for ROS 2 releases starting with Galactic. It can also be used by users of the Rolling distribution.

At the moment, binary packages are only available for Ubuntu 20.04 x86 64-bit.

`rmw_connextdds` must be built from source on all other platforms (e.g. aarch64), operating systems (other Linux distributions, macOS, and Windows), and previous ROS 2 releases (starting with Dashing). See *Installation from source* for more information.

`rmw_connextddsmicro` must always be built from source.

Warning: The binary version of `rmw_connextdds` is built with RTI Connext DDS Professional 5.3.1. You must build it from source in order to use it with a more recent version (e.g. 6.0.1).

1.1.3 Installation from source

Source quick-start

1. Install ROS 2 on your system. Refer to the ROS 2 documentation for available options.
2. Load ROS 2 into the shell environment, e.g. if you installed Foxy using the binary packages available for Ubuntu:

```
source /opt/ros/foxy/setup.bash
```

2. Install RTI Connext DDS Professional and load it in your environment, e.g. using `NDDSHOME`:

```
export NDDSHOME=~/rti_connext_dds-6.0.1
```

You can also use variable `CONNEXTDDS_DIR` instead of `NDDSHOME`. `CONNEXTDDS_DIR` will take precedence if set.

Skip this step if you don't want to build `rmw_connextdds` (e.g. if you only want to build `rmw_connextddsmicro`).

Note: `rmw_connextdds` will only be compiled if RTI Connext DDS Professtional 5.3.1 or later is installed and configured in the environment.

Replace the example path with your local Connext installation.

See *RTI Connext DDS Professional Requirements* for more information about installing and configuring RTI Connext DDS Professional.

---

3. Optionally, install and load RTI Connext DDS Micro in your environment, e.g.:

```
export RTIMEHOME=~/rti_connext_dds_micro-3.0.3
```

Often, `RTIMEHOME` can be automatically guessed from the installation of RTI Connext DDS Professional.

This step is only necessary if you are using an "exported" installation of RTI Connext DDS Micro 3.x outside of an RTI Connext DDS 6.x installation, or if you are planning on building `rmw_connextddsmicro` only (thus you are not loading RTI Connext DDS Professional in your environment).

Note: `rmw_connextddsmicro` will only be compiled if RTI Connext DDS Micro 3.0.3 or later is installed and configured in the environment.

Replace the example path with the location of your Micro installation. See *RTI Connext DDS Micro Requirements* for more information about installing and configuring RTI Connext DDS Micro.

---

4. Create an overlay directory and clone the repository:

```
mkdir -p ~/ros2_connextdds/src/ros2
cd ~/ros2_connextdds
git clone https://github.com/ros2/rmw_connextdds.git src/ros2/rmw_connextdds
```

5. Build the RMW:

```
colcon build
```

6. Load the generated environment script:

```
source ~/ros2_connextdds/install/setup.bash
```

7. Run ROS 2 applications with RTI Connext DDS Professional by setting `RMW_IMPLEMENTATION=rmw_connextdds`, e.g.:

```
RMW_IMPLEMENTATION=rmw_connextdds ros2 run demo_nodes_cpp talker
```

8. Run ROS 2 applications with RTI Connext DDS Micro by setting `RMW_IMPLEMENTATION=rmw_connextddsmicro`, e.g.:

```
RMW_IMPLEMENTATION=rmw_connextddsmicro \
RMW_CONNEXT_INITIAL_PEERS=_shmem:// \
  ros2 run demo_nodes_cpp listener
```

Supported ROS 2 Releases

Each versions of ROS 2 supported by `rmw_connextdds` and `rmw_connextddsmicro` is stored in a dedicated branch of ros2/rmw_connextdds.

The following table summarizes the available branches, and the level of support offered for each ROS 2 release:

ROS 2 Release	Branch	Status
Rolling	`master`	Developed
Foxy	`foxy`	LTS (May 2023)
Eloquent	`eloquent`	EOL (Nov 2020)
Dashing	`dashing`	LTS (May 2021)

Branch `master` is actively developed and maintained. It is used to create other branches for specific ROS 2 releases (starting from Galactic).

Branches marked as `LTS` will receive updates for critical bug fixes and important patches only (until they reach `EOL`).

