



VortexDDS

Release 0.1.0

ADLINK

Mar 05, 2018

Contents

1	Install VortexDDS	1
1.1	System requirements	1
1.2	Linux	1
1.2.1	Ubuntu	1
1.2.2	Red Hat	2
1.2.3	Tarball	2
1.2.4	Paths	2
1.3	Windows	2
1.3.1	MSI	2
1.3.2	ZIP	2
1.4	Test your installation	3
1.5	License	4
2	Building VortexDDS applications	5
2.1	Building the <i>Hello World!</i> example	5
2.1.1	Build Files	5
2.1.2	Linux Native Build	5
2.1.3	Windows Native Build	6
2.2	Building With CMake	6
2.2.1	CMake	6
2.2.2	Hello World! CMake (VortexDDS Package)	7
2.2.3	Hello World! Configuration	8
2.2.4	Hello World! Build	8
2.3	Summary	9
3	Hello World! in more detail	11
3.1	Hello World! DataType	11
3.1.1	Data-Centric Architecture	11
3.2	HelloWorldData.idl	11
3.2.1	Hello World! IDL	12
3.2.2	Generate Sources and Headers	12
3.2.3	HelloWorldData.c & HelloWorldData.h	13
3.3	Hello World! Business Logic	13
3.3.1	<i>Hello World!</i> Subscriber Source Code	13
3.3.2	<i>Hello World!</i> Publisher Source Code	16
4	What's next?	19

4.1	The OMG DDS Specification	19
4.2	AdLink Documentation and Tutorials	19
4.3	AdLink on Youtube and Slideshare	19
4.4	AdLink on Social Media	20
4.5	The DDS community	20
4.6	Support	20
5	Uninstalling VortexDDS	21
5.1	Linux	21
5.2	Windows	21
5.2.1	Original MSI	21
5.2.2	Apps & features	22
6	Vortex DDS C API Reference	23
7	Indices and tables	157

1.1 System requirements

Currently AdLink VortexDDS is supported on the following platforms:

Operating systems	Architecture	Compiler
Ubuntu 16.04 LTS	64-bit	gcc 5.4 or later
Windows 10	64-bit	VS2015

1.2 Linux

1.2.1 Ubuntu

On Ubuntu and other debian-derived platforms, the product can be installed using a native package.

```
sudo dpkg -i vortex-dds_<version>_<architecture>.deb  
sudo dpkg -i vortex-dds-dev_<version>_<architecture>.deb
```

Post install steps

The installation package installs examples in system directories. In order to have a better user experience when building the VortexDDS examples, it is advised to copy the examples to a user-defined location. This is to be able to build the examples natively and experiment with the example source code.

For this, the installation package provides the `vdds_install_examples` script, located in `/usr/bin`.

Create an user writable directory where the examples should go. Navigate to that directory and execute the script. Answer 'yes' to the questions and the examples will be installed in the current location.

Type `vdds_install_examples -h` for more information.

1.2.2 Red Hat

Not supported yet (CHAM-326).

1.2.3 Tarball

For more generic Linux installations, different tar-balls (with the same content) are provided.

Tarball	Description
VortexDDS-<version>-Linux.tar.Z	Tar Compress compression.
VortexDDS-<version>-Linux.tar.gz	Tar GZip compression.
VortexDDS-<version>-Linux.tar.sh	Self extracting Tar GZip compression.

By extracting one of them at any preferred location, VortexDDS can be used.

1.2.4 Paths

To be able to run VortexDDS executables, the required libraries (like libddsc.so) need to be available to the executables. Normally, these are installed in system default locations and it works out-of-the-box. However, if they are not installed in those locations, it is possible that the library search path has to be changed. This can be achieved by executing the command:

```
export LD_LIBRARY_PATH=<install_dir>/lib:$LD_LIBRARY_PATH
```

1.3 Windows

1.3.1 MSI

The default deployment method on Windows is to install the product using the MSI installer.

The installation process is self-explanatory. Three components are available:

1. a runtime component, containing the runtime libraries
2. a development component, containing the header files, the IDL compiler, a precompiled Hello Word! example and other examples.
3. an examples component, containing the source code of the VortexDDS examples.

The runtime and development components are (by default) installed in “Program Files” while the VortexDDS example component will be installed in the User Profile directory. The VortexDDS example code in the User Profile directory can be changed by the user.

1.3.2 ZIP

The Windows installation is also provided as a ZIP file. By extracting it at any preferred location, VortexDDS can be used.

Paths

To be able to run VortexDDS executables, the required libraries (like `ddsc.dll`) need to be available to the executables. Normally, these are installed in system default locations and it works out-of-the-box. However, if they are not installed on those locations, it is possible that the library search path has to be changed. This can be achieved by executing the command:

```
set PATH=<install_dir>/bin;%PATH%
```

Note: The MSI installer will add this path to the PATH environment variable automatically.

1.4 Test your installation

The installation provides a simple prebuilt *Hello World!* application which can be run in order to test your installation. The *Hello World!* application consists of two executables: a so called `HelloworldPublisher` and a `HelloworldSubscriber`, typically located in `/usr/share/VortexDDS/examples/helloworld/bin` on Linux and in `C:\Program Files\ADLINK\Vortex DDS\share\VortexDDS\examples\helloworld\bin` on Windows.

To run the example application, please open two console windows and navigate to the appropriate directory in both console windows. Run the `HelloworldSubscriber` in one of the console windows by the typing following command:

Windows `HelloworldSubscriber.exe`

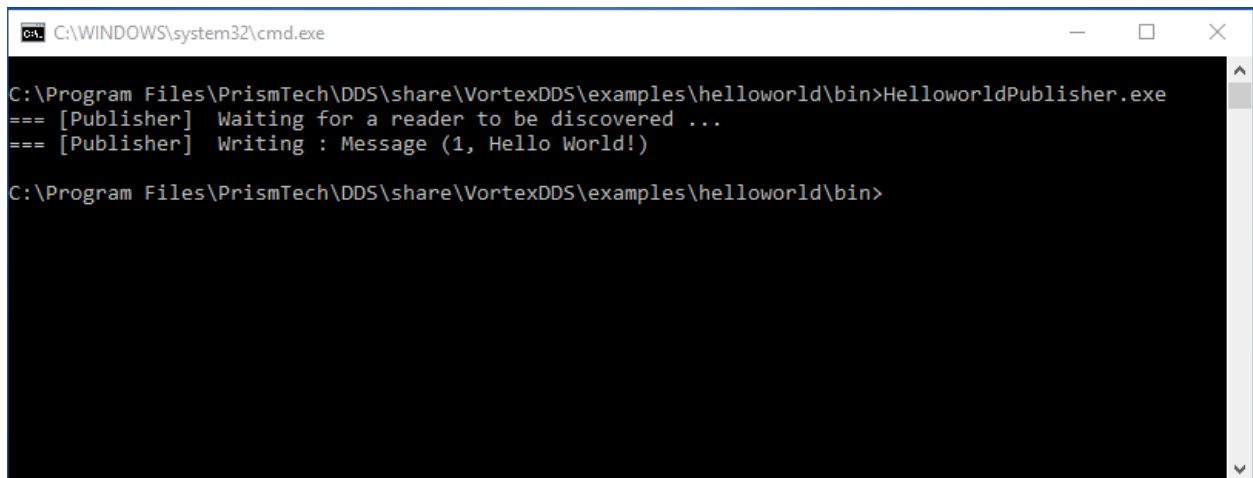
Linux `./HelloworldSubscriber`

and the `HelloworldPublisher` in the other console window by typing:

Windows `HelloworldPublisher.exe`

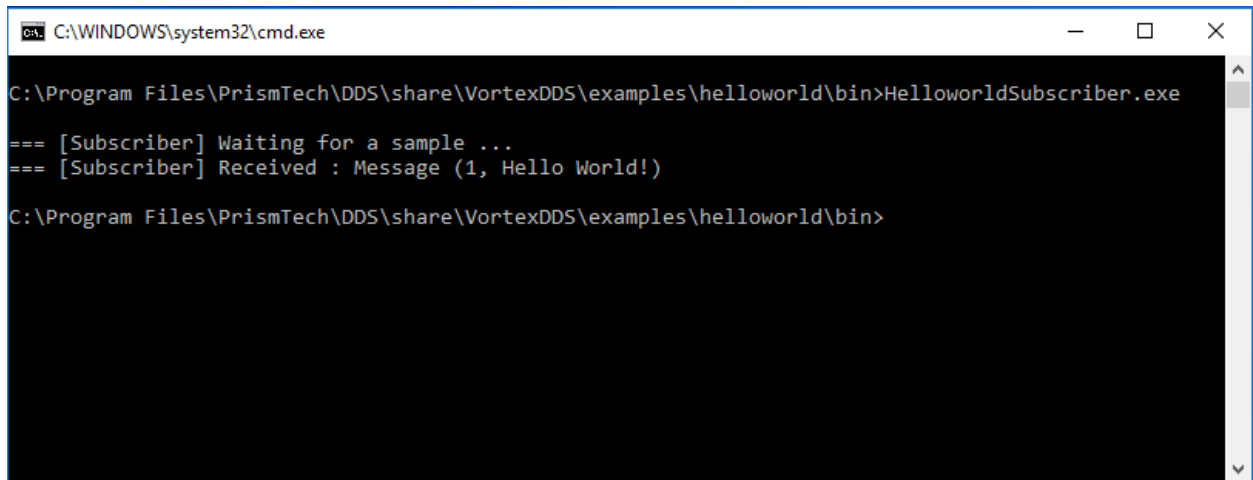
Linux `./HelloworldPublisher`

The output `HelloworldPublisher` should look like

A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The prompt is at "C:\Program Files\PrismTech\DDS\share\VortexDDS\examples\helloworld\bin>". The user has entered "HelloworldPublisher.exe". The output shows "=== [Publisher] Waiting for a reader to be discovered ..." followed by "=== [Publisher] Writing : Message (1, Hello World!)". The prompt is now "C:\Program Files\PrismTech\DDS\share\VortexDDS\examples\helloworld\bin>".

```
C:\WINDOWS\system32\cmd.exe
C:\Program Files\PrismTech\DDS\share\VortexDDS\examples\helloworld\bin>HelloworldPublisher.exe
=== [Publisher] Waiting for a reader to be discovered ...
=== [Publisher] Writing : Message (1, Hello World!)
C:\Program Files\PrismTech\DDS\share\VortexDDS\examples\helloworld\bin>
```

while the `HelloworldSubscriber` will be looking like this

A screenshot of a Windows command prompt window. The title bar shows 'C:\WINDOWS\system32\cmd.exe'. The command prompt shows the following text:

```
C:\Program Files\PrismTech\DDS\share\VortexDDS\examples\helloworld\bin>HelloworldSubscriber.exe  
=== [Subscriber] Waiting for a sample ...  
=== [Subscriber] Received : Message (1, Hello World!)  
C:\Program Files\PrismTech\DDS\share\VortexDDS\examples\helloworld\bin>
```

For more information on how to build this application your own and the code which has been used, please have a look at the [Hello World!](#) chapter.

1.5 License

TODO: CHAM-325

Building VortexDDS applications

2.1 Building the *Hello World!* example

To test the *installation*, a small *Hello World!* application is used. This application will also be used as an introduction to DDS.

This chapter explains how to build this example, without details regarding the source code. The next chapter will explain what has to be done to code the *Hello World!* example.

The procedure used to build the *Hello World!* example can also be used for building your own applications.

Windows It is advised to have the VortexDDS examples component installed (see *Windows installation*) when actively building the VortexDDS examples on Windows. This chapter refers to the VortexDDS examples installed in the User Profile directory on Windows.

Linux It is advised to have copied the VortexDDS examples to a user-friendly location as described in *this* paragraph when actively building the VortexDDS examples on Linux. This chapter refers to the VortexDDS examples installed in the user-defined location.

2.1.1 Build Files

Three files are available *Hello World!* root directory to support building the example. Both *Windows native* (HelloWorld.sln) and *Linux native* (Makefile) build files will only be available for this *Hello World!* example. All the other examples make use of the *CMake* build system and thus only have the CMakeLists.txt build related file.

2.1.2 Linux Native Build

A Linux native Makefile is provided in the `examples/helloworld` directory within the destination location entered in the *vdds_install_examples script*. In a terminal, go to that directory and type

```
make
```

The build process should have access to the include files and the ddsc library. The Makefile expects them to be present at system default locations so that it can find them automatically. If this isn't the case on your machine, then please update the commented out CFLAGS and LDFLAGS within the Makefile to point to the proper locations.

This will build the HelloWorldSubscriber and HelloWorldPublisher executables in the helloworld source directory (not the bin directory that contains the pre-build binaries).

The *Hello World!* example can now be executed, like described in [Test your installation](#), using the binaries that were just build. Be sure to use the right directories.

2.1.3 Windows Native Build

For the Windows Native Build, a Visual Studio solution file is available in the `examples/helloworld` directory. Use a file explorer to navigate to that directory and double click on the `HelloWorld.sln` file. Visual Studio should now start with the HelloWorld solution that contains three projects.

Project	Description
HelloWorldPublisher	Information to build the example publisher.
HelloWorldSubscriber	Information to build the example subscriber.
HelloWorldType	Information to (re)generate <i>HelloWorldData_Msg</i> data type.

Creating the *Hello World!* example executables is as simple as selecting the required configuration and building the solution.

`helloworld\vs\directories.props` contains the location of where the VortexDDS header files and libraries are be placed. These locations are based on the default installation directory structure. When VortexDDS is installed in a different directory, the following paths in `helloworld\vs\directories.props` should be changed, like:

```
<VortexDDS_lib_dir>C:/Path/To/VortexDDS/Installation/lib</VortexDDS_lib_dir>
<VortexDDS_inc_dir>C:/Path/To/VortexDDS/Installation/include</VortexDDS_inc_dir>
<VortexDDS_idlc_dir>C:/Path/To/VortexDDS/Installation/share/VortexDDS/idlc</VortexDDS_
↪ idlc_dir>
```

To run the example, Visual Studio should run both the publisher and subscriber simultaneously. It is capable of doing so, but it's not its default setting. To change it, open the HelloWorld solution property page by right clicking the solution and selecting Properties. Then go to Common Properties -> Startup Project, select Multiple startup project and set Action "Start" for HelloWorldPublisher and HelloWorldSubscriber. Finish the change by selecting OK.

Visual Studio is now ready to actually run the *Hello World!* example, which can be done by selecting Debug -> Start without debugging. Both the HelloWorldSubscriber and the HelloWorldPublisher will be started and the HelloWorldPublisher will write a message that is received by the HelloWorldSubscriber.

2.2 Building With CMake

In the earlier chapters, building the *Hello World!* example is done natively. However, the *Hello World!* example can also be build using the [CMake tool](#). This is what is recommended. In fact, all the other examples don't provide native makefiles, only CMake files.

2.2.1 CMake

CMake is an open-source, cross-platform family of tools designed to build, test and package software. CMake is used to control the software compilation process using simple platform and compiler independent configuration files, and

generate native makefiles and workspaces that can be used in the compiler environment of your choice.

In other words, CMake's main strength is build portability. CMake uses the native tools, and other than requiring itself, does not require any additional tools to be installed. The same CMake input files will build with GNU make, Visual studio 6,7,8 IDEs, borland make, nmake, and XCode.

An other advantage of CMake is building out-of-source. It simply works out-of-the-box. There are two important reasons to choose this:

1. Easy cleanup (no cluttering the source tree). Simply remove the build directory if you want to start from scratch.
2. Multiple build targets. It's possible to have up-to-date Debug and Release targets, without having to recompile the entire tree. For systems that do cross-platform compilation, it is easy to have up-to-date builds for the host and target platform.

There are a few other benefits to CMake, but that is out of the scope of this document.

2.2.2 Hello World! CMake (VortexDDS Package)

After the CMake digression, we're back with the *Hello World!* example. Apart from the native build files, CMake build files are provided as well. See `examples/helloworld/CMakeLists.txt`

```

1 cmake_minimum_required(VERSION 3.5)
2
3 if (NOT TARGET VortexDDS::ddsc)
4     # Find the VortexDDS package. If it is not in a default location, try
5     # finding it relative to the example where it most likely resides.
6     find_package(VortexDDS REQUIRED PATHS "${CMAKE_SOURCE_DIR}/../..")
7 endif()
8
9 # This is a convenience function, provided by the VortexDDS package,
10 # that will supply a library target related the the given idl file.
11 # In short, it takes the idl file, generates the source files with
12 # the proper data types and compiles them into a library.
13 idlc_generate(HelloWorldData_lib "HelloWorldData.idl")
14
15 # Both executables have only one related source file.
16 add_executable(HelloworldPublisher publisher.c)
17 add_executable(HelloworldSubscriber subscriber.c)
18
19 # Both executables need to be linked to the idl data type library and
20 # the ddsc API library.
21 target_link_libraries(HelloworldPublisher HelloWorldData_lib VortexDDS::ddsc)
22 target_link_libraries(HelloworldSubscriber HelloWorldData_lib VortexDDS::ddsc)

```

It will try to find the VortexDDS CMake package. When it has found it, every path and dependencies are automatically set. After that, an application can use it without fuss. CMake will look in the default locations for the code: *VortexDDS* package. The *VortexDDS* package provides the `ddsc` library that contains the DDS API that the application needs. But apart from that, it also contains helper functionality (`idlc_generate`) to generate library targets from IDL files. These library targets can be easily used when compiling an application that depends on a data type described in an IDL file.

Two applications will be created, `HelloworldPublisher` and `HelloworldSubscriber`. Both consist only out of one source file.

Both applications need to be linked to the `ddsc` library in the *VortexDDS* package and `HelloWorldData_lib` that was generated by the call to `idlc_generate`.

2.2.3 Hello World! Configuration

The *Hello World!* example is prepared to be built by CMake through the use of its `CMakeLists.txt` file. The first step is letting CMake configure the build environment.

It's good practice to build examples or applications out-of-source. In order to do that, create a `build` directory in the `examples/helloworld` directory and go there, making our location `examples/helloworld/build`.

Here, we can let CMake configure the build environment for us by typing:

```
cmake ../
```

Note: CMake does a pretty good job at guessing which generator to use, but some environments require that you supply a specific generator. For example, only 64-bit libraries are shipped for Windows, but CMake will generate a 32-bit project by default, resulting in linker errors. When generating a Visual Studio project keep in mind to append **Win64** to the generator. The example below shows how to generate a Visual Studio 2015 project.

```
cmake -G "Visual Studio 14 2015 Win64" ..
```

Note: CMake generators can also create IDE environments. For instance, the “Visual Studio 14 2015 Win64” will generate a Visual Studio solution file. Other IDE's are also possible, like Eclipse.

CMake will use the `CMakeLists.txt` in the `helloworld` directory to create makefiles that fit the native platform.

Since everything is prepared, we can actually build the applications (`HelloWorldPublisher` and `HelloWorldSubscriber` in this case).

2.2.4 Hello World! Build

After the configuration step, building the example is as easy as typing:

```
cmake --build .
```

Note: On Windows, it is likely that you have to supply the config of Visual Studio:

```
cmake --build . --config "Release"
```

while being in the build directory created during the configuration step: `examples/helloworld/build`.

The resulting Publisher and Subscriber applications can be found in:

Windows `examples\helloworld\build\Release`.

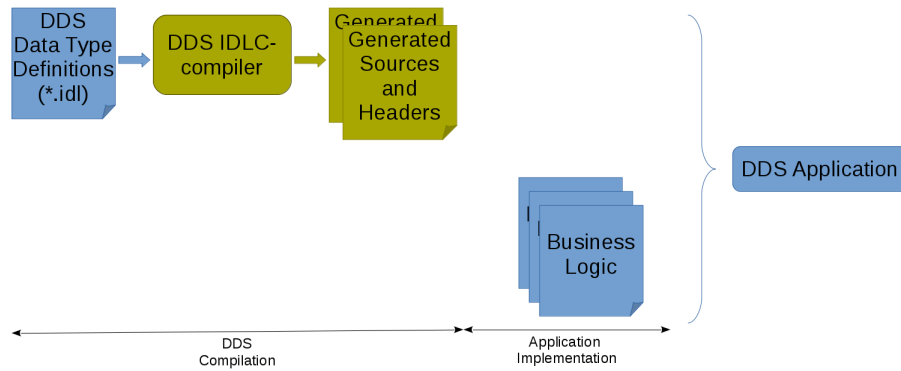
Linux `examples/helloworld/build`.

The *Hello World!* example can now be executed, like described in [Test your installation](#), using the binaries that were just build. Be sure to use the right directories.

2.3 Summary

We've seen that a VortexDDS application can be build by using a Makefile on Linux or a Visual Studio Solutions on Windows. Also CMake can be used to build a VortexDDS application. In fact, it is the preferred way of building.

In the end, a predefined way of generating and building the source code should be followed when building VortexDDS applications. The figure below shows how a typical VortexDDS application is build.



Next chapter will provide an overview of all steps mentioned in the figure above.

Hello World! in more detail

The previous chapter focused on building the *Hello World!* example while this chapter will focus on the code itself; what has to be done to code this small example.

3.1 Hello World! DataType

3.1.1 Data-Centric Architecture

By creating a Data-centric architecture, you get a loosely coupled information-driven system. It emphasizes a data layer that is common for all distributed applications within the system. Because there is no direct coupling among the applications in the DDS model, they can be added and removed easily in a modular and scalable manner. This makes that the complexity of a data-centric architecture doesn't really increase when more and more publishers/subscribers are added.

The *Hello World!* example has a very simple 'data layer' of only one data type `HelloWorldData_Msg` (please read on). The subscriber and publisher are not aware of each other. The former just waits until somebody provides the data it requires, while the latter just publishes the data without considering the number of interested parties. In other words, it doesn't matter for the publisher if there are none or multiple subscribers (try running the *Hello World!* example by starting multiple `HelloWorldSubscribers` before starting a `HelloWorldPublisher`). A publisher just writes the data. The DDS middleware takes care of delivering the data when needed.

3.2 HelloWorldData.idl

To be able to send data from a writer to a reader, DDS needs to know the data type. For the *Hello World!* example, this data type is described using **IDL** and is located in `HelloWorldData.idl`. This IDL file will be compiled by a IDL compiler which in turn generates a C language source and header file. These generated source and header file will be used by the `HelloWorldSubscriber` and `HelloWorldPublisher` in order to communicate the *Hello World!* message between the `HelloWorldPublisher` and the `HelloWorldSubscriber`.

3.2.1 Hello World! IDL

There are a few ways to describe the structures that make up the data layer. The HelloWorld uses the IDL language to describe the data type in HelloWorldData.idl:

```
1 module HelloWorldData
2 {
3     struct Msg
4     {
5         long userID;
6         string message;
7     };
8     #pragma keylist Msg userID
9 };
```

An extensive explanation of IDL lies outside the scope of this example. Nevertheless, a quick overview of this example is given anyway.

First, there's the module `HelloWorldData`. This is a kind of namespace or scope or similar. Within that module, there's the struct `Msg`. This is the actual data structure that is used for the communication. In this case, it contains a `userID` and `message`.

