# rtmdds User's Guide

Geoffrey Biggs

June 28, 2011

## 1 Introduction

rtmdds adds support for the Data Distribution Service middleware standard to OpenRTM-aist. It provides two new types of ports for OpenRTM-aist RT-Components: a publisher port and a subscriber port.

This software is developed at the National Institute of Advanced Industrial Science and Technology. Approval number H23PRO-1274. This software is licensed under the Lesser General Public License. See COPYING and COPYING.LESSER in the source.

## 2 Requirements

rtmdss requires the C++ version of OpenRTM-aist-1.0.0.

rtmdds uses the CMake build system[1]. You will need at least version 2.8 to be able to build the library.

rtmdds has currently only been tested with the DDS implementation provided by RTI[2]. You will need to install RTI DDS before compiling rtmdss. As RTI DDS is a commercial product, you must obtain a valid license from RTI. If you are at a research institute or a university, you may be eligible for a free R&D license. See http://www.rti.com/downloads/dds-research.html for details.

## 3 Installation

### 3.1 Binary

Users of Windows can install the library using the binary installer. This is the recommended method of installation in Windows.

1. Install DDS.
2. Set up your environment for using DDS as described in your implementation's documentation.
   - For RTI DDS, this means adding the ndds/scripts directory to your PATH environment variable (so that rtiddsgen can be found) and setting the NDDSHOME environment variable to the location of the "ndds" directory.
3. Download the rtmdds installer from the website.
4. Double-click the executable file to begin installation.
5. Follow the instructions to install the library.
6. You may need to restart your computer for environment variable changes to take effect before using the component.

[1] http://www.cmake.org/
[2] http://www.rti.com

### 3.2 From source

Follow these steps to install :

1. Install DDS.
2. Set up your environment for using DDS as described in your implementation's documentation.
    - For RTI DDS, this means adding the ndds/scripts directory to your PATH environment variable (so that rtiddsgen can be found) and setting the NDDSHOME environment variable to the location of the "ndds" directory.
3. Download the source, either from the repository or a source archive, and extract it somewhere.

    ```
    tar -xvzf rtmdds.tar.gz
    ```
4. Change to the directory containing the extracted source.

    ```
    cd rtmdds
    ```
5. Create a directory called "build":

    ```
    mkdir build
    ```
6. Change to that directory.

    ```
    cd build
    ```
7. Run cmake or cmake-gui.

    ```
    cmake ../
    ```
8. If no errors occurred, run make.

    ```
    make
    ```
9. Finally, install the library. Ensure the necessary permissions to install into the chosen prefix are available.

    ```
    make install
    ```
10. The install destination can be changed by executing ccmake and changing the variable `CMAKE_INSTALL_PREFIX`.

    ```
    ccmake ../
    ```

The library is now ready for use. See the next section for instructions on using the library.

# 4 Using the library

To use the library in your own components, you must create publisher and subscriber port instances. The steps involved are:

- Create an instance of RTC::DDSPubPort or RTC::DDSSubPort. The template parameters must be the data type to be transported by the port and either the DataWriter or the DataReader for that data type. A DataWriter must be used with a DDSPubPort and a DataReader with a DDSSubPort.
- In the constructor of your component, initialise the port instance with its name and the name of your data type (as a string).

- In the onInitialize() method, register the port with your component using either the addDDSPubPort() or addDDSSubPort() method.
- Register your data type with DDS. The way this is done will vary depending on your DDS implementation. The example components provided with rtmdds use the RTI DDS API.
- In the onExecute() method:
  - For publishing ports, call write(), passing in a reference to an instance of your data type.
  - For subscribing ports, call read(), passing in a pointer to a data type.
- The port instance will be cleaned up automatically by the component when it destructs.

See the included example components for more detail.

# 5 Connecting ports

The DDS ports are not DataPorts. This means that their method of connection is different to DataPorts and ServicePorts. There are two ways to connect a DDS port: dynamic topic and pre-configured topic.

When a DDS port is connected, it will look in the connector profile for the topic to connect to. This is specified in the `ddsport.topic` property of the connector profile.

## 5.1 Pre-configured topic

If the `ddsport.topic` property is found in the connector profile when the DDS port is connected, its value will be used as the name of the topic to connect the port to. This is particularly useful for connecting a port to an existing topic. For example, the `rtcon` command from rtshell can be used to connect a port to an existing topic:

```
$ rtcon pubcomp0.rtc:PublisherPort -p ddsport.topic=my_topic
```

Multiple ports can, of course, be connected to the same topic at the same time:

```
$ rtcon pubcomp0.rtc:PublisherPort subcomp0.rtc:SubscriberPort -p ddsport.topic=a_topic
```

A pre-configured topic can also be set in the configuration properties of the port. If the `ddsport.topic` property is not set in the connector profile, then the `default_topic` property of the port will be checked next. This can be set in the component's configuration file.

## 5.2 Dynamic topic

If there is not a topic already in existance that you wish to connect your ports to, you can have one created for you dynamically. If both the `ddsport.topic` connector profile property and the `default_topic` port property are empty, a new topic will be created by the first port in the connection. This topic will be named after the port's name.

```
$ rtcon pubcomp0.rtc:PublisherPort subcomp0.rtc:SubscriberPort
```

# 6 Properties

The properties that can be set for a port are given in Table 1.

| Property | Type | Default | Effect |
|---|---|---|---|
| verbose | Boolean | NO | Enables or disables debugging output from DDS. |
| qos_file | String | | Specifies the URLs of XML files giving QoS properties for the port. See 7. |
| ignore_user_profile | Boolean | NO | Ignore the user QoS XML file. See the DDS documentation for details. |
| ignore_env_profile | Boolean | NO | Ignore the environment QoS XML file. See the DDS documentation for details. |
| ignore_resource_profile | Boolean | NO | Ignore the resource QoS XML file. See the DDS documentation for details. |
| domain | Integer | 0 | Specifies the domain to operate in. Ports on different domains are invisible to each other. |
| default_topic | String | | Sets the default topic to connect to if the `ddsport.topic` property is not set in the connector profile. |

Table 1: Properties available on DDS ports.

# 7 QoS

DDS supports an extensive array of Quality of Service properties. It is capable of enforcing these properties in the transport at run-time, allowing you to configure how reliable a connection between two ports is without changing the code. These QoS properties are specified in an XML file that is loaded by the DDS implementation. You can specify an XML file for a port to load by setting the `qos_file` property on that port. This must be a set of URLs. Only one file will be loaded. The URLs provide alternative locations for the XML file. If it cannot be found at the first URL, the second will be tried, and so on. Specify the URLs in the following format:

```
[URL1|URL2|URL3|...|URLn]
```

# 8 Examples

A pair of example components, one publisher and one subscriber, are provided. They are installed to `refix$p/share/rtmdds/examples`. Follow these steps to build them (assuming rtmdds was installed in `/usr/local`):

1. `cd $HOME`
2. `mkdir rtmdds_examples`
3. `cd rtmdds_examples`
4. `cmake /usr/local/share/rtmdds/examples`
5. `make`

The example can be executed in conjunction with rtshell for connecting the ports.

1. `rtm-naming`
2. `./rtmdds_pubcomp_standalone`
3. `./rtmdds_subcomp_standalone`
4. `rtcon /localhost/pubcomp0.rtc:PublisherPort /localhost/subcomp0.rtc:SubscriberPort`
5. `rtact /localhost/pubcomp0.rtc`

6. `rtact /localhost/subcomp0.rtc`

You should see data being published by `pubcomp0.rtc` appearing in the terminal running `subcomp0.rtc`.