Branches marked as `EOL` will not receive any more updates.

1.2 Build Options

`rmw_connextdds` and `rmw_connextddsmicro` expose a few compile-time options that can be passed to `cmake` to modify the source code included by each compilation unit.

You can specify these options to `colcon build` using argument `--cmake-args`, e.g.:

```
colcon build --cmake-args -DRMW_CONNEXT_LOG_MODE=printf
```

If you already built your workspace, use option `--cmake-force-configure` to make sure to run `cmake`'s configure step again (or delete the `build/` directory altogether, for a more "drastic" approach).

These options have the same effect for both `rmw_connextdds`, and `rmw_connextddsmicro`.

Note: All options are case insentitive.

1.2.1 RMW_CONNEXT_LOG_MODE

Option `RMW_CONNEXT_LOG_MODE` controls the logging calls made by `rmw_connextdds`, and `rmw_connextddsmicro`.

Every log call in the code is wrapped by macros, whose definition can be altered to select different logging "backends", and to selectively exclude them from compilation based on the log severity.

default Default logging mode, used if `RMW_CONNEXT_LOG_MODE` is not specified.

Use standard ROS 2 logging facilities provided by `rcutils` as logging backend.

Include log messages at the `error` and `warning` severities.

all Use `rcutils` as logging backend. Include log messages of all severities.

Note that by default, only messages of `info` or higher severity will be printed. Consult the ROS 2 documentation for more information on how to increase logging verbosity.

> **Warning:** This mode should be used with caution, since it will likely cause an increase in CPU usage even if messages are not printed to the console.
>
> Since `rmw_connextdds` and `rmw_connextddsmicro` do not log any messages with `info` severity, it is recommended to use mode `printf` for debugging purposes.

printf Use `printf()` as logging backend. Include all logging messages.

This mode will generate **a lot** of output, but it can be useful to debug issues in the code with little overhead.

1.2.2 RMW_CONNEXT_WAITSET_MODE

Option `RMW_CONNEXT_WAITSET_MODE` can be used to select which of the two available implementations will be used to provision WaitSets, and middleware Events.

By default, `rmw_connextdds` and `rmw_connextddsmicro` will use an implementation based on the standard WaitSet objects included in the DDS API. While this implementation will report the most accurate event notifications and always supports all available features, it may also introduce a non-negligible CPU overhead, which may be undesirable in some scenarios.

For this reason, an alternative more lightweight implementation based on the standard C++ library is also available. This implementation trades off some of the event reporting accuracy, for a much lower runtime overhead.

If this implementation is used, the "event status" structures reported by the RMW will not contain valid "status changes". Applications may only rely on the "absolute" counters contained in these structures.

dds Default implementation based on DDS WaitSets and Conditions.

std Alternative implementation based on C++ standard library.

1.3 Runtime Configuration

In addition to standard configuration facilities provided by the ROS2 RMW interface, `rmw_connextdds`, and `rmw_connextddsmicro` support the additional configuration of some aspects of their runtime behavior via custom environment variables.

1.3.1 RMW_CONNEXT_CYCLONE_COMPATIBILITY_MODE

Enable different policies to improve interoperability with `rmw_cyclonedds_cpp`.

By default, ROS2 applications using `rmw_connextdds` will be able to communicate with those using `rmw_cyclonedds_cpp` only via ROS2 publishers and subscribers, while ROS2 clients and services will not interoperate across vendors.

The reason for this incompatibility lies in `rmw_cyclonedds_cpp`'s use of a custom mapping for propagating request metadata between clients and services.

When this "compatibility mode" is enabled, `rmw_connextdds` (and `rmw_connextddsmicro`) will use this non-standard profile in order to interoperate with `rmw_cyclonedds_cpp`, instead of using one the two standard profiles defined by the DDS-RPC specification (see *RMW_CONNEXT_REQUEST_REPLY_MAPPING*).

1.3.2 RMW_CONNEXT_DISABLE_LARGE_DATA_OPTIMIZATIONS

By default, `rmw_connextdds` will try to detect the use of "large data" types, and automatically optimize the QoS of DDS DataWriters and DataReaders using these types, to improve out of the box performance on reliable streams.

These optimizations will be applied to any endpoint whose type has a serialized size of at least 1MB (configured by a compile-time limit).