The combination of this module and struct translates to the following when using the c language.

```
typedef struct HelloWorldData_Msg
{
    int32_t userID;
    char * message;
} HelloWorldData_Msg;
```

When it is translated to a different language, it will look different and more tailored towards that language. This is the advantage of using a data oriented language, like IDL, to describe the data layer. It can be translated into different languages after which the resulting applications can communicate without concerns about the (possible different) programming languages these application are written in.

3.2.2 Generate Sources and Headers

Like already mentioned in the *Hello World! IDL* chapter, an IDL file contains the description of data type(s). This needs to be translated into programming languages to be useful in the creation of DDS applications.

To be able to do that, there's a pre-compile step that actually compiles the IDL file into the desired programming language.

A java application `com.prismtech.vortex.compilers.Idlc` is supplied to support this pre-compile step. This is available in `idlc-jar-with-dependencies.jar`

The compilation from IDL into c source code is as simple as starting that java application with an IDL file. In the case of the *Hello World!* example, that IDL file is `HelloWorldData.idl`.

```
java -classpath "<install_dir>/share/VortexDDS/idlc/idlc-jar-with-dependencies.jar"
↪com.prismtech.vortex.compilers.Idlc HelloWorldData.idl
```

Windows The `HelloWorldType` project within the HelloWorld solution.

Linux The `make datatype` command.

This will result in new generated `HelloWorldData.c` and `generated/HelloWorldData.h` files that can be used in the *Hello World!* publisher and subscriber applications.

The application has to be rebuilt when the data type source files were re-generated.

Again, this is all for the native builds. When using CMake, all this is done automatically.

3.2.3 HelloWorldData.c & HelloWorldData.h

As described in the *Hello World! DataType* paragraph, the IDL compiler will generate this source and header file. These files contain the data type of the messages that are sent and received.

While the c source has no interest for the application developers, HelloWorldData.h contains some information that they depend on. For example, it contains the actual message structure that is used when writing or reading data.

```
typedef struct HelloWorldData_Msg
{
    int32_t userID;
    char * message;
} HelloWorldData_Msg;
```

It also contains convenience macros to allocate and free memory space for the specific data types.

```
HelloWorldData_Msg__alloc()
HelloWorldData_Msg_free(d,o)
```

It contains an extern variable that describes the data type to the DDS middleware as well.

```
HelloWorldData_Msg_desc
```

3.3 Hello World! Business Logic

Apart from the *HelloWorldData data type files* that the *Hello World!* example uses to send messages, the *Hello World!* example also contains two (user) source files (*subscriber.c* and *publisher.c*), containing the business logic.

3.3.1 Hello World! Subscriber Source Code

Subscriber.c contains the source that will wait for a *Hello World!* message and reads it when it receives one.

```
1  #include "ddsc/dds.h"
2  #include "HelloWorldData.h"
3  #include <stdio.h>
4  #include <string.h>
5  #include <stdlib.h>
6
7  /* An array of one message (aka sample in dds terms) will be used. */
8  #define MAX_SAMPLES 1
9
10 int main (int argc, char ** argv)
11 {
12     dds_entity_t participant;
13     dds_entity_t topic;
14     dds_entity_t reader;
15     HelloWorldData_Msg *msg;
16     void *samples[MAX_SAMPLES];
17     dds_sample_info_t infos[MAX_SAMPLES];
18     dds_return_t ret;
```

```

19     dds_qos_t *qos;
20
21     /* Create a Participant. */
22     participant = dds_create_participant (DDS_DOMAIN_DEFAULT, NULL, NULL);
23     DDS_ERR_CHECK (participant, DDS_CHECK_REPORT | DDS_CHECK_EXIT);
24
25     /* Create a Topic. */
26     topic = dds_create_topic (participant, &HelloWorldData_Msg_desc,
27                             "HelloWorldData_Msg", NULL, NULL);
28     DDS_ERR_CHECK (topic, DDS_CHECK_REPORT | DDS_CHECK_EXIT);
29
30     /* Create a reliable Reader. */
31     qos = dds_qos_create ();
32     dds_qoset_reliability (qos, DDS_RELIABILITY_RELIABLE, DDS_SECS (10));
33     reader = dds_create_reader (participant, topic, qos, NULL);
34     DDS_ERR_CHECK (reader, DDS_CHECK_REPORT | DDS_CHECK_EXIT);
35     dds_qos_delete (qos);
36
37     printf ("\n=== [Subscriber] Waiting for a sample ...\n");
38
39     /* Initialize sample buffer, by pointing the void pointer within
40      * the buffer array to a valid sample memory location. */
41     samples[0] = HelloWorldData_Msg__alloc ();
42
43     /* Poll until data has been read. */
44     while (true)
45     {
46         /* Do the actual read.
47          * The return value contains the number of read samples. */
48         ret = dds_read (reader, samples, infos, MAX_SAMPLES, MAX_SAMPLES);
49         DDS_ERR_CHECK (ret, DDS_CHECK_REPORT | DDS_CHECK_EXIT);
50
51         /* Check if we read some data and it is valid. */
52         if ((ret > 0) && (infos[0].valid_data))
53         {
54             /* Print Message. */
55             msg = (HelloWorldData_Msg*) samples[0];
56             printf ("=== [Subscriber] Received : ");
57             printf ("Message (%d, %s)\n", msg->userID, msg->message);
58             break;
59         }
60         else
61         {
62             /* Polling sleep. */
63             dds_sleepfor (DDS_MSECS (20));
64         }
65     }
66
67     /* Free the data location. */
68     HelloWorldData_Msg_free (samples[0], DDS_FREE_ALL);
69
70     /* Deleting the participant will delete all its children recursively as well. */
71     ret = dds_delete (participant);
72     DDS_ERR_CHECK (ret, DDS_CHECK_REPORT | DDS_CHECK_EXIT);
73
74     return EXIT_SUCCESS;
75 }

```

We will be using the DDS API and the *HelloWorldData_Msg* type to receive data. For that, we need to include the appropriate header files.

```
#include "ddsc/dds.h"
#include "HelloWorldData.h"
```

The main starts with defining a few variables that will be used for reading the *Hello World!* message. The entities are needed to create a reader.

```
dds_entity_t participant;
dds_entity_t topic;
dds_entity_t reader;
```

Then there are some buffers that are needed to actually read the data.

```
HelloWorldData_Msg *msg;
void *samples[MAX_SAMPLES];
dds_sample_info_t info[MAX_SAMPLES];
```

To be able to create a reader, we first need a participant. This participant is part of a specific communication domain. In the *Hello World!* example case, it is part of the default domain.

```
participant = dds_create_participant (DDS_DOMAIN_DEFAULT, NULL, NULL);
```

The another requisite is the topic which basically describes the data type that is used by the reader. When creating the topic, the *data description* for the DDS middleware that is present in the *HelloWorldData.h* is used. The topic also has a name. Topics with the same data type description, but with different names, are considered different topics. This means that readers/writers created with a topic named “A” will not interfere with readers/writers created with a topic named “B”.

```
topic = dds_create_topic (participant, &HelloWorldData_Msg_desc,
                        "HelloWorldData_Msg", NULL, NULL);
```

When we have a participant and a topic, we then can create the reader. Since the order in which the *Hello World!* Publisher and *Hello World!* Subscriber are started shouldn’t matter, we need to create a so called ‘reliable’ reader. Without going into details, the reader will be created like this

```
dds_qos_t *qos = dds_qos_create ();
dds_qos_set_reliability (qos, DDS_RELIABILITY_RELIABLE, DDS_SECS (10));
reader = dds_create_reader (participant, topic, qos, NULL);
dds_qos_delete(qos);
```

We are almost able to read data. However, the read expects an array of pointers to valid memory locations. This means the samples array needs initialization. In this example, we have an array of only one element: `#define MAX_SAMPLES 1`. So, we only need to initialize one element.

```
samples[0] = HelloWorldData_Msg__alloc ();
```

Now everything is ready for reading data. But we don’t know if there is any data. To simplify things, we enter a polling loop that will exit when data has been read.

Within the polling loop, we do the actual read. We provide the initialized array of pointers (*samples*), an array that holds information about the read sample(s) (*info*), the size of the arrays and the maximum number of samples to read. Every read sample in the samples array has related information in the info array at the same index.

```
ret = dds_read (reader, samples, info, MAX_SAMPLES, MAX_SAMPLES);
```

The `dds_read` function returns the number of samples it actually read. We can use that to determine if the function actually read some data. When it has, then it is still possible that the data part of the sample is not valid. This has some use cases when there is no real data, but still the state of the related sample has changed (for instance it was deleted). This will normally not happen in the *Hello World!* example. But we check for it anyway.

```
if ((ret > 0) && (info[0].valid_data))
```

If data has been read, then we can cast the void pointer to the actual message data type and display the contents. The polling loop is quit as well in this case.

```
msg = (HelloWorldData_Msg*) samples[0];
printf ("=== [Subscriber] Received : ");
printf ("Message (%d, %s)\n", msg->userID, msg->message);
break;
```

When data is received and the polling loop is stopped, we need to clean up.

```
HelloWorldData_Msg_free (samples[0], DDS_FREE_ALL);
dds_delete (participant);
```

All the entities that are created using the participant are also deleted. This means that deleting the participant will automatically delete the topic and reader as well.

3.3.2 *Hello World!* Publisher Source Code

Publisher.c contains the source that will write an *Hello World!* message on which the subscriber is waiting.

```
1  #include "ddsc/dds.h"
2  #include "HelloWorldData.h"
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main (int argc, char ** argv)
7  {
8      dds_entity_t participant;
9      dds_entity_t topic;
10     dds_entity_t writer;
11     dds_return_t ret;
12     HelloWorldData_Msg msg;
13
14     /* Create a Participant. */
15     participant = dds_create_participant (DDS_DOMAIN_DEFAULT, NULL, NULL);
16     DDS_ERR_CHECK (participant, DDS_CHECK_REPORT | DDS_CHECK_EXIT);
17
18     /* Create a Topic. */
19     topic = dds_create_topic (participant, &HelloWorldData_Msg_desc,
20                             "HelloWorldData_Msg", NULL, NULL);
21     DDS_ERR_CHECK (topic, DDS_CHECK_REPORT | DDS_CHECK_EXIT);
22
23     /* Create a Writer. */
24     writer = dds_create_writer (participant, topic, NULL, NULL);
25
26     printf("=== [Publisher] Waiting for a reader to be discovered ...\n");
27
28     ret = dds_set_enabled_status(writer, DDS_PUBLICATION_MATCHED_STATUS);
29     DDS_ERR_CHECK (ret, DDS_CHECK_REPORT | DDS_CHECK_EXIT);
30
```

```

31 while(true)
32 {
33     uint32_t status;
34     ret = dds_get_status_changes (writer, &status);
35     DDS_ERR_CHECK (ret, DDS_CHECK_REPORT | DDS_CHECK_EXIT);
36
37     if (status == DDS_PUBLICATION_MATCHED_STATUS) {
38         break;
39     }
40     /* Polling sleep. */
41     dds_sleepfor (DDS_MSECS (20));
42 }
43
44 /* Create a message to write. */
45 msg.userID = 1;
46 msg.message = "Hello World";
47
48 printf ("=== [Publisher] Writing : ");
49 printf ("Message (%d, %s)\n", msg.userID, msg.message);
50
51 ret = dds_write (writer, &msg);
52 DDS_ERR_CHECK (ret, DDS_CHECK_REPORT | DDS_CHECK_EXIT);
53
54 /* Deleting the participant will delete all its children recursively as well. */
55 ret = dds_delete (participant);
56 DDS_ERR_CHECK (ret, DDS_CHECK_REPORT | DDS_CHECK_EXIT);
57
58 return EXIT_SUCCESS;
59 }

```

We will be using the DDS API and the *HelloWorldData_Msg* type to sent data. For that, we need to include the appropriate header files.

```

#include "ddsc/dds.h"
#include "HelloWorldData.h"

```

Just like with the *reader in subscriber.c*, we need a participant and a topic to be able to create a writer. We use the same topic name as in *subscriber.c*. Otherwise the reader and writer are not considered related and data will not be sent between them.

```

dds_entity_t participant;
dds_entity_t topic;
dds_entity_t writer;

participant = dds_create_participant (DDS_DOMAIN_DEFAULT, NULL, NULL);
topic = dds_create_topic (participant, &HelloWorldData_Msg_desc,
                        "HelloWorldData_Msg", NULL, NULL);
writer = dds_create_writer (participant, topic, NULL, NULL);

```

The DDS middleware is a publication/subscription implementation. This means that it will discover related readers and writers (i.e. readers and writers sharing the same data type and topic name) and connect them so that written data can be received by readers without the application having to worry about it. There is a catch though: this discovery and coupling takes a small amount of time. There are various ways to work around this problem. The following can be done to properly connect readers and writers:

- Wait for the publication/subscription matched events
 - The Subscriber should wait for a subscription matched event

- The Publisher should wait for a publication matched event.

The use of these events will be outside the scope of this example

- Poll for the publication/subscription matches statuses
 - The Subscriber should poll for a subscription matched status to be set
 - The Publisher should poll for a publication matched status to be set

The Publisher in this example uses the polling schema.

- Let the publisher sleep for a second before writing a sample. This is not recommended since a second may not be enough on several networks
- Accept that the reader miss a few samples at startup. This may be acceptable in cases where the publishing rate is high enough.

As said, the publisher of this example polls for the publication matched status. To make this happen, the writer must be instructed to ‘listen’ for this status. The following line of code makes sure the writer does so.

```
dds_set_enabled_status(writer, DDS_PUBLICATION_MATCHED_STATUS);
```

Now the polling may start:

```
while(true)
{
    uint32_t status;
    ret = dds_get_status_changes (writer, &status);
    DDS_ERR_CHECK(ret, DDS_CHECK_REPORT | DDS_CHECK_EXIT);

    if (status == DDS_PUBLICATION_MATCHED_STATUS) {
        break;
    }
    /* Polling sleep. */
    dds_sleepfor (DDS_MSECS (20));
}
```

After this loop, we are sure that a matching reader has been started. Now, we commence to writing the data. First the data must be initialized

```
HelloWorldData_Msg msg;

msg.userID = 1;
msg.message = "Hello World";
```

Then we can actually sent the message to be received by the subscriber.

```
ret = dds_write (writer, &msg);
```

After the sample is written, we need to clean up.

```
ret = dds_delete (participant);
```

All the entities that are created using the participant are also deleted. This means that deleting the participant will automatically delete the topic and writer as well.

CHAPTER 4

What's next?

Want to know more about VortexDDS? Please consider following a tutorial or visit some of the pages listed below. The Vortex DDS Launcher (provided with this installation) is also a good starting point.

Windows The Vortex DDS Launcher can be started from within the Windows Start Menu

Linux Type 'vortexddslaucher' in a console window

4.1 The OMG DDS Specification

PrismTech (aquired by AdLink) has been an active member of the Object Management Group® (OMG®) for over several years and is heavily involved in the development of the DDS specification. Please visit the OMG website at <http://www.omg.org> and specifically the [DDS Getting Started](#) page and the [DDS specification](#) itself.

4.2 AdLink Documentation and Tutorials

- [Documentation](#)
- [DDS Tutorial](#)

4.3 AdLink on Youtube and Slideshare

AdLink is also active on Youtube and Slideshare. Please following the links below to view some interesting videos and presentations.

- [Overview](#)
- [Vortex Youtube](#)
- [Vortex Slideshare](#)
- [Vortex Demo](#)

4.4 AdLink on Social Media

- Twitter (@ADLINKTech_usa)
- Facebook
- LinkedIn

4.5 The DDS community

- The AdLink DDS-community
- The AdLink DDS Forum

4.6 Support

- Knowledge base
- Support (registered users)

Uninstalling VortexDDS

5.1 Linux

Uninstalling VortexDDS on Linux can be established by invoking the following two commands (of which the first is optional):

```
sudo dpkg --remove vortex-dds-dev
sudo dpkg --remove vortex-dds
```

Note: Mind the order in which these commands are run. The development package (`vortex-dds-dev`) need to be removed first since it depends on the library version (`vortex-dds`).

5.2 Windows

There are two ways to uninstall VortexDDS from Windows

1. By using the original VortexDDS *MSI* file
2. By using Windows “Apps & features”

5.2.1 Original MSI

Locate the original VortexDDS MSI file on your system and start it. After clicking *Next*, an overview of options appears, amongst which is the remove option. By clicking *Remove*, all files and folders are removed, except the VortexDDS examples (if installed).

5.2.2 Apps & features

Go to Windows Settings by clicking the Settings-icon (⚙️) in the Windows Start Menu. Choose Apps in the Windows Settings screen. A list of all installed apps and programs pops up. Select VortexDDS and choose Uninstall. All installed files and folders will be removed, except the VortexDDS examples (if installed).

struct dds_aligned_allocator

Public Members

void **(*alloc)** (size_t size, size_t align)

void **(*free)** (size_t size, void *ptr)

struct dds_allocator

Public Members

void **(*malloc)** (size_t size)

void **(*realloc)** (void *ptr, size_t size)

void **(*free)** (void *ptr)

struct dds_history_qospolicy

#include <dds_public_qos.h> History QoS: Applies to Topic, DataReader, DataWriter

Public Members

dds_history_kind_t **kind**

int32_t **depth**

struct dds_inconsistent_topic_status

#include <dds_public_status.h> DCPS_Status_InconsistentTopic

Public Members

```
uint32_t total_count
int32_t total_count_change
struct dds_key_descriptor
```

Public Members

```
const char *m_name
uint32_t m_index
struct dds_liveliness_changed_status
#include <dds_public_status.h> DCPS_Status_LivelinessChanged
```

Public Members

```
uint32_t alive_count
uint32_t not_alive_count
int32_t alive_count_change
int32_t not_alive_count_change
dds_instance_handle_t last_publication_handle
struct dds_liveliness_lost_status
#include <dds_public_status.h> DCPS_Status_LivelinessLost
```

Public Members

```
uint32_t total_count
int32_t total_count_change
struct dds_offered_deadline_missed_status
#include <dds_public_status.h> DCPS_Status_OfferedDeadlineMissed
```

Public Members

```
uint32_t total_count
int32_t total_count_change
dds_instance_handle_t last_instance_handle
struct dds_offered_incompatible_qos_status
#include <dds_public_status.h> DCPS_Status_OfferedIncompatibleQoS
```

Public Members

uint32_t **total_count**

int32_t **total_count_change**

uint32_t **last_policy_id**

struct dds_publication_matched_status

#include <dds_public_status.h> DCPS_Status_PublicationMatched

Public Members

uint32_t **total_count**

int32_t **total_count_change**

uint32_t **current_count**

int32_t **current_count_change**

dds_instance_handle_t **last_subscription_handle**

struct dds_requested_deadline_missed_status

#include <dds_public_status.h> DCPS_Status_RequestedDeadlineMissed

Public Members

uint32_t **total_count**

int32_t **total_count_change**

dds_instance_handle_t **last_instance_handle**

struct dds_requested_incompatible_qos_status

#include <dds_public_status.h> DCPS_Status_RequestedIncompatibleQoS

Public Members

uint32_t **total_count**

int32_t **total_count_change**

uint32_t **last_policy_id**

struct dds_resource_limits_qospolicy

#include <dds_public_qos.h> ResourceLimits QoS: Applies to Topic, DataReader, DataWriter

Public Members

int32_t **max_samples**

int32_t **max_instances**

int32_t **max_samples_per_instance**

struct dds_sample_info

#include <dds.h> Contains information about the associated data value

Structure `dds_sample_info_t` - contains information about the associated data value

1. `sample_state` - *dds_sample_state_t*
2. `view_state` - *dds_view_state_t*
3. `instance_state` - *dds_instance_state_t*
4. `valid_data` - indicates whether there is a data associated with a sample
 - true, indicates the data is valid
 - false, indicates the data is invalid, no data to read
5. `source_timestamp` - timestamp of a data instance when it is written
6. `instance_handle` - handle to the data instance
7. `publication_handle` - handle to the publisher
8. `disposed_generation_count` - count of instance state change from NOT_ALIVE_DISPOSED to ALIVE
9. `no_writers_generation_count` - count of instance state change from NOT_ALIVE_NO_WRITERS to ALIVE
10. `sample_rank` - indicates the number of samples of the same instance that follow the current one in the collection
11. `generation_rank` - difference in generations between the sample and most recent sample of the same instance that appears in the returned collection
12. `absolute_generation_rank` - difference in generations between the sample and most recent sample of the same instance when read/take was called
13. `reception_timestamp` - timestamp of a data instance when it is added to a read queue

Public Members

dds_sample_state_t **sample_state**

Sample state

dds_view_state_t **view_state**

View state

dds_instance_state_t **instance_state**

Instance state

bool **valid_data**

Indicates whether there is a data associated with a sample

- true, indicates the data is valid
- false, indicates the data is invalid, no data to read

dds_time_t **source_timestamp**

timestamp of a data instance when it is written

dds_instance_handle_t **instance_handle**

handle to the data instance

dds_instance_handle_t **publication_handle**

handle to the publisher

uint32_t **disposed_generation_count**

count of instance state change from NOT_ALIVE_DISPOSED to ALIVE

uint32_t **no_writers_generation_count**

count of instance state change from NOT_ALIVE_NO_WRITERS to ALIVE

uint32_t **sample_rank**

indicates the number of samples of the same instance that follow the current one in the collection

uint32_t **generation_rank**

difference in generations between the sample and most recent sample of the same instance that appears in the returned collection

uint32_t **absolute_generation_rank**

difference in generations between the sample and most recent sample of the same instance when read/take was called

dds_time_t **reception_timestamp**

timestamp of a data instance when it is added to a read queue

struct dds_sample_lost_status

#include <dds_public_status.h> DCPS_Status_SampleLost

Public Members

uint32_t **total_count**

int32_t **total_count_change**

struct dds_sample_rejected_status

#include <dds_public_status.h> DCPS_Status_SampleRejected

Public Members

uint32_t **total_count**

int32_t **total_count_change**

dds_sample_rejected_status_kind **last_reason**

dds_instance_handle_t **last_instance_handle**

struct dds_sequence

Public Members

uint32_t **_maximum**

uint32_t **_length**

uint8_t * **_buffer**

bool **_release**

struct dds_stream

Public Members

dds_uptr_t **m_buffer**

size_t **m_size**

size_t **m_index**

bool **m_endian**

bool **m_failed**

struct dds_subscription_matched_status

#include <dds_public_status.h> DCPS_Status_SubscriptionMatched

Public Members

uint32_t **total_count**

int32_t **total_count_change**

uint32_t **current_count**

int32_t **current_count_change**

dds_instance_handle_t **last_publication_handle**

struct dds_topic_descriptor

Public Members

const size_t **m_size**

const uint32_t **m_align**

const uint32_t **m_flagset**

const uint32_t **m_nkeys**

const char ***m_typename**

const *dds_key_descriptor_t* ***m_keys**

const uint32_t **m_nops**

const uint32_t ***m_ops**

const char ***m_meta**

union dds_uptr_t

Public Members

uint8_t ***p8**

uint16_t ***p16**

uint32_t ***p32**

uint64_t ***p64**

float ***pf**

double *pd

void *pv

file **dds.h**

```
#include "os/os_public.h" #include "ddsc/dds_export.h" #include "ddsc/dds_public_stream.h" #include
"ddsc/dds_public_impl.h" #include "ddsc/dds_public_alloc.h" #include "ddsc/dds_public_time.h" #include
"ddsc/dds_public_qos.h" #include "ddsc/dds_public_error.h" #include "ddsc/dds_public_status.h" #include
"ddsc/dds_public_listener.h" #include "ddsc/dds_public_log.h" #include "dds_dcps_builtintopics.h" C DDS
header.
```

Communication Status definitions

DDS_INCONSISTENT_TOPIC_STATUS

Another topic exists with the same name but with different characteristics.