`rmw_connextdds` will modify a "large data" endpoint's RTPS reliability protocol parameters to more quickly recover samples, which typically improves performance in the presence of very fragmented data, but it might also end up increasing network traffic unnecessarily, particularly if data is not exchanged at a fast periodic pace.

Variable `RMW_CONNEXT_DISABLE_LARGE_DATA_OPTIMIZATIONS` may be used to disable these automatic optimizations, and revert to Connext's default behavior.

1.3.3 RMW_CONNEXT_DISABLE_FAST_ENDPOINT_DISCOVERY

By default, `rmw_connextdds` modifies the QoS of its DomainParticipant to enable the optimizations defined by RTI Connext DDS' built-in QoS snippet `Optimization.Discovery.Endpoint.Fast`.

These optimizations speed up the discovery process between different applications but they also introduce an overhead in network traffic, which might be undesirable for larger systems.

Variable `RMW_CONNEXT_DISABLE_FAST_ENDPOINT_DISCOVERY` may be used to disable these automatic optimizations, and to leave the DomainParticipant's QoS to its defaults.

1.3.4 RMW_CONNEXT_ENDPOINT_QOS_OVERRIDE_POLICY

When this variable is not set or set to `always`, the QoS settings specified in the default profile will be used and the ros QoS profile will be applied on top of it. You can use topic filters in XML profile files to have different defaults for different topics, but you have to use the mangled topic names (see [ROS topic mangling conventions](#ros-topic-mangling-conventions)).

In case this variable is set to `never`, the QoS settings will be loaded from the default profile as before but the ros QoS profile will be ignored. Be aware of configuring the QoS of rcl topics (`rt/rosout`, `rt/parameter_events`, etc.) and the rmw internal topic `ros_discovery_info` correctly.

This variable can also be set to `dds_topics: <regex>`, e.g.: `dds_topics: rt/my_topic|rt/my_ns/another_topic`. In this case, QoS settings for topics matching the provided regex will be loaded in the same way as the `never` policy, and the ones that don't match will be loaded in the same way as the `always` policy.

ROS topic mangling conventions

ROS mangles topic names in the following way:

- Topics are prefixed with `rt`. e.g.: `/my/fully/qualified/ros/topic` is converted to `rt/my/fully/qualified/ros/topic`.
- The service request topics are prefixed with `rq` and suffixed with `Request`. e.g.: `/my/fully/qualified/ros/service` request topic is `rq/my/fully/qualified/ros/serviceRequest`.
- The service response topics are prefixed with `rr` and suffixed with `Response`. e.g.: `/my/fully/qualified/ros/service` response topic is `rr/my/fully/qualified/ros/serviceResponse`.

1.3.5 RMW_CONNEXT_INITIAL_PEERS

Variable `RMW_CONNEXT_INITIAL_PEERS` can be used to specify a list of comma-separated values of "address locators" that the DomainParticipant created by the RMW will use to try to make contact with remote peer applications during the DDS discovery phase.

The values will be parsed, trimmed, and stored in QoS field `DDS_DomainParticipantQos::discovery::initial_peers`, overwriting any value it previously contained.

While both `rmw_connextdds` and `rmw_connextddsmicro` will honor this variable, equivalent, and more advanced, functionality is already available in RTI Connext DDS, for example using variable `NDDS_DISCOVERY_PEERS`.

For this reason, only users of `rmw_connextddsmicro` should consider specifying `RMW_CONNEXT_INITIAL_PEERS`.

For example, `rmw_connextddsmicro` will use `lo` as its default UDP network interface (see *RMW_CONNEXT_UDP_INTERFACE*), which will prevent it from accessing the default discovery peer (multicast address `239.255.0.1`). The default peer configuration will also prevent the DomainParticipant from carrying out discovery over the built-in shared-memory transport. To enable discovery over this transport, in addition to the default multicast peer:

```
RMW_IMPLEMENTATION=rmw_connextddsmicro \
RMW_CONNEXT_INITIAL_PEERS="_shmem://, 239.255.0.1" \
  ros2 run demo_nodes_cpp listener
```

1.3.6 RMW_CONNEXT_LEGACY_RMW_COMPATIBILITY_MODE

ROS2 applications using `rmw_connextdds` will not be able to interoperate with applications using the previous RMW implementation for RTI Connext DDS, `rmw_connext_cpp`, unless variable `RMW_CONNEXT_LEGACY_RMW_COMPATIBILITY_MODE` is used to enable a "compatibility" mode with these older implementation.