DDS_OFFERED_DEADLINE_MISSED_STATUS

The deadline that the writer has committed through its deadline QoS policy was not respected for a specific instance.

DDS_REQUESTED_DEADLINE_MISSED_STATUS

The deadline that the reader was expecting through its deadline QoS policy was not respected for a specific instance.

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS

A QoS policy setting was incompatible with what was requested.

DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS

A QoS policy setting was incompatible with what is offered.

DDS_SAMPLE_LOST_STATUS

A sample has been lost (never received).

DDS_SAMPLE_REJECTED_STATUS

A (received) sample has been rejected.

DDS_DATA_ON_READERS_STATUS

New information is available.

DDS_DATA_AVAILABLE_STATUS

New information is available.

DDS_LIVELINESS_LOST_STATUS

The liveliness that the DDS_DataWriter has committed through its liveliness QoS policy was not respected; thus readers will consider the writer as no longer “alive”.

DDS_LIVELINESS_CHANGED_STATUS

The liveliness of one or more writers, that were writing instances read through the readers has changed. Some writers have become “alive” or “not alive”.

DDS_PUBLICATION_MATCHED_STATUS

The writer has found a reader that matches the topic and has a compatible QoS.

DDS_SUBSCRIPTION_MATCHED_STATUS

The reader has found a writer that matches the topic and has a compatible QoS.

Typedefs

typedef enum *dds_sample_state* dds_sample_state_t

Read state for a data value

typedef enum *dds_view_state* dds_view_state_t

View state of an instance relative to the samples

typedef enum *dds_instance_state* dds_instance_state_t

Defines the state of the instance

typedef struct *dds_sample_info* dds_sample_info_t

Contains information about the associated data value

typedef bool (*dds_topic_filter_fn) (const void *sample)

Topic filter function

typedef bool (*dds_querycondition_filter_fn) (const void *sample)

typedef intptr_t dds_attach_t

Waitset attachment argument.

Every entity that is attached to the waitset can be accompanied by such an attachment argument. When the waitset wait is unblocked because of an entity that triggered, then the returning array will be populated with these attachment arguments that are related to the triggered entity.

Enums

enum dds_sample_state

Read state for a data value

Values:

DDS_SST_READ = 1u

DataReader has already accessed the sample by read

DDS_SST_NOT_READ = 2u

DataReader has not accessed the sample before

DDS_SST_READ = 1u

DDS_SST_NOT_READ = 2u

enum dds_view_state

View state of an instance relative to the samples

Values:

DDS_VST_NEW = 4u

DataReader is accessing the sample for the first time when the instance is alive

DDS_VST_OLD = 8u

DataReader accessed the sample before

DDS_VST_NEW = 4u

DDS_VST_OLD = 8u

enum dds_instance_state

Defines the state of the instance

Values:

DDS_IST_ALIVE = 16u

Samples received for the instance from the live data writers

DDS_IST_NOT_ALIVE_DISPOSED = 32u

Instance was explicitly disposed by the data writer

DDS_IST_NOT_ALIVE_NO_WRITERS = 64u

Instance has been declared as not alive by data reader as there are no live data writers writing that instance

DDS_IST_ALIVE = 16u

DDS_IST_NOT_ALIVE_DISPOSED = 32u

DDS_IST_NOT_ALIVE_NO_WRITERS = 64u

Functions

typedef _Return_type_success_(return >= 0)

Return code indicating success (DDS_RETCODE_OK) or failure. If a given operation failed the value will be a unique error code and *dds_err_nr()* must be used to extract the DDS_RETCODE_* value.

typedef _Return_type_success_(return, 0)

Handle to an entity. A valid entity handle will always have a positive integer value. Should the value be negative, the value represents a unique error code. *dds_err_nr()* can be used to extract the DDS_RETCODE_* value.

DDS_EXPORT dds_domainid_t dds_domain_default(void)

Returns the default domain identifier.

The default domain identifier can be configured in the configuration file or be set through an environment variable (VORTEX_DOMAIN).

Return Default domain identifier

_Pre_satisfies_(entity &0x7F000000)

Enable entity.

Get the topic.

Checks whether the entity has one of its enabled statuses triggered.

This operation takes an instance handle and return a key-value corresponding to it.

This operation takes a sample and returns an instance handle to be used for subsequent operations.

Get the domain id to which this entity is attached.

Get entity children.

Get entity participant.

Get entity parent.

Set entity listeners.

Get entity listeners.

Set entity QoS policies.

Get entity QoS policies.

Set status enabled on entity.

Get enabled status on entity.

Get changed status(es)

Read the status set for the entity.

Returns the instance handle that represents the entity.

Delete given entity.

This operation enables the `dds_entity_t`. Created `dds_entity_t` objects can start in either an enabled or disabled state. This is controlled by the value of the entityfactory policy on the corresponding parent entity for the given entity. Enabled entities are immediately activated at creation time meaning all their immutable QoS settings can no longer be changed. Disabled Entities are not yet activated, so it is still possible to change their immutable QoS settings. However, once activated the immutable QoS settings can no longer be changed. Creating disabled entities can make sense when the creator of the `DDS_Entity` does not yet know which QoS settings to apply, thus allowing another piece of code to set the QoS later on.

Note Delayed entity enabling is not supported yet (CHAM-96).

The default setting of `DDS_EntityFactoryQosPolicy` is such that, by default, entities are created in an enabled state so that it is not necessary to explicitly call `dds_enable` on newly-created entities.

The `dds_enable` operation produces the same results no matter how many times it is performed. Calling `dds_enable` on an already enabled `DDS_Entity` returns `DDS_RETCODE_OK` and has no effect.

If an Entity has not yet been enabled, the only operations that can be invoked on it are: the ones to set, get or copy the `QosPolicy` settings, the ones that set (or get) the Listener, the ones that get the Status and the `dds_get_status_changes` operation (although the status of a disabled entity never changes). Other operations will return the error `DDS_RETCODE_NOT_ENABLED`.

Entities created with a parent that is disabled, are created disabled regardless of the setting of the entity-factory policy.

If the entityfactory policy has `autoenable_created_entities` set to `TRUE`, the `dds_enable` operation on the parent will automatically enable all child entities created with the parent.

The Listeners associated with an Entity are not called until the Entity is enabled. Conditions associated with an Entity that is not enabled are “inactive”, that is, have a `trigger_value` which is `FALSE`.

This operation will delete the given entity. It will also automatically delete all its children, childrens' children, etc entities.

Return A `dds_return_t` indicating success or failure.

Parameters

- `entity`: The entity to enable.

Return Value

- `DDS_RETCODE_OK`: The listeners of to the entity have been successfully been copied into the specified listener parameter.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The parent of the given Entity is not enabled.

This operation reads the status(es) set for the entity based on the enabled status and mask set. It does not clear the read status(es).

Return A `dds_return_t` indicating success or failure.

Return A `dds_return_t` indicating success or failure.

Parameters

- `entity`: Entity to delete.

Return Value

- `DDS_RETCODE_OK`: The entity and its children (recursive) are deleted.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

Parameters

- `entity`: Entity of which to get the instance handle.
- `ihdl`: Pointer to `dds_instance_handle_t`.

Return Value

- `DDS_RETCODE_OK`: Success.
- `DDS_RETCODE_ERROR`: An internal error has occurred.

This operation reads the status(es) set for the entity based on the enabled status and mask set. It clears the status set after reading.

Return A `dds_return_t` indicating success or failure.

Parameters

- `entity`: Entity on which the status has to be read.
- `status`: Returns the status set on the entity, based on the enabled status.
- `mask`: Filter the status condition to be read (can be NULL).

Return Value

- `DDS_RETCODE_OK`: Success.
- `DDS_RETCODE_BAD_PARAMETER`: The entity parameter is not a valid parameter.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation returns the status changes since they were last read.

Return A `dds_return_t` indicating success or failure.

Parameters

- `entity`: Entity on which the status has to be read.
- `status`: Returns the status set on the entity, based on the enabled status.
- `mask`: Filter the status condition to be read (can be NULL).

Return Value

- `DDS_RETCODE_OK`: Success.
- `DDS_RETCODE_BAD_PARAMETER`: The entity parameter is not a valid parameter.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.

- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation returns the status enabled on the entity

Return A `dds_return_t` indicating success or failure.

Parameters

- `entity`: Entity on which the statuses are read.
- `status`: Returns the current set of triggered statuses.

Return Value

- `DDS_RETCODE_OK`: Success.
- `DDS_RETCODE_BAD_PARAMETER`: The entity parameter is not a valid parameter.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation enables the status(es) based on the mask set

Return A `dds_return_t` indicating success or failure.

Parameters

- `entity`: Entity to get the status.
- `status`: Status set on the entity.

Return Value

- `DDS_RETCODE_OK`: Success.
- `DDS_RETCODE_BAD_PARAMETER`: The entity parameter is not a valid parameter.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation allows access to the existing set of QoS policies for the entity.

Return A `dds_return_t` indicating success or failure.

Parameters

- `entity`: Entity to enable the status.
- `mask`: Status value that indicates the status to be enabled.

Return Value

- `DDS_RETCODE_OK`: Success.
- `DDS_RETCODE_BAD_PARAMETER`: The entity parameter is not a valid parameter.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation replaces the existing set of Qos Policy settings for an entity. The parameter `qos` must contain the struct with the `QosPolicy` settings which is checked for self-consistency.

Return A `dds_return_t` indicating success or failure.

Parameters

- `entity`: Entity on which to get qos.

- `qos`: Pointer to the qos structure that returns the set policies.

Return Value

- `DDS_RETCODE_OK`: The existing set of QoS policy values applied to the entity has successfully been copied into the specified qos parameter.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The qos parameter is NULL.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

The set of QoS policy settings specified by the qos parameter are applied on top of the existing QoS, replacing the values of any policies previously set (provided, the operation returned `DDS_RETCODE_OK`).

Not all policies are changeable when the entity is enabled.

This operation allows access to the existing listeners attached to the entity.

Note Currently only Latency Budget and Ownership Strength are changeable QoS that can be set.

Return A `dds_return_t` indicating success or failure.

Parameters

- `entity`: Entity from which to get qos.
- `qos`: Pointer to the qos structure that provides the policies.

Return Value

- `DDS_RETCODE_OK`: The new QoS policies are set.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The qos parameter is NULL.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_IMMUTABLE_POLICY`: The entity is enabled and one or more of the policies of the QoS are immutable.
- `DDS_RETCODE_INCONSISTENT_POLICY`: A few policies within the QoS are not consistent with each other.

This operation attaches a `dds_listener_t` to the `dds_entity_t`. Only one Listener can be attached to each Entity. If a Listener was already attached, this operation will replace it with the new one. In other words, all related callbacks are replaced (possibly with NULL).

Return A `dds_return_t` indicating success or failure.

Parameters

- `entity`: Entity on which to get the listeners.
- `listener`: Pointer to the listener structure that returns the set of listener callbacks.

Return Value

- `DDS_RETCODE_OK`: The listeners of to the entity have been successfully been copied into the specified listener parameter.
- `DDS_RETCODE_ERROR`: An internal error has occurred.

- `DDS_RETCODE_BAD_PARAMETER`: The listener parameter is NULL.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

When listener parameter is NULL, all listener callbacks that were possibly set on the Entity will be removed.

For each communication status, the `StatusChangedFlag` flag is initially set to FALSE. It becomes TRUE whenever that plain communication status changes. For each plain communication status activated in the mask, the associated Listener callback is invoked and the communication status is reset to FALSE, as the listener implicitly accesses the status which is passed as a parameter to that operation. The status is reset prior to calling the listener, so if the application calls the `get_<status_name>` from inside the listener it will see the status already reset.

Note Not all listener callbacks are related to all entities.

In case a related callback within the Listener is not set, the Listener of the Parent entity is called recursively, until a Listener with the appropriate callback set has been found and called. This allows the application to set (for instance) a default behaviour in the Listener of the containing Publisher and a `DataWriter` specific behaviour when needed. In case the callback is not set in the Publishers' Listener either, the communication status will be propagated to the Listener of the `DomainParticipant` of the containing `DomainParticipant`. In case the callback is not set in the `DomainParticipants`' Listener either, the `Communication Status` flag will be set, resulting in a possible `WaitSet` trigger.

This operation returns the parent to which the given entity belongs. For instance, it will return the Participant that was used when creating a `Publisher` (when that `Publisher` was provided here).

Return A `dds_return_t` indicating success or failure.

Parameters

- `entity`: Entity on which to get the listeners.
- `listener`: Pointer to the listener structure that contains the set of listener callbacks (maybe NULL).

Return Value

- `DDS_RETCODE_OK`: The listeners of to the entity have been successfully been copied into the specified listener parameter.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

When a reader or a writer are created with a partition, then a subscriber or publisher respectively are created implicitly. These implicit subscribers or publishers will be deleted automatically when the reader or writer is deleted. However, when this function returns such an implicit entity, it is from there on out considered 'explicit'. This means that it isn't deleted automatically anymore. The application should explicitly call `dds_delete` on those entities now (or delete the parent participant which will delete all entities within its hierarchy).

This operation returns the participant to which the given entity belongs. For instance, it will return the Participant that was used when creating a `Publisher` that was used to create a `DataWriter` (when that `DataWriter` was provided here).

Return A valid entity handle or an error code.

Parameters

- `entity`: Entity from which to get its parent.

Return Value

- `>0`: A valid entity handle.
- `DDS_ENTITY_NIL`: Called with a participant.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

TODO: Link to generic dds entity relations documentation.

This operation returns the children that the entity contains. For instance, it will return all the Topics, Publishers and Subscribers of the Participant that was used to create those entities (when that Participant is provided here).

Return A valid entity or an error code.

Parameters

- `entity`: Entity from which to get its participant.

Return Value

- `>0`: A valid participant handle.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This functions takes a pre-allocated list to put the children in and will return the number of found children. It is possible that the given size of the list is not the same as the number of found children. If less children are found, then the last few entries in the list are untouched. When more children are found, then only 'size' number of entries are inserted into the list, but still complete count of the found children is returned. Which children are returned in the latter case is undefined.

When supplying NULL as list and 0 as size, you can use this to acquire the number of children without having to pre-allocate a list.

When a reader or a writer are created with a partition, then a subscriber or publisher respectively are created implicitly. These implicit subscribers or publishers will be deleted automatically when the reader or writer is deleted. However, when this function returns such an implicit entity, it is from there on out considered 'explicit'. This means that it isn't deleted automatically anymore. The application should explicitly call `dds_delete` on those entities now (or delete the parent participant which will delete all entities within its hierarchy).

When creating a participant entity, it is attached to a certain domain. All the children (like Publishers) and childrens' children (like DataReaders), etc are also attached to that domain.

Return Number of children or an error code.

Parameters

- `entity`: Entity from which to get its children.
- `children`: Pre-allocated array to contain the found children.
- `size`: Size of the pre-allocated children's list.

Return Value

- `>=0`: Number of childer found children (can be larger than 'size').
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The children parameter is NULL, while a size is provided.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This function will return the original domain ID when called on any of the entities within that hierarchy.

This operation returns a topic (handle) when the function call is done with reader, writer, read condition or query condition. For instance, it will return the topic when it is used for creating the reader or writer. For the conditions, it returns the topic that is used for creating the reader which was used to create the condition.

Return A `dds_return_t` indicating success or failure.

Return instance handle or `DDS_HANDLE_NIL` if instance could not be found from key.

Return A `dds_return_t` indicating success or failure.

Return A `dds_return_t` indicating success or failure.

Parameters

- `entity`: Entity from which to get its children.
- `id`: Pointer to put the domain ID in.

Return Value

- `DDS_RETCODE_OK`: Domain ID was returned.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The id parameter is NULL.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

Parameters

- `entity`: Reader or Writer entity.
- `data`: Sample with a key fields set.

Parameters

- `entity`: Reader or writer entity.
- `inst`: Instance handle.
- `data`: pointer to an instance, to which the key ID corresponding to the instance handle will be returned, the sample in the instance should be ignored.

Return Value

- `DDS_RETCODE_OK`: The operation was successful.
- `DDS_RETCODE_BAD_PARAMETER`: One of the parameters was invalid or the topic does not exist.
- `DDS_RETCODE_ERROR`: An internal error has occurred.

Parameters

- `entity`: Entity for which to check for triggered status.

Return Value

- `DDS_RETCODE_OK`: The operation was successful.
- `DDS_RETCODE_BAD_PARAMETER`: The entity parameter is not a valid parameter.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation will delete the given entity. It will also automatically delete all its children, childrens' children, etc entities.

Return A `dds_return_t` indicating success or failure.

Parameters

- `entity`: The entity.

Return Value

- `DDS_RETCODE_OK`: The operation was successful.
- `DDS_RETCODE_BAD_PARAMETER`: The entity parameter is not a valid parameter.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

TODO: Link to generic dds entity relations documentation.

Description : Read the status(es) set for the entity based on the enabled status and mask set. This operation does not clear the read status(es).

Return 0 - Success (`DDS_RETCODE_OK`).

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `entity`: Entity from which to get its parent.

Return Value

- `DDS_RETCODE_ERROR`: The entity and its children (recursive are deleted).
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

Arguments :

1. `e` Entity on which the status has to be read
2. `status` Returns the status set on the entity, based on the enabled status
3. `mask` Filter the status condition to be read (can be NULL)
4. Returns 0 on success, or a non-zero error value if the mask does not correspond to the entity

Description : Read the status(es) set for the entity based on the enabled status and mask set. This operation clears the status set after reading.

Arguments :

1. `e` Entity on which the status has to be read
2. `status` Returns the status set on the entity, based on the enabled status

3. mask Filter the status condition to be read (can be NULL)
4. Returns 0 on success, or a non-zero error value if the mask does not correspond to the entity

Description : Returns the status changes since they were last read.

Arguments :

1. e Entity on which the statuses are read
2. Returns the current set of triggered statuses.

Description : This operation returns the status enabled on the entity

Arguments :

1. e Entity to get the status
2. Returns the status that are enabled for the entity

Description : This operation enables the status(es) based on the mask set

Arguments :

1. e Entity to enable the status
2. mask Status value that indicates the status to be enabled
3. Returns 0 on success, or a non-zero error value indicating failure if the mask does not correspond to the entity.

This operation allows access to the existing set of QoS policies for the entity.

TODO: Link to generic QoS information documentation.

This operation replaces the existing set of QoS Policy settings for an entity. The parameter qos must contain the struct with the QoSPolicy settings which is checked for self-consistency.

Return 0 - Success (DDS_RETCODE_OK).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- e: Entity on which to get qos
- qos: Pointer to the qos structure that returns the set policies

Return Value

- DDS_RETCODE_OK: The existing set of QoS policy values applied to the entity has successfully been copied into the specified qos parameter.
- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER: The qos parameter is NULL.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

The set of QoSPolicy settings specified by the qos parameter are applied on top of the existing QoS, replacing the values of any policies previously set (provided, the operation returned DDS_RETCODE_OK).

Not all policies are changeable when the entity is enabled.

TODO: Link to generic QoS information documentation.

This operation allows access to the existing listeners attached to the entity.

Note Currently only Latency Budget and Ownership Strength are changeable QoS that can be set.

Return 0 - Success (DDS_RETCODE_OK).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *e*: Entity from which to get qos
- *qos*: Pointer to the qos structure that provides the policies

Return Value

- DDS_RETCODE_OK: The new QoS policies are set.
- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER: The qos parameter is NULL.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.
- DDS_RETCODE_IMMUTABLE_POLICY: The entity is enabled and one or more of the policies of the QoS are immutable.
- DDS_RETCODE_INCONSISTENT_POLICY: A few policies within the QoS are not consistent with each other.

TODO: Link to (generic) Listener and status information.

This operation attaches a *dds_listener_t* to the *dds_entity_t*. Only one Listener can be attached to each Entity. If a Listener was already attached, this operation will replace it with the new one. In other words, all related callbacks are replaced (possibly with NULL).

Return 0 - Success (DDS_RETCODE_OK).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *e*: Entity on which to get the listeners
- *listener*: Pointer to the listener structure that returns the set of listener callbacks.

Return Value

- DDS_RETCODE_OK: The listeners of to the entity have been successfully been copied into the specified listener parameter.
- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER: The listener parameter is NULL.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

When listener parameter is NULL, all listener callbacks that were possibly set on the Entity will be removed.

TODO: Link to (generic) Listener and status information.

Note Not all listener callbacks are related to all entities.

For each communication status, the *StatusChangedFlag* flag is initially set to FALSE. It becomes TRUE whenever that plain communication status changes. For each plain communication status activated in the mask, the associated Listener callback is invoked and the communication status is reset to FALSE, as the

listener implicitly accesses the status which is passed as a parameter to that operation. The status is reset prior to calling the listener, so if the application calls the `get_<status_name>` from inside the listener it will see the status already reset.

In case a related callback within the Listener is not set, the Listener of the Parent entity is called recursively, until a Listener with the appropriate callback set has been found and called. This allows the application to set (for instance) a default behaviour in the Listener of the containing Publisher and a DataWriter specific behaviour when needed. In case the callback is not set in the Publishers' Listener either, the communication status will be propagated to the Listener of the DomainParticipant of the containing DomainParticipant. In case the callback is not set in the DomainParticipants' Listener either, the Communication Status flag will be set, resulting in a possible WaitSet trigger.

This operation returns the parent to which the given entity belongs. For instance, it will return the Participant that was used when creating a Publisher (when that Publisher was provided here).

Return 0 - Success (DDS_RETCODE_OK).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- `e`: Entity on which to get the listeners
- `listener`: Pointer to the listener structure that contains the set of listener callbacks (maybe NULL).

Return Value

- DDS_RETCODE_OK: The listeners of to the entity have been successfully been copied into the specified listener parameter.
- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

TODO: Link to generic dds entity relations documentation.

This operation returns the participant to which the given entity belongs. For instance, it will return the Participant that was used when creating a Publisher that was used to create a DataWriter (when that DataWriter was provided here).

Return >0 - Success (valid entity handle).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- `entity`: Entity from which to get its parent.

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

TODO: Link to generic dds entity relations documentation.

This operation returns the children that the entity contains. For instance, it will return all the Topics, Publishers and Subscribers of the Participant that was used to create those entities (when that Participant is provided here).