In particular, when this mode is enabled, `rmw_connextdds` will revert to adding a suffix (_) to the end of the names of the attributes of the ROS2 data types propagated via DDS discovery.

1.3.7 RMW_CONNEXT_REQUEST_REPLY_MAPPING

The DDS-RPC specification defines two profiles for mapping "request/reply" interactions over DDS messages (e.g. ROS2 clients and services):

- the *basic* profile conveys information about the originator of a request as an inline payload, serialized before the actual request/reply payloads.
- The *extended* profile relies on DDS' metadata to convey request/reply information out of band.

By default, `rmw_connextdds` uses the *extended* profile when sending requests from a ROS2 client to a service, while `rmw_connextddsmicro` uses the *basic* one.

Variable `RMW_CONNEXT_REQUEST_REPLY_MAPPING` can be used to select the actual profile used at runtime. Either `"basic"` or `"extended"` may be specified.

At the moment, the *extended* profile is only available with `rmw_connextdds`. In this configuration, `rmw_connextdds` will interoperate with `rmw_fastrtps_cpp`, e.g.:

```
RMW_IMPLEMENTATION=rmw_connextdds \
  ros2 run demo_nodes_cpp add_two_ints_server

RMW_IMPLEMENTATION=rmw_fastrtps_cpp \
  ros2 run demo_nodes_cpp add_two_ints_client
```

When using the *basic* profile, `rmw_connextdds` will interoperate with `rmw_connextddsmicro`, e.g.:

```
RMW_IMPLEMENTATION=rmw_connextdds \
RMW_CONNEXT_REQUEST_REPLY_MAPPING=basic \
  ros2 run demo_nodes_cpp add_two_ints_server

RMW_IMPLEMENTATION=rmw_connextddsmicro \
RMW_CONNEXT_INITIAL_PEER=localhost \
  ros2 run demo_nodes_cpp add_two_ints_client
```

Use variable *RMW_CONNEXT_CYCLONE_COMPATIBILITY_MODE* to enable interoperability with `rmw_cyclonedds_cpp` using a non-standard version of the *basic* profile, e.g.:

```
RMW_IMPLEMENTATION=rmw_connextdds \
RMW_CONNEXT_CYCLONE_COMPATIBILITY_MODE=y \
  ros2 run demo_nodes_cpp add_two_ints_server

RMW_IMPLEMENTATION=rmw_cyclonedds_cpp \
  ros2 run demo_nodes_cpp add_two_ints_client
```

1.3.8 RMW_CONNEXT_UDP_INTERFACE

RTI Connext DDS Micro requires applications to explicitly configure the network interface to use for UDPv4 communication.

`rmw_connextddsmicro` makes the arbitrary decision of using *lo* as the default interface.

This is undesireable if non-local communication is required, and/or if the default DDS multicast peer (`239.255.0.1`) is to be used.

Variable `RMW_CONNEXT_UDP_INTERFACE` may be used to customize the network interface actually used by RTI Connext DDS Micro's UDPv4 transport, e.g. to use `eth0`:

```
RMW_IMPLEMENTATION=rmw_connextddsmicro \
RMW_CONNEXT_UDP_INTERFACE=eth0 \
  ros2 run demo_nodes_cpp listener
```

This variable is not used by `rmw_connextdds`.

1.3.9 RMW_CONNEXT_USE_DEFAULT_PUBLISH_MODE

`rmw_connextdds` will always set `DDS_DataWriterQos::publish_mode::kind` of any DataWriter it creates to `DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS`, in order to enable out of the box support for "large data".

This behavior might not be always desirable, and it can be disabled by setting `RMW_CONNEXT_USE_DEFAULT_PUBLISH_MODE` to a non-empty value.

This variable is not used by `rmw_connextddsmicro`, since it doesn't automatically override `DDS_DataWriterQos::publish_mode::kind`.

1.4 Release History

CHAPTER

TWO

DEVELOPER MANUAL

2.1 Coding Style