Return >0 - Success (valid entity handle).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *entity*: Entity from which to get its participant.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This function takes a pre-allocated list to put the children in and will return the number of found children. It is possible that the given size of the list is not the same as the number of found children. If less children are found, then the last few entries in the list are untouched. When more children are found, then only 'size' number of entries are inserted into the list, but still complete count of the found children is returned. Which children are returned in the latter case is undefined.

When supplying NULL as list and 0 as size, you can use this to acquire the number of children without having to pre-allocate a list.

TODO: Link to generic dds entity relations documentation.

When creating a participant entity, it is attached to a certain domain. All the children (like Publishers) and children's children (like DataReaders), etc are also attached to that domain.

Return >=0 - Success (number of found children, can be larger than 'size').

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *entity*: Entity from which to get its children.
- *children*: Pre-allocated array to contain the found children.
- *size*: Size of the pre-allocated children's list.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The children parameter is NULL, while a size is provided.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This function will return the original domain ID when called on any of the entities within that hierarchy.

Description : Checks whether the entity has one of its enabled statuses triggered.

Return 0 - Success (`DDS_RETCODE_OK`).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *entity*: Entity from which to get its children.
- *id*: Pointer to put the domain ID in.

Return Value

- `DDS_RETCODE_OK`: Domain ID was returned.
- `DDS_RETCODE_ERROR`: An internal error has occurred.

- DDS_RETCODE_BAD_PARAMETER: The id parameter is NULL.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

Arguments :

1. e Entity for which to check for triggered status

`_Pre_satisfies_(((writer &(0x7F000000))==DDS_KIND_WRITER))`

Get entity publisher.

This operation returns the publisher to which the given entity belongs. For instance, it will return the Publisher that was used when creating a DataWriter (when that DataWriter was provided here).

Return A valid entity or an error code.

Parameters

- entity: Entity from which to get its publisher.

Return Value

- >0: A valid publisher handle.
- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

`_Pre_satisfies_(((entity &(0x7F000000))==DDS_KIND_READER) || ((entity &(0x7F000000))==DDS_KIND_COND_READ))`

Get entity subscriber.

This operation returns the subscriber to which the given entity belongs. For instance, it will return the Subscriber that was used when creating a DataReader (when that DataReader was provided here).

Return A valid subscriber handle or an error code.

Parameters

- entity: Entity from which to get its subscriber.

Return Value

- >0: A valid subscriber handle.
- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

`_Pre_satisfies_(((condition &(0x7F000000))==DDS_KIND_COND_READ) || ((condition &(0x7F000000))==DDS_KIND_COND_READ))`

Get entity datareader.

Get the mask of a condition.

This operation returns the datareader to which the given entity belongs. For instance, it will return the DataReader that was used when creating a ReadCondition (when that ReadCondition was provided here).

This operation returns the mask that was used to create the given condition.

Return A valid reader handle or an error code.

Parameters

- `entity`: Entity from which to get its datareader.

Return Value

- `>0`: A valid reader handle.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation returns the mask that was used to create the given condition.

Return A `dds_return_t` indicating success or failure.

Parameters

- `condition`: Read or Query condition that has a mask.

Return Value

- `DDS_RETCODE_OK`: Success (given mask is set).
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The mask arg is NULL.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

Return 0 - Success (given mask is set).

Return `<0` - Failure (use `dds_err_nr()` to get error value).

Parameters

- `condition`: Read or Query condition that has a mask.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The mask arg is NULL.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

DDS_EXPORT Must inspect result `dds_entity_t dds_create_participant(_In_ const dds_domain_t domain, const dds_qos_t qos, const dds_listener_t listener)`

Creates a new instance of a DDS participant in a domain.

If domain is set (not `DDS_DOMAIN_DEFAULT`) then it must match if the domain has also been configured or an error status will be returned. Currently only a single domain can be configured by providing configuration file. If no configuration file exists, the default domain is configured as 0.

Return A valid participant handle or an error code.

Parameters

- `domain`: The domain in which to create the participant (can be `DDS_DOMAIN_DEFAULT`). Valid values for domain id are between 0 and 230. `DDS_DOMAIN_DEFAULT` is for using the domain in the configuration.
- `qos`: The QoS to set on the new participant (can be NULL).
- `listener`: Any listener functions associated with the new participant (can be NULL).

Return Value

- >0: A valid participant handle.
- DDS_RETCODE_ERROR: An internal error has occurred.

DDS_EXPORT _Check_return_ dds_return_t dds_lookup_participant(_In_ dds_domainid_t domain_id, dds_participant_t* participants, size_t size)

Get participants of a domain.

This operation acquires the participants created on a domain and returns the number of found participants.

This function takes a domain id with the size of pre-allocated participant's list in and will return the number of found participants. It is possible that the given size of the list is not the same as the number of found participants. If less participants are found, then the last few entries in an array stay untouched. If more participants are found and the array is too small, then the participants returned are undefined.

Return Number of participants found or an error code.

Parameters

- domain_id: The domain id.
- participants: The participant for domain.
- size: Size of the pre-allocated participant's list.

Return Value

- >0: Number of participants found.
- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER: The participant parameter is NULL, while a size is provided.

_Pre_satisfies_((participant &(0x7F000000)) == DDS_KIND_PARTICIPANT)

Creates a new topic.

Create a waitset and allocate the resources required.

Creates a new instance of a DDS publisher.

Creates a new instance of a DDS subscriber.

Finds a named topic.

The type name for the topic is taken from the generated descriptor. Topic matching is done on a combination of topic name and type name.

The returned topic should be released with dds_delete.

Return A valid topic handle or an error code.

Parameters

- participant: Participant on which to create the topic.
- descriptor: An IDL generated topic descriptor.
- name: Name of the topic.
- qos: QoS to set on the new topic (can be NULL).
- listener: Any listener functions associated with the new topic (can be NULL).

Return Value

- >=0: A valid topic handle.

- `DDS_RETCODE_BAD_PARAMETER`: Either participant, descriptor, name or qos is invalid.

A WaitSet object allows an application to wait until one or more of the conditions of the attached entities evaluates to TRUE or until the timeout expires.

Return A valid topic handle or an error code.

Return A valid subscriber handle or an error code.

Return A valid publisher handle or an error code.

Parameters

- `participant`: The participant on which to find the topic.
- `name`: The name of the topic to find.

Return Value

- `>0`: A valid topic handle.
- `DDS_RETCODE_BAD_PARAMETER`: Participant was invalid.

Parameters

- `participant`: The participant on which the subscriber is being created.
- `qos`: The QoS to set on the new subscriber (can be NULL).
- `listener`: Any listener functions associated with the new subscriber (can be NULL).

Return Value

- `>0`: A valid subscriber handle.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the parameters is invalid.

Parameters

- `participant`: The participant to create a publisher for.
- `qos`: The QoS to set on the new publisher (can be NULL).
- `listener`: Any listener functions associated with the new publisher (can be NULL).

Return Value

- `>0`: A valid publisher handle.
- `DDS_RETCODE_ERROR`: An internal error has occurred.

Description : Finds a named topic. Returns NULL if does not exist. The returned topic should be released with `dds_delete`.

Return A valid waitset handle or an error code.

Parameters

- `participant`: Domain participant which the WaitSet contains.

Return Value

- `>=0`: A valid waitset handle.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

Arguments :

1. pp The participant on which to find the topic
2. name The name of the topic to find
3. Returns a topic, NULL if could not be found or error

A WaitSet object allows an application to wait until one or more of the conditions of the attached entities evaluates to TRUE or until the timeout expires.

Return >0 - Success (valid handle of a subscriber entity).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Return >0 - Success (valid handle of a publisher entity).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- participant: The participant on which the subscriber is being created
- qos: The QoS to set on the new subscriber (can be NULL)
- listener: Any listener functions associated with the new subscriber (can be NULL)

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER One of the parameters is invalid

Parameters

- participant: The participant to create a publisher for
- qos: The QoS to set on the new publisher (can be NULL)
- listener: Any listener functions associated with the new publisher (can be NULL)

Return >0 - Success (valid waitset).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- participant: Domain participant which the WaitSet contains.

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

`_Pre_satisfies_((topic &(0x7F000000)) == DDS_KIND_TOPIC)`

Returns the name of a given topic.

Gets the filter for a topic.

Sets a filter on a topic.

Returns the type name of a given topic.

Description : Returns a topic type name.

Return A dds_return_t indicating success or failure.

Return A dds_return_t indicating success or failure.

Return DDS_RETCODE_OK Success.

Return The topic filter.

Parameters

- `topic`: The topic.
- `name`: Buffer to write the topic name to.
- `size`: Number of bytes available in the buffer.

Return Value

- DDS_RETCODE_OK: Success.

Parameters

- `topic`: The topic.
- `name`: Buffer to write the topic type name to.
- `size`: Number of bytes available in the buffer.

Parameters

- `topic`: The topic on which the content filter is set.
- `filter`: The filter function used to filter topic samples.
- `topic`: The topic from which to get the filter.

Arguments :

1. `topic` The topic
2. Returns The topic type name or NULL to indicate an error

`_Out_writes_z_ (size)`

`_Pre_satisfies_((publisher & (0x7F000000)) == DDS_KIND_PUBLISHER)`

Suspends the publications of the Publisher.

Resumes the publications of the Publisher.

This operation is a hint to the Service so it can optimize its performance by e.g., collecting modifications to DDS writers and then batching them. The Service is not required to use the hint.

Every invocation of this operation must be matched by a corresponding call to This operation is a hint to the Service to indicate that the application has completed changes initiated by a previous `dds_suspend()`. The Service is not required to use the hint.

See `dds_resume` indicating that the set of modifications has completed.

Return A `dds_return_t` indicating success or failure.

Parameters

- `publisher`: The publisher for which all publications will be suspended.

Return Value

- DDS_RETCODE_OK: Publications suspended successfully.
- DDS_RETCODE_BAD_PARAMETER: The pub parameter is not a valid publisher.
- DDS_RETCODE_UNSUPPORTED: Operation is not supported.

The call to `resume_publications` must match a previous call to `suspend_publications`. This operation is a hint to the Service to indicate that the application has completed changes initiated by a previous `suspend_publications`. The call to `resume_publications` must match a previous call to `suspend_publications`.

See `suspend_publications`.

Return A `dds_return_t` indicating success or failure.

See `suspend`. The Service is not required to use the hint.

See `suspend_publications`.

Return `>0` - Success.

Return `<0` - Failure (use `dds_err_nr()` to get error value).

See `dds_suspend`.

Parameters

- `publisher`: The publisher for which all publications will be resumed.

Return Value

- `DDS_RETCODE_OK`: Publications resumed successfully.
- `DDS_RETCODE_BAD_PARAMETER`: The `pub` parameter is not a valid publisher.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: No previous matching `dds_suspend()`.
- `DDS_RETCODE_UNSUPPORTED`: Operation is not supported.

Parameters

- `publisher`: The publisher for which all publications will be resumed

Return Value

- `DDS_RETCODE_OK`: Publications resumed successfully.
- `DDS_RETCODE_BAD_PARAMETER`: The `pub` parameter is not a valid publisher.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: No previous matching

Return Value

- `DDS_RETCODE_UNSUPPORTED`: Operation is not supported.

`_Pre_satisfies_(((publisher_or_writer & (0x7F000000)) == DDS_KIND_WRITER) || ((publisher_or_`

`writer & (0x7F000000)) == DDS_KIND_WRITER) || ((publisher_or_`
Waits at most for the duration timeout for acks for data in the publisher or writer.

This operation blocks the calling thread until either all data written by the publisher or writer is acknowledged by all matched reliable reader entities, or else the duration specified by the timeout parameter elapses, whichever happens first.

Return A `dds_return_t` indicating success or failure.

Parameters

- `publisher_or_writer`: Publisher or writer whose acknowledgments must be waited for
- `timeout`: How long to wait for acknowledgments before time out

Return Value

- `DDS_RETCODE_OK`: All acknowledgments successfully received with the timeout.
- `DDS_RETCODE_BAD_PARAMETER`: The `publisher_or_writer` is not a valid publisher or writer.

- `DDS_RETCODE_TIMEOUT`: Timeout expired before all acknowledgments from reliable reader entities were received.
- `DDS_RETCODE_UNSUPPORTED`: Operation is not supported.

`_Pre_satisfies_((participant_or_subscriber &(0x7F000000)) == DDS_KIND_SUBSCRIBER) | ((pa`
Creates a new instance of a DDS reader.

This implicit subscriber will be deleted automatically when the created reader is deleted.

Return A valid reader handle or an error code.

Parameters

- `participant_or_subscriber`: The participant or subscriber on which the reader is being created.
- `topic`: The topic to read.
- `qos`: The QoS to set on the new reader (can be NULL).
- `listener`: Any listener functions associated with the new reader (can be NULL).

Return Value

- `>0`: A valid reader handle.
- `DDS_RETCODE_ERROR`: An internal error occurred.

`_Pre_satisfies_(reader &(0x7F000000)) = =DDS_KIND_READER)`

Wait until reader receives all historic data.

Read and copy the status set for the loaned sample.

Read and copy the status set for the entity.

Read, copy and remove the status set for the entity.

Creates a queryondition associated to the given reader.

Creates a readcondition associated to the given reader.

The operation blocks the calling thread until either all “historical” data is received, or else the duration specified by the `max_wait` parameter elapses, whichever happens first. A return value of 0 indicates that all the “historical” data was received; a return value of `TIMEOUT` indicates that `max_wait` elapsed before all the data was received.

The readcondition allows specifying which samples are of interest in a data reader’s history, by means of a mask. The mask is or’d with the flags that are `dds_sample_state_t`, `dds_view_state_t` and `dds_instance_state_t`.

Return a status, 0 on success, `TIMEOUT` on timeout or a negative value to indicate error.

Parameters

- `reader`: The reader on which to wait for historical data.
- `max_wait`: How long to wait for historical data before time out.

Based on the mask value set, the readcondition gets triggered when data is available on the reader.

Waitsets allow waiting for an event on some of any set of entities. This means that the readcondition can be used to wake up a waitset when data is in the reader history with states that matches the given mask.

The queryondition allows specifying which samples are of interest in a data reader’s history, by means of a mask and a filter. The mask is or’d with the flags that are `dds_sample_state_t`, `dds_view_state_t` and `dds_instance_state_t`.

Note The parent reader and every of its associated conditions (whether they are readconditions or queryconditions) share the same resources. This means that one of these entities reads or takes data, the states of the data will change for other entities automatically. For instance, if one reads a sample, then the sample state will become ‘read’ for all associated reader/conditions. Or if one takes a sample, then it’s not available to any other associated reader/condition.

Return A valid condition handle or an error code.

Parameters

- `reader`: Reader to associate the condition to.
- `mask`: Interest (`dds_sample_state_t``ldds_view_state_t``ldds_instance_state_t`).

Return Value

- `>0`: A valid condition handle
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

Based on the mask value set and data that matches the filter, the querycondition gets triggered when data is available on the reader.

Waitsets allow waiting for an event on some of any set of entities. This means that the querycondition can be used to wake up a waitset when data is in the reader history with states that matches the given mask and filter.

This operation copies the next, non-previously accessed data value and corresponding sample info and removes from the data reader. As an entity, only reader is accepted.

Note The parent reader and every of its associated conditions (whether they are readconditions or queryconditions) share the same resources. This means that one of these entities reads or takes data, the states of the data will change for other entities automatically. For instance, if one reads a sample, then the sample state will become ‘read’ for all associated reader/conditions. Or if one takes a sample, then it’s not available to any other associated reader/condition.

Return A valid condition handle or an error code

Parameters

- `reader`: Reader to associate the condition to.
- `mask`: Interest (`dds_sample_state_t``ldds_view_state_t``ldds_instance_state_t`).
- `filter`: Callback that the application can use to filter specific samples.

Return Value

- `>=0`: A valid condition handle.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation copies the next, non-previously accessed data value and corresponding sample info and removes from the data reader. As an entity, only reader is accepted.

Return A `dds_return_t` indicating success or failure.

Parameters

- `reader`: The reader entity.
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL).
- `si`: The pointer to *dds_sample_info_t* returned for a data value.

Return Value

- `DDS_RETCODE_OK`: The operation was successful.
- `DDS_RETCODE_BAD_PARAMETER`: The entity parameter is not a valid parameter.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

After `dds_take_next_wl` function is being called and the data has been handled, `dds_return_loan` function must be called to possibly free memory.

This operation copies the next, non-previously accessed data value and corresponding sample info. As an entity, only reader is accepted.

Return A `dds_return_t` indicating success or failure.

Parameters

- `reader`: The reader entity.
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL).
- `si`: The pointer to *dds_sample_info_t* returned for a data value.

Return Value

- `DDS_RETCODE_OK`: The operation was successful.
- `DDS_RETCODE_BAD_PARAMETER`: The entity parameter is not a valid parameter.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation copies the next, non-previously accessed data value and corresponding loaned sample info. As an entity, only reader is accepted.

Return A `dds_return_t` indicating success or failure.

Parameters

- `reader`: The reader entity.
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL).
- `si`: The pointer to *dds_sample_info_t* returned for a data value.

Return Value

- `DDS_RETCODE_OK`: The operation was successful.
- `DDS_RETCODE_BAD_PARAMETER`: The entity parameter is not a valid parameter.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

After `dds_read_next_wl` function is being called and the data has been handled, `dds_return_loan` function must be called to possibly free memory.

The readcondition allows specifying which samples are of interest in a data reader's history, by means of a mask. The mask is or'd with the flags that are `dds_sample_state_t`, `dds_view_state_t` and `dds_instance_state_t`.

Return A `dds_return_t` indicating success or failure.

Parameters

- `reader`: The reader entity.
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL).
- `si`: The pointer to `dds_sample_info_t` returned for a data value.

Return Value

- `DDS_RETCODE_OK`: The operation was successful.
- `DDS_RETCODE_BAD_PARAMETER`: The entity parameter is not a valid parameter.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

Based on the mask value set, the readcondition gets triggered when data is available on the reader.

Waitsets allow waiting for an event on some of any set of entities. This means that the readcondition can be used to wake up a waitset when data is in the reader history with states that matches the given mask.

Note The parent reader and every of its associated conditions (whether they are readconditions or queryconditions) share the same resources. This means that one of these entities reads or takes data, the states of the data will change for other entities automatically. For instance, if one reads a sample, then the sample state will become 'read' for all associated reader/conditions. Or if one takes a sample, then it's not available to any other associated reader/condition.

Return >0 - Success (valid condition).

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader`: Reader to associate the condition to.
- `mask`: Interest (`dds_sample_state_t|dds_view_state_t|dds_instance_state_t`).

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

`_Pre_satisfies_((participant_or_publisher &(0x7F000000))==DDS_KIND_PUBLISHER) | ((part.`
Creates a new instance of a DDS writer.

This implicit publisher will be deleted automatically when the created writer is deleted.

Return A valid writer handle or an error code.

Return >0 A valid writer handle.

Return `DDS_RETCODE_ERROR` An internal error occurred.

Parameters

- `participant_or_publisher`: The participant or publisher on which the writer is being created.
- `topic`: The topic to write.
- `qos`: The QoS to set on the new writer (can be NULL).
- `listener`: Any listener functions associated with the new writer (can be NULL).

`_Pre_satisfies_((writer &(0x7F000000)) == DDS_KIND_WRITER)`

Registers an instance.

Write the value of a data instance along with the source timestamp passed.

Write a CDR serialized value of a data instance.

Write the value of a data instance.

This operation disposes an instance with a specific timestamp, identified by the instance handle.

This operation disposes an instance, identified by the instance handle.

This operation disposes an instance with a specific timestamp, identified by the data sample.

This operation disposes an instance, identified by the data sample.

This operation modifies and disposes a data instance with a specific timestamp.

This operation modifies and disposes a data instance.

Unregisters an instance.

This operation registers an instance with a key value to the data writer and returns an instance handle that could be used for successive write & dispose operations. When the handle is not allocated, the function will return an error and the handle will be un-touched.

This operation reverses the action of register instance, removes all information regarding the instance and unregisters an instance with a key value from the data writer.

Return A `dds_return_t` indicating success or failure.

Parameters

- `writer`: The writer to which instance has be associated.
- `handle`: The instance handle.
- `data`: The instance with the key value.

Return Value

- `DDS_RETCODE_OK`: The operation was successful.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.

This operation unregisters the instance which is identified by the key fields of the given typed instance handle.

Return A `dds_return_t` indicating success or failure.

Parameters

- `writer`: The writer to which instance is associated.
- `data`: The instance with the key value.

Return Value

- `DDS_RETCODE_OK`: The operation was successful.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.

This operation reverses the action of register instance, removes all information regarding the instance and unregisters an instance with a key value from the data writer. It also provides a value for the timestamp explicitly.

Return A `dds_return_t` indicating success or failure.

Parameters

- `writer`: The writer to which instance is associated.
- `handle`: The instance handle.

Return Value

- `DDS_RETCODE_OK`: The operation was successful.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.

This operation unregisters an instance with a key value from the handle. Instance can be identified from instance handle. If an unregistered key ID is passed as an instance data, an error is logged and not flagged as return value.

Return A `dds_return_t` indicating success or failure.

Parameters

- `writer`: The writer to which instance is associated.
- `data`: The instance with the key value.
- `timestamp`: The timestamp used at registration.

Return Value

- `DDS_RETCODE_OK`: The operation was successful.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.

This operation requests the Data Distribution Service to modify the instance and mark it for deletion. Copies of the instance and its corresponding samples, which are stored in every connected reader and, dependent on the QoS policy settings (also in the Transient and Persistent stores) will be modified and marked for deletion by setting their `dds_instance_state_t` to `DDS_IST_NOT_ALIVE_DISPOSED`.

Return A `dds_return_t` indicating success or failure.

Parameters

- `writer`: The writer to which instance is associated.
- `handle`: The instance handle.
- `timestamp`: The timestamp used at registration.

Return Value

- `DDS_RETCODE_OK`: The operation was successful
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid

- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object

If the history QoS policy is set to `DDS_HISTORY_KEEP_ALL`, the `dds_writedispose` operation on the writer may block if the modification would cause data to be lost because one of the limits, specified in the `resource_limits` QoS policy, to be exceeded. In case the synchronous attribute value of the reliability QoS policy is set to true for communicating writers and readers then the writer will wait until all synchronous readers have acknowledged the data. Under these circumstances, the `max_blocking_time` attribute of the reliability QoS policy configures the maximum time the `dds_writedispose` operation may block. If `max_blocking_time` elapses before the writer is able to store the modification without exceeding the limits and all expected acknowledgements are received, the `dds_writedispose` operation will fail and returns `DDS_RETCODE_TIMEOUT`.

This operation performs the same functions as `dds_writedispose` except that the application provides the value for the `source_timestamp` that is made available to connected reader objects. This timestamp is important for the interpretation of the `destination_order` QoS policy.

Return A `dds_return_t` indicating success or failure.

Parameters

- `writer`: The writer to dispose the data instance from.
- `data`: The data to be written and disposed.

Return Value

- `DDS_RETCODE_OK`: The sample is written and the instance is marked for deletion.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: At least one of the arguments is invalid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_TIMEOUT`: Either the current action overflowed the available resources as specified by the combination of the reliability QoS policy, history QoS policy and `resource_limits` QoS policy, or the current action was waiting for data delivery acknowledgement by synchronous readers. This caused blocking of this operation, which could not be resolved before `max_blocking_time` of the reliability QoS policy elapsed.

This operation requests the Data Distribution Service to modify the instance and mark it for deletion. Copies of the instance and its corresponding samples, which are stored in every connected reader and, dependent on the QoS policy settings (also in the Transient and Persistent stores) will be modified and marked for deletion by setting their `dds_instance_state_t` to `DDS_IST_NOT_ALIVE_DISPOSED`.

Return A `dds_return_t` indicating success or failure.

Parameters

- `writer`: The writer to dispose the data instance from.
- `data`: The data to be written and disposed.
- `timestamp`: The timestamp used as source timestamp.

Return Value

- `DDS_RETCODE_OK`: The sample is written and the instance is marked for deletion.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: At least one of the arguments is invalid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.

- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_TIMEOUT`: Either the current action overflowed the available resources as specified by the combination of the reliability QoS policy, history QoS policy and resource_limits QoS policy, or the current action was waiting for data delivery acknowledgement by synchronous readers. This caused blocking of this operation, which could not be resolved before `max_blocking_time` of the reliability QoS policy elapsed.

If the history QoS policy is set to `DDS_HISTORY_KEEP_ALL`, the `dds_writedispose` operation on the writer may block if the modification would cause data to be lost because one of the limits, specified in the resource_limits QoS policy, to be exceeded. In case the synchronous attribute value of the reliability QoS policy is set to true for communicating writers and readers then the writer will wait until all synchronous readers have acknowledged the data. Under these circumstances, the `max_blocking_time` attribute of the reliability QoS policy configures the maximum time the `dds_writedispose` operation may block. If `max_blocking_time` elapses before the writer is able to store the modification without exceeding the limits and all expected acknowledgements are received, the `dds_writedispose` operation will fail and returns `DDS_RETCODE_TIMEOUT`.

This operation performs the same functions as `dds_dispose` except that the application provides the value for the source_timestamp that is made available to connected reader objects. This timestamp is important for the interpretation of the destination_order QoS policy.

Return A `dds_return_t` indicating success or failure.

Parameters

- `writer`: The writer to dispose the data instance from.
- `data`: The data sample that identifies the instance to be disposed.

Return Value

- `DDS_RETCODE_OK`: The sample is written and the instance is marked for deletion.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: At least one of the arguments is invalid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_TIMEOUT`: Either the current action overflowed the available resources as specified by the combination of the reliability QoS policy, history QoS policy and resource_limits QoS policy, or the current action was waiting for data delivery acknowledgement by synchronous readers. This caused blocking of this operation, which could not be resolved before `max_blocking_time` of the reliability QoS policy elapsed.

This operation requests the Data Distribution Service to modify the instance and mark it for deletion. Copies of the instance and its corresponding samples, which are stored in every connected reader and, dependent on the QoS policy settings (also in the Transient and Persistent stores) will be modified and marked for deletion by setting their `dds_instance_state_t` to `DDS_IST_NOT_ALIVE_DISPOSED`.

Return A `dds_return_t` indicating success or failure.

Parameters

- `writer`: The writer to dispose the data instance from.
- `data`: The data sample that identifies the instance to be disposed.
- `timestamp`: The timestamp used as source timestamp.

Return Value

- `DDS_RETCODE_OK`: The sample is written and the instance is marked for deletion
- `DDS_RETCODE_ERROR`: An internal error has occurred
- `DDS_RETCODE_BAD_PARAMETER`: At least one of the arguments is invalid
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted
- `DDS_RETCODE_TIMEOUT`: Either the current action overflowed the available resources as specified by the combination of the reliability QoS policy, history QoS policy and resource_limits QoS policy, or the current action was waiting for data delivery acknowledgment by synchronous readers. This caused blocking of this operation, which could not be resolved before `max_blocking_time` of the reliability QoS policy elapsed.

The given instance handle must correspond to the value that was returned by either the `dds_register_instance` operation, `dds_register_instance_ts` or `dds_instance_lookup`. If there is no correspondence, then the result of the operation is unspecified.

This operation performs the same functions as `dds_dispose_ih` except that the application provides the value for the `source_timestamp` that is made available to connected reader objects. This timestamp is important for the interpretation of the `destination_order` QoS policy.

Return A `dds_return_t` indicating success or failure.

Parameters

- `writer`: The writer to dispose the data instance from.
- `handle`: The handle to identify an instance.

Return Value

- `DDS_RETCODE_OK`: The sample is written and the instance is marked for deletion.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: At least one of the arguments is invalid
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this writer

With this API, the value of the source timestamp is automatically made available to the data reader by the service.

Return A `dds_return_t` indicating success or failure.

Parameters

- `writer`: The writer to dispose the data instance from.
- `handle`: The handle to identify an instance.
- `timestamp`: The timestamp used as source timestamp.

Return Value

- `DDS_RETCODE_OK`: The sample is written and the instance is marked for deletion.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: At least one of the arguments is invalid.

- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this writer.

Description : Unregisters an instance with a key value from the data writer. Instance can be identified either from data sample or from instance handle (at least one must be provided).

Return `dds_return_t` indicating success or failure.

Return A `dds_return_t` indicating success or failure.

Return A `dds_return_t` indicating success or failure.

Parameters

- `writer`: The writer entity.
- `data`: Value to be written.

Parameters

- `writer`: The writer entity.
- `cdr`: CDR serialized value to be written.
- `size`: Size (in bytes) of CDR encoded data to be written.

Parameters

- `writer`: The writer entity.
- `data`: Value to be written.
- `timestamp`: Source timestamp.

Arguments :

1. `wr` The writer to which instance is associated
2. `data` Instance with the key value (can be NULL if handle set)
3. `handle` Instance handle (can be `DDS_HANDLE_NIL` if data set)
4. Returns 0 on success, or non-zero value to indicate an error

Note : If an unregistered key ID is passed as instance data, an error is logged and not flagged as return value

Description : Unregisters an instance with a key value from the data writer. Instance can be identified either from data sample or from instance handle (at least one must be provided).

Arguments :

1. `wr` The writer to which instance is associated
2. `data` Instance with the key value (can be NULL if handle set)
3. `handle` Instance handle (can be `DDS_HANDLE_NIL` if data set)
4. `timestamp` used at registration.
5. Returns 0 on success, or non-zero value to indicate an error

Note : If an unregistered key ID is passed as instance data, an error is logged and not flagged as return value

This operation requests the Data Distribution Service to modify the instance and mark it for deletion. Copies of the instance and its corresponding samples, which are stored in every connected reader and, dependent on the QoS policy settings (also in the Transient and Persistent stores) will be modified and marked for deletion by setting their `dds_instance_state_t` to `DDS_IST_NOT_ALIVE_DISPOSED`.

If the history QoS policy is set to `DDS_HISTORY_KEEP_ALL`, the `dds_writedispose` operation on the writer may block if the modification would cause data to be lost because one of the limits, specified in the `resource_limits` QoS policy, to be exceeded. In case the synchronous attribute value of the reliability QoS policy is set to true for communicating writers and readers then the writer will wait until all synchronous readers have acknowledged the data. Under these circumstances, the `max_blocking_time` attribute of the reliability QoS policy configures the maximum time the `dds_writedispose` operation may block. If `max_blocking_time` elapses before the writer is able to store the modification without exceeding the limits and all expected acknowledgements are received, the `dds_writedispose` operation will fail and returns `DDS_RETCODE_TIMEOUT`.

Description : This operation modifies and disposes a data instance with a specific timestamp.

Return 0 - Success.

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `writer`: The writer to dispose the data instance from.
- `data`: The data to be written and disposed.

Return Value

- `DDS_RETCODE_OK`: The sample is written and the instance is marked for deletion.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: At least one of the arguments is invalid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_TIMEOUT`: Either the current action overflowed the available resources as specified by the combination of the reliability QoS policy, history QoS policy and `resource_limits` QoS policy, or the current action was waiting for data delivery acknowledgement by synchronous readers. This caused blocking of this operation, which could not be resolved before `max_blocking_time` of the reliability QoS policy elapsed.

This operation performs the same functions as `dds_writedispose` except that the application provides the value for the `source_timestamp` that is made available to connected reader objects. This timestamp is important for the interpretation of the `destination_order` QoS policy.

This operation requests the Data Distribution Service to modify the instance and mark it for deletion. Copies of the instance and its corresponding samples, which are stored in every connected reader and, dependent on the QoS policy settings (also in the Transient and Persistent stores) will be modified and marked for deletion by setting their `dds_instance_state_t` to `DDS_IST_NOT_ALIVE_DISPOSED`.

Return 0 - Success.

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `writer`: The writer to dispose the data instance from.
- `data`: The data to be written and disposed.
- `timestamp`: The timestamp used as source timestamp.

Return Value

- `DDS_RETCODE_OK`: The sample is written and the instance is marked for deletion.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: At least one of the arguments is invalid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_TIMEOUT`: Either the current action overflowed the available resources as specified by the combination of the reliability QoS policy, history QoS policy and resource_limits QoS policy, or the current action was waiting for data delivery acknowledgement by synchronous readers. This caused blocking of this operation, which could not be resolved before `max_blocking_time` of the reliability QoS policy elapsed.

If the history QoS policy is set to `DDS_HISTORY_KEEP_ALL`, the `dds_writedispose` operation on the writer may block if the modification would cause data to be lost because one of the limits, specified in the resource_limits QoS policy, to be exceeded. In case the synchronous attribute value of the reliability QoS policy is set to true for communicating writers and readers then the writer will wait until all synchronous readers have acknowledged the data. Under these circumstances, the `max_blocking_time` attribute of the reliability QoS policy configures the maximum time the `dds_writedispose` operation may block. If `max_blocking_time` elapses before the writer is able to store the modification without exceeding the limits and all expected acknowledgements are received, the `dds_writedispose` operation will fail and returns `DDS_RETCODE_TIMEOUT`.

Description : This operation disposes an instance with a specific timestamp, identified by the data sample.

Return 0 - Success.

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `writer`: The writer to dispose the data instance from.
- `data`: The data sample that identifies the instance to be disposed.

Return Value

- `DDS_RETCODE_OK`: The sample is written and the instance is marked for deletion.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: At least one of the arguments is invalid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_TIMEOUT`: Either the current action overflowed the available resources as specified by the combination of the reliability QoS policy, history QoS policy and resource_limits QoS policy, or the current action was waiting for data delivery acknowledgement by synchronous readers. This caused blocking of this operation, which could not be resolved before `max_blocking_time` of the reliability QoS policy elapsed.

This operation performs the same functions as `dds_dispose` except that the application provides the value for the `source_timestamp` that is made available to connected reader objects. This timestamp is important for the interpretation of the `destination_order` QoS policy.

This operation requests the Data Distribution Service to modify the instance and mark it for deletion. Copies of the instance and its corresponding samples, which are stored in every connected reader and,

dependent on the QoS policy settings (also in the Transient and Persistent stores) will be modified and marked for deletion by setting their `dds_instance_state_t` to `DDS_IST_NOT_ALIVE_DISPOSED`.

Return 0 - Success.

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- `writer`: The writer to dispose the data instance from.
- `data`: The data sample that identifies the instance to be disposed.
- `timestamp`: The timestamp used as source timestamp.

Return Value

- `DDS_RETCODE_OK`: The sample is written and the instance is marked for deletion.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: At least one of the arguments is invalid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_TIMEOUT`: Either the current action overflowed the available resources as specified by the combination of the reliability QoS policy, history QoS policy and resource_limits QoS policy, or the current action was waiting for data delivery acknowledgement by synchronous readers. This caused blocking of this operation, which could not be resolved before `max_blocking_time` of the reliability QoS policy elapsed.

The given instance handle must correspond to the value that was returned by either the `dds_register_instance` operation, `dds_register_instance_ts` or `dds_instance_lookup`. If there is no correspondence, then the result of the operation is unspecified.

Description : This operation disposes an instance with a specific timestamp, identified by the instance handle.

Return 0 - Success.

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- `writer`: The writer to dispose the data instance from.
- `handle`: The handle to identify an instance.

Return Value

- `DDS_RETCODE_OK`: The sample is written and the instance is marked for deletion.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: At least one of the arguments is invalid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this writer.

This operation performs the same functions as `dds_dispose_ih` except that the application provides the value for the `source_timestamp` that is made available to connected reader objects. This timestamp is important for the interpretation of the `destination_order` QoS policy.

With this API, the value of the source timestamp is automatically made available to the data reader by the service.

Return 0 - Success.

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- **writer:** The writer to dispose the data instance from.
- **handle:** The handle to identify an instance.
- **timestamp:** The timestamp used as source timestamp.

Return Value

- **DDS_RETCODE_OK:** The sample is written and the instance is marked for deletion.
- **DDS_RETCODE_ERROR:** An internal error has occurred.
- **DDS_RETCODE_BAD_PARAMETER:** At least one of the arguments is invalid.
- **DDS_RETCODE_ILLEGAL_OPERATION:** The operation is invoked on an inappropriate object.
- **DDS_RETCODE_ALREADY_DELETED:** The entity has already been deleted.
- **DDS_RETCODE_PRECONDITION_NOT_MET:** The instance handle has not been registered with this writer.

Untyped API, which take serialized blobs now. Whether they remain exposed like this with X-types isn't entirely clear yet. TODO: make a decide about dds_takecdr

Return - dds_return_t indicating success or failure

Parameters

- **writer:** The writer entity
- **data:** Value to be written

Return - A dds_return_t indicating success or failure

Return - A dds_return_t indicating success or failure

Parameters

- **writer:** The writer entity
- **cdr:** CDR serialized value to be written
- **size:** Size (in bytes) of CDR encoded data to be written

Parameters

- **writer:** The writer entity
- **data:** Value to be written
- **timestamp:** Source timestamp

_Pre_satisfies_((waitset &(0x7F000000)) = DDS_KIND_WAITSET)

Acquire previously attached entities.

This operation allows an application thread to wait for the a status change or other trigger on (one of) the entities that are attached to the WaitSet.

Sets the trigger_value associated with a waitset.

This operation detaches an Entity to the WaitSet.

This operation attaches an Entity to the WaitSet.

This functions takes a pre-allocated list to put the entities in and will return the number of found entities. It is possible that the given size of the list is not the same as the number of found entities. If less entities are found, then the last few entries in the list are untouched. When more entities are found, then only 'size' number of entries are inserted into the list, but still the complete count of the found entities is returned. Which entities are returned in the latter case is undefined.

This operation attaches an Entity to the WaitSet. The `dds_waitset_wait()` will block when none of the attached entities are triggered. 'Triggered' (`dds_triggered()`) doesn't mean the same for every entity:

- Reader/Writer/Publisher/Subscriber/Topic/Participant
 - These are triggered when their status changed.
- WaitSet
 - Triggered when trigger value was set to true by the application. It stays triggered until application sets the trigger value to false (`dds_waitset_set_trigger()`). This can be used to wake up an waitset for different reasons (f.i. termination) than the 'normal' status change (like new data).
- ReadCondition/QueryCondition
 - Triggered when data is available on the related Reader that matches the Condition.

Return A `dds_return_t` with the number of children or an error code.

Parameters

- `waitset`: Waitset from which to get its attached entities.
- `entities`: Pre-allocated array to contain the found entities.
- `size`: Size of the pre-allocated entities' list.

Return Value

- `>=0`: Number of children found (can be larger than 'size').
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The entities parameter is NULL, while a size is provided.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The waitset has already been deleted.

Multiple entities can be attached to a single waitset. A particular entity can be attached to multiple waitsets. However, a particular entity can not be attached to a particular waitset multiple times.

When the waitset is attached to itself and the trigger value is set to 'true', then the waitset will wake up just like with an other status change of the attached entities.

Return A `dds_return_t` indicating success or failure.

Return A `dds_return_t` indicating success or failure.

Parameters

- `waitset`: The waitset to attach the given entity to.
- `entity`: The entity to attach.
- `x`: Blob that will be supplied when the waitset wait is triggered by the given entity.

Return Value

- DDS_RETCODE_OK: Entity attached.
- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER: The given waitset or entity are not valid.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The waitset has already been deleted.
- DDS_RETCODE_PRECONDITION_NOT_MET: The entity was already attached.

Parameters

- waitset: The waitset to detach the given entity from.
- entity: The entity to detach.

Return Value

- DDS_RETCODE_OK: Entity attached.
- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER: The given waitset or entity are not valid.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The waitset has already been deleted.
- DDS_RETCODE_PRECONDITION_NOT_MET: The entity is not attached.

This can be used to forcefully wake up a waitset, for instance when the application wants to shut down. So, when the trigger value is true, the waitset will wake up or not wait at all.

The trigger value will remain true until the application sets it false again deliberately.

The “dds_waitset_wait” operation blocks until the some of the attached entities have triggered or “retime-out” has elapsed. ‘Triggered’ (dds_triggered()) doesn’t mean the same for every entity:

- Reader/Writer/Publisher/Subscriber/Topic/Participant
 - These are triggered when their status changed.
- WaitSet
 - Triggered when trigger value was set to true by the application. It stays triggered until application sets the trigger value to false (dds_waitset_set_trigger()). This can be used to wake up an waitset for different reasons (f.i. termination) than the ‘normal’ status change (like new data).
- ReadCondition/QueryCondition
 - Triggered when data is available on the related Reader that matches the Condition.

Return A dds_return_t indicating success or failure.

Parameters

- waitset: The waitset to set the trigger value on.
- trigger: The trigger value to set.

Return Value

- DDS_RETCODE_OK: Entity attached.
- DDS_RETCODE_ERROR: An internal error has occurred.

- `DDS_RETCODE_BAD_PARAMETER`: The given waitset is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The waitset has already been deleted.

This function takes a pre-allocated list to put the “xs” blobs in (that were provided during the attach of the related entities) and will return the number of triggered entities. It is possible that the given size of the list is not the same as the number of triggered entities. If less entities were triggered, then the last few entries in the list are untouched. When more entities are triggered, then only ‘size’ number of entries are inserted into the list, but still the complete count of the triggered entities is returned. Which “xs” blobs are returned in the latter case is undefined.

In case of a time out, the return value is 0.

Deleting the waitset while the application is blocked results in an error code (i.e. < 0) returned by “wait”.

Multiple threads may block on a single waitset at the same time; the calls are entirely independent.

An empty waitset never triggers (i.e., `dds_waitset_wait` on an empty waitset is essentially equivalent to a sleep).

The “`dds_waitset_wait_until`” operation is the same as the “`dds_waitset_wait`” except that it takes an absolute timeout.

The “`dds_waitset_wait`” operation blocks until the some of the attached entities have triggered or “abstime-out” has been reached. ‘Triggered’ (`dds_triggered()`) doesn’t mean the same for every entity:

- **Reader/Writer/Publisher/Subscriber/Topic/Participant**
 - These are triggered when their status changed.
- **WaitSet**
 - Triggered when trigger value was set to true by the application. It stays triggered until application sets the trigger value to false (`dds_waitset_set_trigger()`). This can be used to wake up an waitset for different reasons (f.i. termination) than the ‘normal’ status change (like new data).
- **ReadCondition/QueryCondition**
 - Triggered when data is available on the related Reader that matches the Condition.

Return A `dds_return_t` with the number of entities triggered or an error code

Parameters

- `waitset`: The waitset to set the trigger value on.
- `xs`: Pre-allocated list to store the ‘blobs’ that were provided during the attach of the triggered entities.
- `nxs`: The size of the pre-allocated blobs list.
- `reltimeout`: Relative timeout

Return Value

- >0: Number of entities triggered.
- 0: Time out (no entities were triggered).
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The given waitset is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.

- `DDS_RETCODE_ALREADY_DELETED`: The waitset has already been deleted.

This function takes a pre-allocated list to put the “xs” blobs in (that were provided during the attach of the related entities) and will return the number of triggered entities. It is possible that the given size of the list is not the same as the number of triggered entities. If less entities were triggered, then the last few entries in the list are untouched. When more entities are triggered, then only ‘size’ number of entries are inserted into the list, but still the complete count of the triggered entities is returned. Which “xs” blobs are returned in the latter case is undefined.

In case of a time out, the return value is 0.

Deleting the waitset while the application is blocked results in an error code (i.e. < 0) returned by “wait”.

Multiple threads may block on a single waitset at the same time; the calls are entirely independent.

An empty waitset never triggers (i.e., `dds_waitset_wait` on an empty waitset is essentially equivalent to a sleep).

The “`dds_waitset_wait`” operation is the same as the “`dds_waitset_wait_until`” except that it takes an relative timeout.

The “`dds_waitset_wait`” operation is the same as the “`dds_wait`” except that it takes an absolute timeout.

This operation attaches an Entity to the WaitSet. The `dds_waitset_wait()` will block when none of the attached entities are triggered. ‘Triggered’ (`dds_triggered()`) doesn’t mean the same for every entity:

- Reader/Writer/Publisher/Subscriber/Topic/Participant
 - These are triggered when their status changed.
- WaitSet
 - Triggered when trigger value was set to true by the application. It stays triggered until application sets the trigger value to false (`dds_waitset_set_trigger()`). This can be used to wake up an waitset for different reasons (f.i. termination) than the ‘normal’ status change (like new data).
- ReadCondition/QueryCondition
 - Triggered when data is available on the related Reader that matches the Condition.

Return A `dds_return_t` with the number of entities triggered or an error code.

Parameters

- `waitset`: The waitset to set the trigger value on.
- `xs`: Pre-allocated list to store the ‘blobs’ that were provided during the attach of the triggered entities.
- `nxs`: The size of the pre-allocated blobs list.
- `abstimeout`: Absolute timeout

Return Value

- > 0 : Number of entities triggered.
- 0: Time out (no entities were triggered).
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The given waitset is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The waitset has already been deleted.

Multiple entities can be attached to a single waitset. A particular entity can be attached to multiple waitsets. However, a particular entity can not be attached to a particular waitset multiple times.

When the waitset is attached to itself and the trigger value is set to 'true', then the waitset will wake up just like with an other status change of the attached entities.

Return 0 - Success (entity attached).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Return 0 - Success (entity attached).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *waitset*: The waitset to attach the given entity to.
- *entity*: The entity to attach.
- *x*: Blob that will be supplied when the waitset wait is triggered by the given entity.

Return Value

- *DDS_RETCODE_ERROR*: An internal error has occurred.
- *DDS_RETCODE_BAD_PARAMETER*: The given waitset or entity are not valid.
- *DDS_RETCODE_ILLEGAL_OPERATION*: The operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED*: The waitset has already been deleted.
- *DDS_RETCODE_PRECONDITION_NOT_MET*: The entity was already attached.

Parameters

- *waitset*: The waitset to detach the given entity from.
- *entity*: The entity to detach.

Return Value

- *DDS_RETCODE_ERROR*: An internal error has occurred.
- *DDS_RETCODE_BAD_PARAMETER*: The given waitset or entity are not valid.
- *DDS_RETCODE_ILLEGAL_OPERATION*: The operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED*: The waitset has already been deleted.
- *DDS_RETCODE_PRECONDITION_NOT_MET*: The entity is not attached.

This can be used to forcefully wake up a waitset, for instance when the application wants to shut down. So, when the trigger value is true, the waitset will wake up or not wait at all.

The trigger value will remain true until the application sets it false again deliberately.

The "dds_waitset_wait" operation blocks until the some of the attached entities have triggered or "retime-out" has elapsed. 'Triggered' (*dds_triggered()*) doesn't mean the same for every entity:

- Reader/Writer/Publisher/Subscriber/Topic/Participant
 - These are triggered when their status changed.
- WaitSet
 - Triggered when trigger value was set to true by the application. It stays triggered until application sets the trigger value to false (*dds_waitset_set_trigger()*). This can be used to wake up an waitset for different reasons (f.i. termination) than the 'normal' status change (like new data).

- ReadCondition/QueryCondition
 - Triggered when data is available on the related Reader that matches the Condition.

Return 0 - Success (entity attached).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- waitset: The waitset to set the trigger value on.
- trigger: The trigger value to set.

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER: The given waitset is not valid.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The waitset has already been deleted.

This functions takes a pre-allocated list to put the “xs” blobs in (that were provided during the attach of the related entities) and will return the number of triggered entities. It is possible that the given size of the list is not the same as the number of triggered entities. If less entities were triggered, then the last few entries in the list are untouched. When more entities are triggered, then only ‘size’ number of entries are inserted into the list, but still the complete count of the triggered entities is returned. Which “xs” blobs are returned in the latter case is undefined.

In case of a time out, the return value is 0.

Deleting the waitset while the application is blocked results in an error code (i.e. < 0) returned by “wait”.

Multiple threads may block on a single waitset at the same time; the calls are entirely independent.

An empty waitset never triggers (i.e., *dds_waitset_wait* on an empty waitset is essentially equivalent to a sleep).

The “*dds_waitset_wait_until*” operation is the same as the “*dds_waitset_wait*” except that it takes an absolute timeout.

The “*dds_waitset_wait*” operation blocks until the some of the attached entities have triggered or “abstime-out” has been reached. ‘Triggered’ (*dds_triggered()*) doesn’t mean the same for every entity:

- Reader/Writer/Publisher/Subscriber/Topic/Participant
 - These are triggered when their status changed.
- WaitSet
 - Triggered when trigger value was set to true by the application. It stays triggered until application sets the trigger value to false (*dds_waitset_set_trigger()*). This can be used to wake up an waitset for different reasons (f.i. termination) than the ‘normal’ status change (like new data).
- ReadCondition/QueryCondition
 - Triggered when data is available on the related Reader that matches the Condition.

Return >0 - Success (number of entities triggered).

Return 0 - Time out (no entities were triggered).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- `waitset`: The waitset to set the trigger value on.
- `xs`: Pre-allocated list to store the ‘blobs’ that were provided during the attach of the triggered entities.
- `nxs`: The size of the pre-allocated blobs list.
- `reltimeout`: Relative timeout

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The given waitset is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The waitset has already been deleted.

This functions takes a pre-allocated list to put the “xs” blobs in (that were provided during the attach of the related entities) and will return the number of triggered entities. It is possible that the given size of the list is not the same as the number of triggered entities. If less entities were triggered, then the last few entries in the list are untouched. When more entities are triggered, then only ‘size’ number of entries are inserted into the list, but still the complete count of the triggered entities is returned. Which “xs” blobs are returned in the latter case is undefined.

In case of a time out, the return value is 0.

Deleting the waitset while the application is blocked results in an error code (i.e. < 0) returned by “wait”.

Multiple threads may block on a single waitset at the same time; the calls are entirely independent.

An empty waitset never triggers (i.e., `dds_waitset_wait` on an empty waitset is essentially equivalent to a sleep).

The “`dds_waitset_wait`” operation is the same as the “`dds_waitset_wait_until`” except that it takes an relative timeout.

The “`dds_waitset_wait`” operation is the same as the “`dds_wait`” except that it takes an absolute timeout.

Return >0 - Success (number of entities triggered).

Return 0 - Time out (no entities were triggered).

Return <0 - Failure (use *`dds_err_nr()`* to get error value).

Parameters

- `waitset`: The waitset to set the trigger value on.
- `xs`: Pre-allocated list to store the ‘blobs’ that were provided during the attach of the triggered entities.
- `nxs`: The size of the pre-allocated blobs list.
- `abstimeout`: Absolute timeout

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The given waitset is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.

- `DDS_RETCODE_ALREADY_DELETED`: The waitset has already been deleted.

`_Out_writes_to_` (size)

`_Out_writes_to_opt_` (nxs)

`_Pre_satisfies_(((reader_or_condition &(0x7F000000))==DDS_KIND_READER) || ((reader_or_co`

Access and read the collection of data values (of same type) and sample info from the data reader, readcondition or querycondition.

Return loaned samples to data-reader or condition associated with a data-reader.

Access loaned samples of data reader, readcondition or querycondition based on mask and scoped by the given instance handle.

Take the collection of data values (of same type) and sample info from the data reader, readcondition or querycondition based on mask and scoped by the given instance handle.

Access loaned samples of data reader, readcondition or querycondition, scoped by the given instance handle.

Access the collection of data values (of same type) and sample info from the data reader, readcondition or querycondition but scoped by the given instance handle.

Access loaned samples of data reader, readcondition or querycondition based on mask.

Take the collection of data values (of same type) and sample info from the data reader, readcondition or querycondition based on mask.

Access loaned samples of data reader, readcondition or querycondition.

Access the collection of data values (of same type) and sample info from the data reader, readcondition or querycondition.

Access and read loaned samples of data reader, readcondition or querycondition based on mask, scoped by the provided instance handle.

Read the collection of data values and sample info from the data reader, readcondition or querycondition based on mask and scoped by the provided instance handle.

Access and read loaned samples of data reader, readcondition or querycondition, scoped by the provided instance handle.

Access and read the collection of data values (of same type) and sample info from the data reader, readcondition or querycondition, copied by the provided instance handle.

Access and read loaned samples of data reader, readcondition or querycondition based on mask.

Read the collection of data values and sample info from the data reader, readcondition or querycondition based on mask.

Access and read loaned samples of data reader, readcondition or querycondition.

Return value provides information about number of samples read, which will be \leq maxs. Based on the count, the buffer will contain data to be read only when `valid_data` bit in sample info structure is set. The buffer required for data values, could be allocated explicitly or can use the memory from data reader to prevent copy. In the latter case, buffer and `sample_info` should be returned back, once it is no longer using the Data. Data values once read will remain in the buffer with the `sample_state` set to `READ` and `view_state` set to `NOT_NEW`.

After `dds_read_wl` function is being called and the data has been handled, `dds_return_loan` function must be called to possibly free memory.

Return A `dds_return_t` with the number of samples read or an error code.

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity.
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL).
- `si`: Pointer to an array of *dds_sample_info_t* returned for each data value.
- `bufsz`: The size of buffer provided.
- `maxs`: Maximum number of samples to read.

Return Value

- `>=0`: Number of samples read.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

When using a readcondition or querycondition, their masks are or'd with the given mask.

Return A `dds_return_t` with the number of samples read or an error code

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of *dds_sample_info_t* returned for each data value
- `maxs`: Maximum number of samples to read

Return Value

- `>=0`: Number of samples read.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

When using a readcondition or querycondition, their masks are or'd with the given mask.

Return A `dds_return_t` with the number of samples read or an error code.

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity.
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL).
- `si`: Pointer to an array of *dds_sample_info_t* returned for each data value.
- `bufsz`: The size of buffer provided.
- `maxs`: Maximum number of samples to read.
- `mask`: Filter the data based on `dds_sample_state_t|dds_view_state_t|dds_instance_state_t`.

Return Value

- `>=0`: Number of samples read.

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

After `dds_read_mask_wl` function is being called and the data has been handled, `dds_return_loan` function must be called to possibly free memory

This operation implements the same functionality as `dds_read`, except that only data scoped to the provided instance handle is read.

Return A `dds_return_t` with the number of samples read or an error code.

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity.
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL).
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value.
- `maxs`: Maximum number of samples to read.
- `mask`: Filter the data based on `dds_sample_state_t`, `ldds_view_state_t`, `ldds_instance_state_t`.

Return Value

- `>=0`: Number of samples read.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation implements the same functionality as `dds_read_wl`, except that only data scoped to the provided instance handle is read.

Return A `dds_return_t` with the number of samples read or an error code.

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity.
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL).
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value.
- `bufsz`: The size of buffer provided.
- `maxs`: Maximum number of samples to read.
- `handle`: Instance handle related to the samples to read.

Return Value

- `>=0`: Number of samples read.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

This operation implements the same functionality as `dds_read_mask`, except that only data scoped to the provided instance handle is read.

Return A `dds_return_t` with the number of samples read or an error code.

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity.
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL).
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value.
- `maxs`: Maximum number of samples to read.
- `handle`: Instance handle related to the samples to read.

Return Value

- `>=0`: Number of samples read.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

This operation implements the same functionality as `dds_read_mask_wl`, except that only data scoped to the provided instance handle is read.

Return A `dds_return_t` with the number of samples read or an error code.

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity.
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL).
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value.
- `bufsz`: The size of buffer provided.
- `maxs`: Maximum number of samples to read.
- `handle`: Instance handle related to the samples to read.
- `mask`: Filter the data based on `dds_sample_state_t`, `ldds_view_state_t`, `ldds_instance_state_t`.

Return Value

- `>=0`: Number of samples read.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

Data value once read is removed from the Data Reader cannot to 'read' or 'taken' again. Return value provides information about number of samples read, which will be \leq maxs. Based on the count, the buffer will contain data to be read only when valid_data bit in sample info structure is set. The buffer required for data values, could be allocated explicitly or can use the memory from data reader to prevent copy. In the latter case, buffer and sample_info should be returned back, once it is no longer using the Data.

Return A dds_return_t with the number of samples read or an error code.

Parameters

- reader_or_condition: Reader, readcondition or querycondition entity.
- buf: An array of pointers to samples into which data is read (pointers can be NULL).
- si: Pointer to an array of *dds_sample_info_t* returned for each data value.
- maxs: Maximum number of samples to read.
- handle: Instance handle related to the samples to read.
- mask: Filter the data based on dds_sample_state_t ldds_view_state_t ldds_instance_state_t.

Return Value

- ≥ 0 : Number of samples read.
- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER: One of the given arguments is not valid.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.
- DDS_RETCODE_PRECONDITION_NOT_MET: The instance handle has not been registered with this reader.

After dds_take_wl function is being called and the data has been handled, dds_return_loan function must be called to possibly free memory

Return A dds_return_t with the number of samples read or an error code.

Parameters

- reader_or_condition: Reader, readcondition or querycondition entity.
- buf: An array of pointers to samples into which data is read (pointers can be NULL).
- si: Pointer to an array of *dds_sample_info_t* returned for each data value.
- bufsz: The size of buffer provided.
- maxs: Maximum number of samples to read.

Return Value

- ≥ 0 : Number of samples read.
- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER: One of the given arguments is not valid.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

When using a readcondition or querycondition, their masks are or'd with the given mask.

Return A `dds_return_t` with the number of samples read or an error code.

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity.
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL).
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value.
- `maxs`: Maximum number of samples to read.

Return Value

- `>=0`: Number of samples read.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

After `dds_take_mask_wl` function is being called and the data has been handled, `dds_return_loan` function must be called to possibly free memory

This operation implements the same functionality as `dds_take`, except that only data scoped to the provided instance handle is taken.

Return A `dds_return_t` with the number of samples read or an error code.

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity.
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL).
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value.
- `maxs`: Maximum number of samples to read.
- `mask`: Filter the data based on `dds_sample_state_t`, `ldds_view_state_t`, `ldds_instance_state_t`.

Return Value

- `>=0`: Number of samples read.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation implements the same functionality as `dds_take_wl`, except that only data scoped to the provided instance handle is read.

Return A `dds_return_t` with the number of samples read or an error code.

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity.
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL).
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value.
- `bufsz`: The size of buffer provided.

- `maxs`: Maximum number of samples to read.
- `handle`: Instance handle related to the samples to read.

Return Value

- `>=0`: Number of samples read.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

This operation implements the same functionality as `dds_take_mask`, except that only data scoped to the provided instance handle is read.

Return A `dds_return_t` with the number of samples read or an error code.

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity.
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL).
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value.
- `maxs`: Maximum number of samples to read.
- `handle`: Instance handle related to the samples to read.

Return Value

- `>=0`: Number of samples read.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

This operation implements the same functionality as `dds_take_mask_wl`, except that only data scoped to the provided instance handle is read.

Return A `dds_return_t` with the number of samples read or an error code.

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity.
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL).
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value.
- `bufsz`: The size of buffer provided.
- `maxs`: Maximum number of samples to read.
- `handle`: Instance handle related to the samples to read.

- `mask`: Filter the data based on `dds_sample_state_t`, `dds_view_state_t`, `dds_instance_state_t`.

Return Value

- `>=0`: Number of samples read.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

Used to release sample buffers returned by a read/take operation. When the application provides an empty buffer, memory is allocated and managed by DDS. By calling `dds_return_loan`, the memory is released so that the buffer can be reused during a successive read/take operation. When a condition is provided, the reader to which the condition belongs is looked up.

Return A `dds_return_t` with the number of samples or an error code.

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity.
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL).
- `si`: Pointer to an array of *`dds_sample_info_t`* returned for each data value.
- `maxs`: Maximum number of samples to read.
- `handle`: Instance handle related to the samples to read.
- `mask`: Filter the data based on `dds_sample_state_t`, `dds_view_state_t`, `dds_instance_state_t`.

Return Value

- `>=0`: Number of samples read.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

After `dds_read_wl` function is being called and the data has been handled, `dds_return_loan` function must be called to possibly free memory.

Return A `dds_return_t` indicating success or failure.

Parameters

- `rd_or_cnd`: Reader or condition that belongs to a reader.
- `buf`: An array of (pointers to) samples.
- `bufsz`: The number of (pointers to) samples stored in `buf`.

When using a readcondition or querycondition, their masks are or'd with the given mask.

Return `>=0` - Success (number of samples read).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *reader_or_condition*: Reader, readcondition or querycondition entity
- *buf*: An array of pointers to samples into which data is read (pointers can be NULL)
- *si*: Pointer to an array of *dds_sample_info_t* returned for each data value
- *maxs*: Maximum number of samples to read

Return Value

- *DDS_RETCODE_ERROR*: An internal error has occurred.
- *DDS_RETCODE_BAD_PARAMETER*: One of the given arguments is not valid.
- *DDS_RETCODE_ILLEGAL_OPERATION*: The operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED*: The entity has already been deleted.

When using a readcondition or querycondition, their masks are or'd with the given mask.

Return >=0 - Success (number of samples read).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *reader_or_condition*: Reader, readcondition or querycondition entity
- *buf*: An array of pointers to samples into which data is read (pointers can be NULL)
- *si*: Pointer to an array of *dds_sample_info_t* returned for each data value
- *bufsz*: The size of buffer provided
- *maxs*: Maximum number of samples to read
- *mask*: Filter the data based on *dds_sample_state_t*, *ldds_view_state_t*, *ldds_instance_state_t*.

Return Value

- *DDS_RETCODE_ERROR*: An internal error has occurred.
- *DDS_RETCODE_BAD_PARAMETER*: One of the given arguments is not valid.
- *DDS_RETCODE_ILLEGAL_OPERATION*: The operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED*: The entity has already been deleted.

After *dds_read_mask_wl* function is being called and the data has been handled, *dds_return_loan* function must be called to possibly free memory

This operation implements the same functionality as *dds_read*, except that only data scoped to the provided instance handle is read.

Return >=0 - Success (number of samples read).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *reader_or_condition*: Reader, readcondition or querycondition entity
- *buf*: An array of pointers to samples into which data is read (pointers can be NULL)
- *si*: Pointer to an array of *dds_sample_info_t* returned for each data value
- *maxs*: Maximum number of samples to read

- `mask`: Filter the data based on `dds_sample_state_t`, `dds_view_state_t`, `dds_instance_state_t`.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation implements the same functionality as `dds_read_wl`, except that only data scoped to the provided instance handle is read.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value
- `bufsz`: The size of buffer provided
- `maxs`: Maximum number of samples to read
- `handle`: Instance handle related to the samples to read

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

This operation implements the same functionality as `dds_read_mask`, except that only data scoped to the provided instance handle is read.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value
- `maxs`: Maximum number of samples to read
- `handle`: Instance handle related to the samples to read

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.

- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

This operation implements the same functionality as `dds_read_mask_wl`, except that only data scoped to the provided instance handle is read.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value
- `bufsz`: The size of buffer provided
- `maxs`: Maximum number of samples to read
- `handle`: Instance handle related to the samples to read
- `mask`: Filter the data based on `dds_sample_state_t`, `dds_view_state_t`, `dds_instance_state_t`.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

Description : Access the collection of data values (of same type) and sample info from the data reader based on the criteria specified in the read condition. Read condition must be attached to the data reader before associating with data read. Return value provides information about number of samples read, which will be \leq maxs. Based on the count, the buffer will contain data to be read only when `valid_data` bit in sample info structure is set. The buffer required for data values, could be allocated explicitly or can use the memory from data reader to prevent copy. In the latter case, buffer and sample_info should be returned back, once it is no longer using the Data. Data values once read will remain in the buffer with the `sample_state` set to `READ` and `view_state` set to `NOT_NEW`.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value
- `maxs`: Maximum number of samples to read
- `handle`: Instance handle related to the samples to read

- `mask`: Filter the data based on `dds_sample_state_t`, `dds_view_state_t`, `dds_instance_state_t`.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

Arguments :

1. `rd` Reader entity
2. `buf` an array of pointers to samples into which data is read (pointers can be NULL)
3. `maxs` maximum number of samples to read
4. `si` pointer to an array of `dds_sample_info_t` returned for each data value
5. `cond` read condition to filter the data samples based on the content
6. Returns the number of samples read, 0 indicates no data to read. Data value once read is removed from the Data Reader cannot to 'read' or 'taken' again. Return value provides information about number of samples read, which will be \leq `maxs`. Based on the count, the buffer will contain data to be read only when `valid_data` bit in sample info structure is set. The buffer required for data values, could be allocated explicitly or can use the memory from data reader to prevent copy. In the latter case, buffer and `sample_info` should be returned back, once it is no longer using the Data.

After `dds_take_wl` function is being called and the data has been handled, `dds_return_loan` function must be called to possibly free memory

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value
- `bufsz`: The size of buffer provided
- `maxs`: Maximum number of samples to read

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

When using a readcondition or querycondition, their masks are or'd with the given mask.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of *dds_sample_info_t* returned for each data value
- `maxs`: Maximum number of samples to read

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

After `dds_take_mask_wl` function is being called and the data has been handled, `dds_return_loan` function must be called to possibly free memory

This operation implements the same functionality as `dds_take`, except that only data scoped to the provided instance handle is taken.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of *dds_sample_info_t* returned for each data value
- `maxs`: Maximum number of samples to read
- `mask`: Filter the data based on `dds_sample_state_t``ldds_view_state_t``ldds_instance_state_t`.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation implements the same functionality as `dds_take_wl`, except that only data scoped to the provided instance handle is read.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of *dds_sample_info_t* returned for each data value
- `bufsz`: The size of buffer provided
- `maxs`: Maximum number of samples to read

- `handle`: Instance handle related to the samples to read

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

This operation implements the same functionality as `dds_take_mask`, except that only data scoped to the provided instance handle is read.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value
- `maxs`: Maximum number of samples to read
- `handle`: Instance handle related to the samples to read

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

This operation implements the same functionality as `dds_take_mask_wl`, except that only data scoped to the provided instance handle is read.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value
- `bufsz`: The size of buffer provided
- `maxs`: Maximum number of samples to read
- `handle`: Instance handle related to the samples to read
- `mask`: Filter the data based on `dds_sample_state_t`, `ldds_view_state_t`, `ldds_instance_state_t`.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

Used to release sample buffers returned by a read/take operation. When the application provides an empty buffer, memory is allocated and managed by DDS. By calling `dds_return_loan`, the memory is released so that the buffer can be reused during a successive read/take operation. When a condition is provided, the reader to which the condition belongs is looked up.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value
- `maxs`: Maximum number of samples to read
- `handle`: Instance handle related to the samples to read
- `mask`: Filter the data based on `dds_sample_state_t`, `ldds_view_state_t`, `ldds_instance_state_t`.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

Return A `dds_return_t` indicating success or failure

Parameters

- `rd_or_cnd`: Reader or condition that belongs to a reader
- `buf`: An array of (pointers to) samples
- `bufsz`: The number of (pointers to) samples stored in `buf`

```
DDS_EXPORT int dds_takecdr(dds_entity_t reader_or_condition, struct serdata ** buf, ui
_Inout_updates_ (bufsz)
```

```
_Pre_satisfies_(((entity & (0x7F000000)) == DDS_KIND_READER) || ((entity & (0x7F000000)) == DD
```

Begin coherent publishing or begin accessing a coherent set in a subscriber.

End coherent publishing or end accessing a coherent set in a subscriber.

Invoking on a Writer or Reader behaves as if `dds_begin_coherent` was invoked on its parent Publisher or Subscriber respectively.

Invoking on a Writer or Reader behaves as if `dds_end_coherent` was invoked on its parent Publisher or Subscriber respectively.

Return A `dds_return_t` indicating success or failure.

Parameters

- `entity`: The entity that is prepared for coherent access.

Return Value

- `DDS_RETCODE_OK`: The operation was successful.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The provided entity is invalid or not supported.

Invoking on a Writer or Reader behaves as if `dds_end_coherent` was invoked on its parent Publisher or Subscriber respectively.

Return A `dds_return_t` indicating success or failure.

Parameters

- `entity`: The entity on which coherent access is finished.

Return Value

- `DDS_RETCODE_OK`: The operation was successful.
- `DDS_RETCODE_BAD_PARAMETER`: The provided entity is invalid or not supported.

Return - A `dds_return_t` indicating success or failure

Parameters

- `e`: - The entity on which coherent access is finished

Return Value

- `DDS_RETCODE_OK`: The operation was successful
- `DDS_RETCODE_BAD_PARAMETER`: The provided entity is invalid or not supported

`_Pre_satisfies_((subscriber &(0x7F000000)) == DDS_KIND_SUBSCRIBER)`

Trigger `DATA_AVAILABLE` event on contained readers.

The `DATA_AVAILABLE` event is broadcast to all readers owned by this subscriber that currently have new data available. Any `on_data_available` listener callbacks attached to respective readers are invoked.

Return A `dds_return_t` indicating success or failure.

Parameters

- `subscriber`: A valid subscriber handle.

Return Value

- `DDS_RETCODE_OK`: The operation was successful.
- `DDS_RETCODE_BAD_PARAMETER`: The provided subscriber is invalid.

Variables

```
DDS_EXPORT const dds_entity_t DDS_BUILTIN_TOPIC_DCPSPARTICIPANT
DDS_EXPORT const dds_entity_t DDS_BUILTIN_TOPIC_CMPARTICIPANT
DDS_EXPORT const dds_entity_t DDS_BUILTIN_TOPIC_DCPSTYPE
DDS_EXPORT const dds_entity_t DDS_BUILTIN_TOPIC_DCPSTOPIC
DDS_EXPORT const dds_entity_t DDS_BUILTIN_TOPIC_DCPSPUBLICATION
DDS_EXPORT const dds_entity_t DDS_BUILTIN_TOPIC_CMPUBLISHER
DDS_EXPORT const dds_entity_t DDS_BUILTIN_TOPIC_DCPSSUBSCRIPTION
DDS_EXPORT const dds_entity_t DDS_BUILTIN_TOPIC_CMSUBSCRIBER
DDS_EXPORT const dds_entity_t DDS_BUILTIN_TOPIC_CMDATAWRITER
DDS_EXPORT const dds_entity_t DDS_BUILTIN_TOPIC_CMDATAREADER
_Out_ void _Out_ dds_sample_info_t _In_ uint32_t _In_ dds_instance_handle_t _In_ uint32_t
_Out_ dds_instance_handle_t* ihdl
_Out_ uint32_t * status
_In_opt_ const dds_qos_t * qos
_In_opt_ const dds_qos_t _In_opt_ const dds_listener_t * listener
_Out_opt_ dds_entity_t* children
_In_ size_t size
_Out_ dds_domainid_t* id
_In_ const dds_topic_descriptor_t* descriptor
_In_z_ const char * name
_In_ uint32_t _In_ dds_querycondition_filter_fn filter
_In_ dds_duration_t timeout
dds_duration_t max_wait
_Out_ void _Out_ dds_sample_info_t _In_ uint32_t _In_ dds_instance_handle_t handle
_In_ const void * data
_In_ const void _In_ dds_time_t timestamp
const void *cdr
_In_ dds_entity_t entity
_In_ dds_entity_t _In_ dds_attach_t x
_In_ bool trigger
_In_ size_t nxs
_In_ size_t _In_ dds_duration_t reltimeout
_In_ size_t _In_ dds_time_t abstimeout
_Out_ void ** buf
_Out_ void _Out_ dds_sample_info_t * si
```

```

_In_ size_t bufsz
_Out_ void _Out_ dds_sample_info_t _In_ uint32_t maxs
dds_instance_handle_t inst

```

file **dds_public_alloc.h**

#include "os/os_public.h" #include "ddsc/dds_export.h" DDS C Allocation API.

This header file defines the public API of allocation convenience functions in the VortexDDS C language binding.

Defines

```

DDS_FREE_KEY_BIT
DDS_FREE_CONTENTS_BIT
DDS_FREE_ALL_BIT

```

Typedefs

```

typedef struct dds_allocator dds_allocator_t
typedef struct dds_aligned_allocator dds_aligned_allocator_t
typedef void (*dds_alloc_fn_t)(size_t)
typedef void (*dds_realloc_fn_t)(void *, size_t)
typedef void (*dds_free_fn_t)(void *)

```

Enums

```

enum dds_free_op_t
    Values:
        DDS_FREE_ALL = 0x01 | 0x02 | 0x04
        DDS_FREE_CONTENTS = 0x01 | 0x02
        DDS_FREE_KEY = 0x01

```

Functions

```

DDS_EXPORT void dds_set_allocator(const dds_allocator_t * n, dds_allocator_t * o)
DDS_EXPORT void dds_set_aligned_allocator(const dds_aligned_allocator_t * n, dds_aligned_allocator_t * o)
DDS_EXPORT void* dds_alloc(size_t size)
DDS_EXPORT void* dds_realloc(void * ptr, size_t size)
DDS_EXPORT void* dds_realloc_zero(void * ptr, size_t size)
DDS_EXPORT void dds_free(void * ptr)
DDS_EXPORT char* dds_string_alloc(size_t size)
DDS_EXPORT char* dds_string_dup(const char * str)

```

```
DDS_EXPORT void dds_string_free(char * str)
```

```
DDS_EXPORT void dds_sample_free(void * sample, const struct dds_topic_descriptor * d
```

file `dds_public_error.h`

```
#include "os/os_public.h"#include "ddsc/dds_export.h" DDS C Error API.
```

This header file defines the public API of error values and convenience functions in the VortexDDS C language binding.

Return codes

DDS_RETCODE_OK

Success

DDS_RETCODE_ERROR

Non specific error

DDS_RETCODE_UNSUPPORTED

Feature unsupported

DDS_RETCODE_BAD_PARAMETER

Bad parameter value

DDS_RETCODE_PRECONDITION_NOT_MET

Precondition for operation not met

DDS_RETCODE_OUT_OF_RESOURCES

When an operation fails because of a lack of resources

DDS_RETCODE_NOT_ENABLED

When a configurable feature is not enabled

DDS_RETCODE_IMMUTABLE_POLICY

When an attempt is made to modify an immutable policy

DDS_RETCODE_INCONSISTENT_POLICY

When a policy is used with inconsistent values

DDS_RETCODE_ALREADY_DELETED

When an attempt is made to delete something more than once

DDS_RETCODE_TIMEOUT

When a timeout has occurred

DDS_RETCODE_NO_DATA

When expected data is not provided

DDS_RETCODE_ILLEGAL_OPERATION

When a function is called when it should not be

DDS_RETCODE_NOT_ALLOWED_BY_SECURITY

When credentials are not enough to use the function

DDS_Error_Type

DDS_CHECK_REPORT

DDS_CHECK_FAIL

DDS_CHECK_EXIT

Macros for error handling

DDS_TO_STRING (n)

DDS_INT_TO_STRING (n)

Defines

DDS_ERR_NR_MASK

DDS_ERR_LINE_MASK

DDS_ERR_FILE_ID_MASK

DDS_SUCCESS

dds_err_nr (e)

Macro to extract error number

dds_err_line (e)

Macro to extract line number

dds_err_file_id (e)

Macro to extract file identifier

DDS_ERR_CHECK (e, f)

Macro that defines dds_err_check function

DDS_FAIL (m)

Macro that defines dds_fail function

Typedefs

typedef void (***dds_fail_fn**) (const char *, const char *)

Failure handler

Functions

DDS_EXPORT const char* **dds_err_str**(dds_return_t err)

Takes the error value and outputs a string corresponding to it.

Return String corresponding to the error value

Parameters

- **err**: Error value to be converted to a string

DDS_EXPORT bool **dds_err_check**(dds_return_t err, unsigned flags, const char * where)

Takes the error number, error type and filename and line number and formats it to a string which can be used for debugging.

Return true - True

Return false - False

Parameters

- **err**: Error value

- `flags`: Indicates Fail, Exit or Report
- `where`: File and line number

DDS_EXPORT void dds_fail_set(dds_fail_fn fn)
Set the failure function.

Parameters

- `fn`: Function to invoke on failure

DDS_EXPORT dds_fail_fn dds_fail_get(void)
Get the failure function.

Return Failure function

DDS_EXPORT void dds_fail(const char * msg, const char * where)
Handles failure through an installed failure handler.

[in] `msg` String containing failure message [in] `where` String containing file and location

file **dds_public_impl.h**
`#include "ddsc/dds_public_alloc.h"#include "ddsc/dds_public_stream.h"#include "os/os_public.h"#include "ddsc/dds_export.h"` DDS C Implementation API.

This header file defines the public API for all kinds of things in the VortexDDS C language binding.

Defines

DDS_LENGTH_UNLIMITED
DDS_TOPIC_NO_OPTIMIZE
DDS_TOPIC_FIXED_KEY
DDS_READ_SAMPLE_STATE
DDS_NOT_READ_SAMPLE_STATE
DDS_ANY_SAMPLE_STATE
DDS_NEW_VIEW_STATE
DDS_NOT_NEW_VIEW_STATE
DDS_ANY_VIEW_STATE
DDS_ALIVE_INSTANCE_STATE
DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE
DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE
DDS_ANY_INSTANCE_STATE
DDS_ANY_STATE
DDS_DOMAIN_DEFAULT
DDS_HANDLE_NIL
DDS_ENTITY_NIL
DDS_ENTITY_KIND_MASK
DDS_OP_RTS

DDS_OP_ADR
DDS_OP_JSR
DDS_OP_JEQ
DDS_OP_VAL_1BY
DDS_OP_VAL_2BY
DDS_OP_VAL_4BY
DDS_OP_VAL_8BY
DDS_OP_VAL_STR
DDS_OP_VAL_BST
DDS_OP_VAL_SEQ
DDS_OP_VAL_ARR
DDS_OP_VAL_UNI
DDS_OP_VAL_STU
DDS_OP_TYPE_1BY
DDS_OP_TYPE_2BY
DDS_OP_TYPE_4BY
DDS_OP_TYPE_8BY
DDS_OP_TYPE_STR
DDS_OP_TYPE_SEQ
DDS_OP_TYPE_ARR
DDS_OP_TYPE_UNI
DDS_OP_TYPE_STU
DDS_OP_TYPE_BST
DDS_OP_TYPE_BOO
DDS_OP_SUBTYPE_BOO
DDS_OP_SUBTYPE_1BY
DDS_OP_SUBTYPE_2BY
DDS_OP_SUBTYPE_4BY
DDS_OP_SUBTYPE_8BY
DDS_OP_SUBTYPE_STR
DDS_OP_SUBTYPE_SEQ
DDS_OP_SUBTYPE_ARR
DDS_OP_SUBTYPE_UNI
DDS_OP_SUBTYPE_STU
DDS_OP_SUBTYPE_BST
DDS_OP_FLAG_KEY

DDS_OP_FLAG_DEF

Typedefs

```
typedef struct dds_sequence dds_sequence_t
typedef struct dds_key_descriptor dds_key_descriptor_t
typedef struct dds_topic_descriptor dds_topic_descriptor_t
typedef enum dds_entity_kind dds_entity_kind_t
typedef uint64_t dds_instance_handle_t
typedef int32_t dds_domainid_t
```

Enums

```
enum dds_entity_kind
    Values:

    DDS_KIND_DONTCARE = 0x00000000
    DDS_KIND_TOPIC = 0x01000000
    DDS_KIND_PARTICIPANT = 0x02000000
    DDS_KIND_READER = 0x03000000
    DDS_KIND_WRITER = 0x04000000
    DDS_KIND_SUBSCRIBER = 0x05000000
    DDS_KIND_PUBLISHER = 0x06000000
    DDS_KIND_COND_READ = 0x07000000
    DDS_KIND_COND_QUERY = 0x08000000
    DDS_KIND_WAITSET = 0x09000000
    DDS_KIND_INTERNAL = 0x0A000000
```

Functions

DDS_EXPORT void dds_write_set_batch(bool enable)

Description : Enable or disable write batching. Overrides default configuration setting for write batching (DDSI2E/Internal/WriteBatch).

Arguments :

1. enable Enables or disables write batching for all writers.

DDS_EXPORT void dds_ssl_plugin(void)

Description : Install tcp/ssl and encryption support. Depends on openssl.

Arguments :

1. None

DDS_EXPORT void dds_durability_plugin(void)

Description : Install client durability support. Depends on OSPL server.

Arguments :

1. None

file **dds_public_listener.h**

#include "ddsc/dds_export.h" #include "ddsc/dds_public_impl.h" #include "ddsc/dds_public_status.h" #include "os/os_public.h" DDS C Listener API.

This header file defines the public API of listeners in the VortexDDS C language binding.

Defines

DDS_LUNSET

Typedefs

```
typedef void (*dds_on_inconsistent_topic_fn) (dds_entity_t      topic,      const
                                              dds_inconsistent_topic_status_t status,
                                              void *arg)
```

```
typedef void (*dds_on_liveliness_lost_fn) (dds_entity_t      writer,      const
                                              dds_liveliness_lost_status_t   status,   void
                                              *arg)
```

```
typedef void (*dds_on_offered_deadline_missed_fn) (dds_entity_t  writer,  const
                                                    dds_offered_deadline_missed_status_t
                                                    status, void *arg)
```

```
typedef void (*dds_on_offered_incompatible_qos_fn) (dds_entity_t  writer,  const
                                                    dds_offered_incompatible_qos_status_t
                                                    status, void *arg)
```

```
typedef void (*dds_on_data_on_readers_fn) (dds_entity_t subscriber, void *arg)
```

```
typedef void (*dds_on_sample_lost_fn) (dds_entity_t      reader,      const
                                         dds_sample_lost_status_t status, void *arg)
```

```
typedef void (*dds_on_data_available_fn) (dds_entity_t reader, void *arg)
```

```
typedef void (*dds_on_sample_rejected_fn) (dds_entity_t      reader,      const
                                              dds_sample_rejected_status_t   status,   void
                                              *arg)
```

```
typedef void (*dds_on_liveliness_changed_fn) (dds_entity_t      reader,      const
                                              dds_liveliness_changed_status_t   sta-
                                              tus, void *arg)
```

```
typedef void (*dds_on_requested_deadline_missed_fn) (dds_entity_t  reader,  const
                                                    dds_requested_deadline_missed_status_t
                                                    status, void *arg)
```

```
typedef void (*dds_on_requested_incompatible_qos_fn) (dds_entity_t  reader,  const
                                                    dds_requested_incompatible_qos_status_t
                                                    status, void *arg)
```

```
typedef void (*dds_on_publication_matched_fn) (dds_entity_t      writer,      const
                                              dds_publication_matched_status_t
                                              status, void *arg)
```

```
typedef void (*dds_on_subscription_matched_fn) (dds_entity_t    reader,    const
                                                dds_subscription_matched_status_t
                                                status, void *arg)

typedef struct c_listener dds_listener_t
```

Functions

`_Ret_notnull_ DDS_EXPORT dds_listener_t* dds_listener_create(_In_opt_ void * arg)`
Allocate memory and initializes to default values (::DDS_LUNSET) of a listener.

Return Returns a pointer to the allocated memory for dds_listener_t structure.

Parameters

- `arg`: optional pointer that will be passed on to the listener callbacks

`DDS_EXPORT void dds_listener_delete(_In_ _Post_invalid_ dds_listener_t * listener)`
Delete the memory allocated to listener structure.

Parameters

- `listener`: pointer to the listener struct to delete

`DDS_EXPORT void dds_listener_reset(_Out_ dds_listener_t * listener)`
Reset the listener structure contents to ::DDS_LUNSET.

Parameters

- `listener`: pointer to the listener struct to reset

`DDS_EXPORT void dds_listener_copy(_Out_ dds_listener_t * dst, _In_ const dds_listener_t * src)`
Copy the listener callbacks from source to destination.

Parameters

- `dst`: The pointer to the destination listener structure, where the content is to copied
- `src`: The pointer to the source listener structure to be copied

`DDS_EXPORT void dds_listener_merge(_Inout_ dds_listener_t * dst, _In_ const dds_listener_t * src)`
Copy the listener callbacks from source to destination, unless already set.

Any listener callbacks already set in `dst` (including NULL) are skipped, only those set to DDS_LUNSET are copied from `src`.

Parameters

- `dst`: The pointer to the destination listener structure, where the content is merged
- `src`: The pointer to the source listener structure to be copied

`DDS_EXPORT void dds_lset_inconsistent_topic(_Inout_ dds_listener_t * listener, _In_opt_ void * callback)`
Set the inconsistent_topic callback in the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be set
- `callback`: The callback to set in the listener, can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lset_liveliness_lost(_Inout_ dds_listener_t * listener, _In_opt_ dds_listener_t * callback);
Set the liveliness_lost callback in the listener structure.

Parameters

- **listener:** The pointer to the listener structure, where the callback will be set
- **callback:** The callback to set in the listener, can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lset_offered_deadline_missed(_Inout_ dds_listener_t * listener, _In_opt_ dds_listener_t * callback);
Set the offered_deadline_missed callback in the listener structure.

Parameters

- **listener:** The pointer to the listener structure, where the callback will be set
- **callback:** The callback to set in the listener, can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lset_offered_incompatible_qos(_Inout_ dds_listener_t * listener, _In_opt_ dds_listener_t * callback);
Set the offered_incompatible_qos callback in the listener structure.

Parameters

- **listener:** The pointer to the listener structure, where the callback will be set
- **callback:** The callback to set in the listener, can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lset_data_on_readers(_Inout_ dds_listener_t * listener, _In_opt_ dds_listener_t * callback);
Set the data_on_readers callback in the listener structure.

Parameters

- **listener:** The pointer to the listener structure, where the callback will be set
- **callback:** The callback to set in the listener, can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lset_sample_lost(_Inout_ dds_listener_t * listener, _In_opt_ dds_listener_t * callback);
Set the sample_lost callback in the listener structure.

Parameters

- **listener:** The pointer to the listener structure, where the callback will be set
- **callback:** The callback to set in the listener, can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lset_data_available(_Inout_ dds_listener_t * listener, _In_opt_ dds_listener_t * callback);
Set the data_available callback in the listener structure.

Parameters

- **listener:** The pointer to the listener structure, where the callback will be set
- **callback:** The callback to set in the listener, can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lset_sample_rejected(_Inout_ dds_listener_t * listener, _In_opt_ dds_listener_t * callback);
Set the sample_rejected callback in the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be set
- `callback`: The callback to set in the listener, can be `NULL`, `::DDS_LUNSET` or a valid callback pointer

DDS_EXPORT void dds_lset_liveliness_changed(_Inout_ dds_listener_t * listener, _In_opt_
Set the `liveliness_changed` callback in the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be set
- `callback`: The callback to set in the listener, can be `NULL`, `::DDS_LUNSET` or a valid callback pointer

DDS_EXPORT void dds_lset_requested_deadline_missed(_Inout_ dds_listener_t * listener, _In_opt_
Set the `requested_deadline_missed` callback in the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be set
- `callback`: The callback to set in the listener, can be `NULL`, `::DDS_LUNSET` or a valid callback pointer

DDS_EXPORT void dds_lset_requested_incompatible_qos(_Inout_ dds_listener_t * listener, _In_opt_
Set the `requested_incompatible_qos` callback in the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be set
- `callback`: The callback to set in the listener, can be `NULL`, `::DDS_LUNSET` or a valid callback pointer

DDS_EXPORT void dds_lset_publication_matched(_Inout_ dds_listener_t * listener, _In_opt_
Set the `publication_matched` callback in the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be set
- `callback`: The callback to set in the listener, can be `NULL`, `::DDS_LUNSET` or a valid callback pointer

DDS_EXPORT void dds_lset_subscription_matched(_Inout_ dds_listener_t * listener, _In_opt_
Set the `subscription_matched` callback in the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be set
- `callback`: The callback to set in the listener, can be `NULL`, `::DDS_LUNSET` or a valid callback pointer

DDS_EXPORT void dds_lget_inconsistent_topic(_In_ const dds_listener_t * listener, _Out_
Get the `inconsistent_topic` callback from the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be retrieved from

- `callback`: Pointer where the retrieved callback can be stored; can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lget_liveliness_lost(_In_ const dds_listener_t * listener, _Outptr_
Get the liveliness_lost callback from the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be retrieved from
- `callback`: Pointer where the retrieved callback can be stored; can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lget_offered_deadline_missed(_In_ const dds_listener_t * listener,
Get the offered_deadline_missed callback from the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be retrieved from
- `callback`: Pointer where the retrieved callback can be stored; can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lget_offered_incompatible_qos(_In_ const dds_listener_t * listener,
Get the offered_incompatible_qos callback from the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be retrieved from
- `callback`: Pointer where the retrieved callback can be stored; can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lget_data_on_readers(_In_ const dds_listener_t * listener, _Outptr_
Get the data_on_readers callback from the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be retrieved from
- `callback`: Pointer where the retrieved callback can be stored; can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lget_sample_lost(_In_ const dds_listener_t * listener, _Outptr_res
Get the sample_lost callback from the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be retrieved from
- `callback`: Pointer where the retrieved callback can be stored; can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lget_data_available(_In_ const dds_listener_t * listener, _Outptr_
Get the data_available callback from the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be retrieved from
- `callback`: Pointer where the retrieved callback can be stored; can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lget_sample_rejected(_In_ const dds_listener_t * listener, _Outptr_ void ** callback);
Get the sample_rejected callback from the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be retrieved from
- `callback`: Pointer where the retrieved callback can be stored; can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lget_liveliness_changed(_In_ const dds_listener_t * listener, _Outptr_ void ** callback);
Get the liveliness_changed callback from the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be retrieved from
- `callback`: Pointer where the retrieved callback can be stored; can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lget_requested_deadline_missed(_In_ const dds_listener_t * listener, _Outptr_ void ** callback);
Get the requested_deadline_missed callback from the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be retrieved from
- `callback`: Pointer where the retrieved callback can be stored; can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lget_requested_incompatible_qos(_In_ const dds_listener_t * listener, _Outptr_ void ** callback);
Get the requested_incompatible_qos callback from the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be retrieved from
- `callback`: Pointer where the retrieved callback can be stored; can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lget_publication_matched(_In_ const dds_listener_t * listener, _Outptr_ void ** callback);
Get the publication_matched callback from the listener structure.

Parameters

- `listener`: The pointer to the listener structure, where the callback will be retrieved from
- `callback`: Pointer where the retrieved callback can be stored; can be NULL, ::DDS_LUNSET or a valid callback pointer

DDS_EXPORT void dds_lget_subscription_matched(_In_ const dds_listener_t * listener, _Outptr_ void ** callback);
Get the subscription_matched callback from the listener structure.

Parameters

- `callback`: Pointer where the retrieved callback can be stored; can be NULL, ::DDS_LUNSET or a valid callback pointer
- `listener`: The pointer to the listener structure, where the callback will be retrieved from

file **dds_public_log.h**

#include "os/os_public.h"#include "ddsc/dds_export.h" DDS C Logging API.

This header file defines the public API for logging in the VortexDDS C language binding.

Functions

DDS_EXPORT void dds_log_info(const char * fmt, ...)

DDS_EXPORT void dds_log_warn(const char * fmt, ...)

DDS_EXPORT void dds_log_error(const char * fmt, ...)

DDS_EXPORT void dds_log_fatal(const char * fmt, ...)

file **dds_public_qos.h**

#include "os/os_public.h"#include "ddsc/dds_export.h" DDS C QoS API.

This header file defines the public API of QoS and Policies in the VortexDDS C language binding.

QoS identifiers

DDS_INVALID_QOS_POLICY_ID

DDS_USERDATA_QOS_POLICY_ID

DDS_DURABILITY_QOS_POLICY_ID

DDS_PRESENTATION_QOS_POLICY_ID

DDS_DEADLINE_QOS_POLICY_ID

DDS_LATENCYBUDGET_QOS_POLICY_ID

DDS_OWNERSHIP_QOS_POLICY_ID

DDS_OWNERSHIPSTRENGTH_QOS_POLICY_ID

DDS_LIVELINESS_QOS_POLICY_ID

DDS_TIMEBASEDFILTER_QOS_POLICY_ID

DDS_PARTITION_QOS_POLICY_ID

DDS_RELIABILITY_QOS_POLICY_ID

DDS_DESTINATIONORDER_QOS_POLICY_ID

DDS_HISTORY_QOS_POLICY_ID

DDS_RESOURCELIMITS_QOS_POLICY_ID

DDS_ENTITYFACTORY_QOS_POLICY_ID

DDS_WRITERDATALIFECYCLE_QOS_POLICY_ID

DDS_READERDATALIFECYCLE_QOS_POLICY_ID

DDS_TOPICDATA_QOS_POLICY_ID

DDS_GROUPDATA_QOS_POLICY_ID

DDS_TRANSPORTPRIORITY_QOS_POLICY_ID

DDS_LIFESPAN_QOS_POLICY_ID

DDS_DURABILITYSERVICE_QOS_POLICY_ID

Typedefs

typedef struct nn_xqos dds_qos_t

QoS structure

typedef enum dds_durability_kind dds_durability_kind_t

Durability QoS: Applies to Topic, DataReader, DataWriter

typedef enum dds_history_kind dds_history_kind_t

History QoS: Applies to Topic, DataReader, DataWriter

typedef enum dds_ownership_kind dds_ownership_kind_t

Ownership QoS: Applies to Topic, DataReader, DataWriter

typedef enum dds_liveliness_kind dds_liveliness_kind_t

Liveliness QoS: Applies to Topic, DataReader, DataWriter

typedef enum dds_reliability_kind dds_reliability_kind_t

Reliability QoS: Applies to Topic, DataReader, DataWriter

typedef enum dds_destination_order_kind dds_destination_order_kind_t

DestinationOrder QoS: Applies to Topic, DataReader, DataWriter

typedef struct dds_history_qospolicy dds_history_qospolicy_t

History QoS: Applies to Topic, DataReader, DataWriter

typedef struct dds_resource_limits_qospolicy dds_resource_limits_qospolicy_t

ResourceLimits QoS: Applies to Topic, DataReader, DataWriter

typedef enum dds_presentation_access_scope_kind dds_presentation_access_scope_kind_t

Presentation QoS: Applies to Publisher, Subscriber

Enums

enum dds_durability_kind

Durability QoS: Applies to Topic, DataReader, DataWriter

Values:

DDS_DURABILITY_VOLATILE

DDS_DURABILITY_TRANSIENT_LOCAL

DDS_DURABILITY_TRANSIENT

DDS_DURABILITY_PERSISTENT

enum dds_history_kind

History QoS: Applies to Topic, DataReader, DataWriter

Values:

DDS_HISTORY_KEEP_LAST

DDS_HISTORY_KEEP_ALL

enum dds_ownership_kind

Ownership QoS: Applies to Topic, DataReader, DataWriter

Values:

DDS_OWNERSHIP_SHARED

DDS_OWNERSHIP_EXCLUSIVE

enum dds_liveliness_kind

Liveliness QoS: Applies to Topic, DataReader, DataWriter

Values:

DDS_LIVELINESS_AUTOMATIC

DDS_LIVELINESS_MANUAL_BY_PARTICIPANT

DDS_LIVELINESS_MANUAL_BY_TOPIC

enum dds_reliability_kind

Reliability QoS: Applies to Topic, DataReader, DataWriter

Values:

DDS_RELIABILITY_BEST_EFFORT

DDS_RELIABILITY_RELIABLE

enum dds_destination_order_kind

DestinationOrder QoS: Applies to Topic, DataReader, DataWriter

Values:

DDS_DESTINATIONORDER_BY_RECEPTION_TIMESTAMP

DDS_DESTINATIONORDER_BY_SOURCE_TIMESTAMP

enum dds_presentation_access_scope_kind

Presentation QoS: Applies to Publisher, Subscriber

Values:

DDS_PRESENTATION_INSTANCE

DDS_PRESENTATION_TOPIC

DDS_PRESENTATION_GROUP

Functions

`_Ret_notnull_ DDS_EXPORT dds_qos_t* dds_qos_create(void)`

Allocate memory and initialize default QoS-policies.

Return - Pointer to the initialized dds_qos_t structure, NULL if unsuccessful.

`DDS_EXPORT void dds_qos_delete(_In_ _Post_invalid_ dds_qos_t * qos)`

Delete memory allocated to QoS-policies structure.

Parameters

- qos: - Pointer to dds_qos_t structure

`DDS_EXPORT void dds_qos_reset(_Out_ dds_qos_t * qos)`

Reset a QoS-policies structure to default values.

Parameters

- qos: - Pointer to the dds_qos_t structure

DDS_EXPORT dds_return_t dds_qos_copy(_Out_ dds_qos_t * dst, _In_ const dds_qos_t * src)
Copy all QoS-policies from one structure to another.

Return - Return-code indicating success or failure

Parameters

- **dst**: - Pointer to the destination dds_qos_t structure
- **src**: - Pointer to the source dds_qos_t structure

DDS_EXPORT void dds_qos_merge(_Inout_ dds_qos_t * dst, _In_ const dds_qos_t * src)
Copy all QoS-policies from one structure to another, unless already set.

Policies are copied from src to dst, unless src already has the policy set to a non-default value.

Parameters

- **dst**: - Pointer to the destination qos structure
- **src**: - Pointer to the source qos structure

DDS_EXPORT void dds_qoset_userdata(_Inout_ dds_qos_t * qos, _In_reads_bytes_opt_(sz) const void * value)
Set the userdata of a qos structure.

Parameters

- **qos**: - Pointer to a dds_qos_t structure that will store the userdata
- **value**: - Pointer to the userdata
- **sz**: - Size of userdata stored in value

DDS_EXPORT void dds_qoset_topicdata(_Inout_ dds_qos_t * qos, _In_reads_bytes_opt_(sz) const void * value)
Set the topicdata of a qos structure.

Parameters

- **qos**: - Pointer to a dds_qos_t structure that will store the topicdata
- **value**: - Pointer to the topicdata
- **sz**: - Size of the topicdata stored in value

DDS_EXPORT void dds_qoset_groupdata(_Inout_ dds_qos_t * qos, _In_reads_bytes_opt_(sz) const void * value)
Set the groupdata of a qos structure.

Parameters

- **qos**: - Pointer to a dds_qos_t structure that will store the groupdata
- **value**: - Pointer to the group data
- **sz**: - Size of groupdata stored in value

DDS_EXPORT void dds_qoset_durability(_Inout_ dds_qos_t * qos, _In_range_(DDS_DURABILITY_QOS_BEST_EFFORT, DDS_DURABILITY_QOS_STRICT) DDS_DurabilityKind_t kind)
Set the durability policy of a qos structure.

Parameters

- **qos**: - Pointer to a dds_qos_t structure that will store the policy
- **kind**: - Durability kind value DCPS_QoS_Durability

DDS_EXPORT void dds_qos_set_history(_Inout_ dds_qos_t * qos, _In_range_(DDS_HISTORY_KEEP, DDS_HISTORY_KEEP_MAX), DDS_HISTORY_KEEP_MAX)
Set the history policy of a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure that will store the policy
- kind: - History kind value DCPS_QoS_History
- depth: - History depth value DCPS_QoS_History

DDS_EXPORT void dds_qos_set_resource_limits(_Inout_ dds_qos_t * qos, _In_range_(0, DDS_RESOURCE_LIMITS_MAX), DDS_RESOURCE_LIMITS_MAX)
Set the resource limits policy of a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure that will store the policy
- max_samples: - Number of samples resource-limit value
- max_instances: - Number of instances resource-limit value
- max_samples_per_instance: - Number of samples per instance resource-limit value

DDS_EXPORT void dds_qos_set_presentation(_Inout_ dds_qos_t * qos, _In_range_(DDS_PRESENTATION_FIRST, DDS_PRESENTATION_LAST), DDS_PRESENTATION_LAST)
Set the presentation policy of a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure that will store the policy
- access_scope: - Access-scope kind
- coherent_access: - Coherent access enable value
- ordered_access: - Ordered access enable value

DDS_EXPORT void dds_qos_set_lifespan(_Inout_ dds_qos_t * qos, _In_range_(0, DDS_LIFESPAN_MAX), DDS_LIFESPAN_MAX)
Set the lifespan policy of a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure that will store the policy
- lifespan: - Lifespan duration (expiration time relative to source timestamp of a sample)

DDS_EXPORT void dds_qos_set_deadline(_Inout_ dds_qos_t * qos, _In_range_(0, DDS_DEADLINE_MAX), DDS_DEADLINE_MAX)
Set the deadline policy of a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure that will store the policy
- deadline: - Deadline duration

DDS_EXPORT void dds_qos_set_latency_budget(_Inout_ dds_qos_t * qos, _In_range_(0, DDS_LATENCY_BUDGET_MAX), DDS_LATENCY_BUDGET_MAX)
Set the latency-budget policy of a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure that will store the policy
- duration: - Latency budget duration

DDS_EXPORT void dds_qos_set_ownership(_Inout_ dds_qos_t * qos, _In_range_(DDS_OWNERSHIP) OwnershipKind kind);
Set the ownership policy of a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure that will store the policy
- kind: - Ownership kind

DDS_EXPORT void dds_qos_set_ownership_strength(_Inout_ dds_qos_t * qos, _In_ int32_t value);
Set the ownership strength policy of a qos structure.

param[in,out] qos - Pointer to a dds_qos_t structure that will store the policy param[in] value - Ownership strength value

DDS_EXPORT void dds_qos_set_liveliness(_Inout_ dds_qos_t * qos, _In_range_(DDS_LIVELINESS) LivelinessKind kind, _In_ lease_duration_t lease_duration);
Set the liveliness policy of a qos structure.

param[in,out] qos - Pointer to a dds_qos_t structure that will store the policy param[in] kind - Liveliness kind param[in] lease_duration - Lease duration

DDS_EXPORT void dds_qos_set_time_based_filter(_Inout_ dds_qos_t * qos, _In_range_(0, DDS_TIME_BASED_FILTER_MAX) DDS_TimeBasedFilterKind kind, _In_ double filter_value);
Set the time-based filter policy of a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure that will store the policy
- minimum_separation: - Minimum duration between sample delivery for an instance

DDS_EXPORT void dds_qos_set_partition(_Inout_ dds_qos_t * qos, _In_ uint32_t n, _In_ DDS_PartitionKind kind);
Set the partition policy of a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure that will store the policy
- n: - Number of partitions stored in ps
-

DDS_EXPORT void dds_qos_set_reliability(_Inout_ dds_qos_t * qos, _In_range_(DDS_RELIABILITY) ReliabilityKind kind, _In_ double max_blocking_time);
Set the reliability policy of a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure that will store the policy
- kind: - Reliability kind
- max_blocking_time: - Max blocking duration applied when kind is reliable.

DDS_EXPORT void dds_qos_set_transport_priority(_Inout_ dds_qos_t * qos, _In_ int32_t value);
Set the transport-priority policy of a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure that will store the policy
- value: - Priority value

DDS_EXPORT void dds_qos_set_destination_order(_Inout_ dds_qos_t * qos, _In_range_(DDS_DESTINATION_ORDER) DestinationOrderKind kind);
Set the destination-order policy of a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure that will store the policy
- kind: - Destination-order kind

DDS_EXPORT void dds_qoset_writer_data_lifecycle(_Inout_ dds_qos_t * qos, _In_ bool auto_dispose_unregistered_instances);
Set the writer data-lifecycle policy of a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure that will store the policy
- autodispose_unregistered_instances: - Automatic disposal of unregistered instances

DDS_EXPORT void dds_qoset_reader_data_lifecycle(_Inout_ dds_qos_t * qos, _In_range_(0, DDS_QOS_DEADLINE_INTERVAL_MAX) double deadline_interval);
Set the reader data-lifecycle policy of a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure that will store the policy
- autopurge_nowriter_samples_delay: - Delay for purging of samples from instances in a no-writers state
- autopurge_disposed_samples_delay: - Delay for purging of samples from disposed instances

DDS_EXPORT void dds_qoset_durability_service(_Inout_ dds_qos_t * qos, _In_range_(0, DDS_QOS_DEADLINE_INTERVAL_MAX) double deadline_interval, DDS_DURABILITY_SERVICE_KIND kind, DDS_DURABILITY_POLICY_POLICY_KIND policy_kind, DDS_DURABILITY_POLICY_DEPTH policy_depth, DDS_DURABILITY_POLICY_RESOURCE_LIMIT policy_resource_limit, DDS_DURABILITY_POLICY_RESOURCE_LIMIT_INSTANCE policy_resource_limit_instance, DDS_DURABILITY_POLICY_RESOURCE_LIMIT_INSTANCE_POLICY policy_resource_limit_instance_policy);
Set the durability-service policy of a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure that will store the policy
- service_cleanup_delay: - Delay for purging of abandoned instances from the durability service
- history_kind: - History policy kind applied by the durability service
- history_depth: - History policy depth applied by the durability service
- max_samples: - Number of samples resource-limit policy applied by the durability service
- max_instances: - Number of instances resource-limit policy applied by the durability service
- max_samples_per_instance: - Number of samples per instance resource-limit policy applied by the durability service

DDS_EXPORT void dds_qoset_userdata(_In_ const dds_qos_t * qos, _Outptr_result_bytebuf_ void ** user_data, DDS_QOS_USERDATA_SIZE_T user_data_size);
Get the userdata from a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- value: - Pointer that will store the userdata
- sz: - Pointer that will store the size of userdata

DDS_EXPORT void dds_qoset_topicdata(_In_ const dds_qos_t * qos, _Outptr_result_bytebuf_ void ** topic_data, DDS_QOS_TOPICDATA_SIZE_T topic_data_size);
Get the topicdata from a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- value: - Pointer that will store the topicdata
- sz: - Pointer that will store the size of topicdata

DDS_EXPORT void dds_qget_groupdata(_In_ const dds_qos_t * qos, _Outptr_result_bytebu
Get the groupdata from a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- value: - Pointer that will store the groupdata
- sz: - Pointer that will store the size of groupdata

DDS_EXPORT void dds_qget_durability(_In_ const dds_qos_t * qos, _Out_ dds_durabilit
Get the durability policy from a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- kind: - Pointer that will store the durability kind

DDS_EXPORT void dds_qget_history(_In_ const dds_qos_t * qos, _Out_opt_ dds_history_
Get the history policy from a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- kind: - Pointer that will store the history kind (optional)
- depth: - Pointer that will store the history depth (optional)

DDS_EXPORT void dds_qget_resource_limits(_In_ const dds_qos_t * qos, _Out_opt_ int32
Get the resource-limits policy from a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- max_samples: - Pointer that will store the number of samples resource-limit (optional)
- max_instances: - Pointer that will store the number of instances resource-limit (optional)
- max_samples_per_instance: - Pointer that will store the number of samples per instance resource-limit (optional)

DDS_EXPORT void dds_qget_presentation(_In_ const dds_qos_t * qos, _Out_opt_ dds_pre
Get the presentation policy from a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- access_scope: - Pointer that will store access scope kind (optional)
- coherent_access: - Pointer that will store coherent access enable value (optional)
- ordered_access: - Pointer that will store orderede access enable value (optional)

DDS_EXPORT void dds_qget_lifespan(_In_ const dds_qos_t * qos, _Out_ dds_duration_t
Get the lifespan policy from a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- lifespan: - Pointer that will store lifespan duration

DDS_EXPORT void dds_qget_deadline(_In_ const dds_qos_t * qos, _Out_ dds_duration_t
Get the deadline policy from a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- deadline: - Pointer that will store deadline duration

DDS_EXPORT void dds_qget_latency_budget(_In_ const dds_qos_t * qos, _Out_ dds_durat
Get the latency-budget policy from a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- duration: - Pointer that will store latency-budget duration

DDS_EXPORT void dds_qget_ownership(_In_ const dds_qos_t * qos, _Out_ dds_ownership_t
Get the ownership policy from a qos structure.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- kind: - Pointer that will store the ownership kind

DDS_EXPORT void dds_qget_ownership_strength(_In_ const dds_qos_t * qos, _Out_ int32_t
Get the ownership strength qos policy.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- value: - Pointer that will store the ownership strength value

DDS_EXPORT void dds_qget_liveliness(_In_ const dds_qos_t * qos, _Out_opt_ dds_livel
Get the liveliness qos policy.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- kind: - Pointer that will store the liveliness kind (optional)
- lease_duration: - Pointer that will store the liveliness lease duration (optional)

DDS_EXPORT void dds_qget_time_based_filter(_In_ const dds_qos_t * qos, _Out_ dds_du
Get the time-based filter qos policy.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- minimum_separation: - Pointer that will store the minimum separation duration (optional)

DDS_EXPORT void dds_qget_partition(_In_ const dds_qos_t * qos, _Out_ uint32_t * n, _
Get the partition qos policy.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- n: - Pointer that will store the number of partitions (optional)
- ps: - Pointer that will store the string(s) containing partition name(s) (optional)

DDS_EXPORT void dds_qget_reliability(_In_ const dds_qos_t * qos, _Out_opt_ dds_reli.
Get the reliability qos policy.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- kind: - Pointer that will store the reliability kind (optional)
- max_blocking_time: - Pointer that will store the max blocking time for reliable reliability (optional)

DDS_EXPORT void dds_qget_transport_priority(_In_ const dds_qos_t * qos, _Out_ int32_t
Get the transport priority qos policy.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- value: - Pointer that will store the transport priority value

DDS_EXPORT void dds_qget_destination_order(_In_ const dds_qos_t * qos, _Out_ dds_de
Get the destination-order qos policy.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- kind: - Pointer that will store the destination-order kind

DDS_EXPORT void dds_qget_writer_data_lifecycle(_In_ const dds_qos_t * qos, _Out_ boo
Get the writer data-lifecycle qos policy.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- autodispose_unregistered_instances: - Pointer that will store the autodispose un-registered instances enable value

DDS_EXPORT void dds_qget_reader_data_lifecycle(_In_ const dds_qos_t * qos, _Out_opt_
Get the reader data-lifecycle qos policy.

Parameters

- qos: - Pointer to a dds_qos_t structure storing the policy
- autopurge_nowriter_samples_delay: - Pointer that will store the delay for auto-purging samples from instances in a no-writer state (optional)
- autopurge_disposed_samples_delay: - Pointer that will store the delay for auto-purging of disposed instances (optional)

DDS_EXPORT void dds_qget_durability_service(_In_ const dds_qos_t * qos, _Out_opt_ d
Get the durability-service qos policy values.

Parameters

- `qos`: - Pointer to a `dds_qos_t` structure storing the policy
- `service_cleanup_delay`: - Pointer that will store the delay for purging of abandoned instances from the durability service (optional)
- `history_kind`: - Pointer that will store history policy kind applied by the durability service (optional)
- `history_depth`: - Pointer that will store history policy depth applied by the durability service (optional)
- `max_samples`: - Pointer that will store number of samples resource-limit policy applied by the durability service (optional)
- `max_instances`: - Pointer that will store number of instances resource-limit policy applied by the durability service (optional)
- `max_samples_per_instance`: - Pointer that will store number of samples per instance resource-limit policy applied by the durability service (optional)

file `dds_public_status.h`

`#include "os/os_public.h" #include "ddsc/dds_export.h"` DDS C Communication Status API.

This header file defines the public API of the Communication Status in the VortexDDS C language binding.

Typedefs

```
typedef struct dds_offered_deadline_missed_status dds_offered_deadline_missed_status_t
    DCPS_Status_OfferedDeadlineMissed

typedef struct dds_offered_incompatible_qos_status dds_offered_incompatible_qos_status_t
    DCPS_Status_OfferedIncompatibleQoS

typedef struct dds_publication_matched_status dds_publication_matched_status_t
    DCPS_Status_PublicationMatched

typedef struct dds_liveliness_lost_status dds_liveliness_lost_status_t
    DCPS_Status_LivelinessLost

typedef struct dds_subscription_matched_status dds_subscription_matched_status_t
    DCPS_Status_SubscriptionMatched

typedef struct dds_sample_rejected_status dds_sample_rejected_status_t
    DCPS_Status_SampleRejected

typedef struct dds_liveliness_changed_status dds_liveliness_changed_status_t
    DCPS_Status_LivelinessChanged

typedef struct dds_requested_deadline_missed_status dds_requested_deadline_missed_status_t
    DCPS_Status_RequestedDeadlineMissed

typedef struct dds_requested_incompatible_qos_status dds_requested_incompatible_qos_status_t
    DCPS_Status_RequestedIncompatibleQoS

typedef struct dds_sample_lost_status dds_sample_lost_status_t
    DCPS_Status_SampleLost

typedef struct dds_inconsistent_topic_status dds_inconsistent_topic_status_t
    DCPS_Status_InconsistentTopic
```

Enums

enum dds_sample_rejected_status_kind

dds_sample_rejected_status_kind

Values:

DDS_NOT_REJECTED

DDS_REJECTED_BY_INSTANCES_LIMIT

DDS_REJECTED_BY_SAMPLES_LIMIT

DDS_REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT

Functions

_Pre_satisfies_((topic &DDS_ENTITY_KIND_MASK) == DDS_KIND_TOPIC)

Get INCONSISTENT_TOPIC status.

This operation gets the status value corresponding to INCONSISTENT_TOPIC and reset the status. The value can be obtained, only if the status is enabled for an entity. NULL value for status is allowed and it will reset the trigger value when status is enabled.

Return 0 - Success

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- **topic:** The entity to get the status
- **status:** The pointer to DCPS_Status_InconsistentTopic to get the status

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER: One of the given arguments is not valid.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

_Pre_satisfies_(((writer &DDS_ENTITY_KIND_MASK) == DDS_KIND_WRITER))

Get PUBLICATION_MATCHED status.

Get OFFERED_INCOMPATIBLE_QOS status.

Get OFFERED_DEADLINE_MISSED status.

Get LIVELINESS_LOST status.

This operation gets the status value corresponding to PUBLICATION_MATCHED and reset the status. The value can be obtained, only if the status is enabled for an entity. NULL value for status is allowed and it will reset the trigger value when status is enabled.

This operation gets the status value corresponding to LIVELINESS_LOST and reset the status. The value can be obtained, only if the status is enabled for an entity. NULL value for status is allowed and it will reset the trigger value when status is enabled.

Return 0 - Success

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- `writer`: The entity to get the status
- `status`: The pointer to `DCPS_Status_PublicationMatched` to get the status

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation gets the status value corresponding to `OFFERED_DEADLINE_MISSED` and reset the status. The value can be obtained, only if the status is enabled for an entity. NULL value for status is allowed and it will reset the trigger value when status is enabled.

Return 0 - Success

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `writer`: The entity to get the status
- `status`: The pointer to `DCPS_Status_LivelinessLost` to get the status

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation gets the status value corresponding to `OFFERED_INCOMPATIBLE_QOS` and reset the status. The value can be obtained, only if the status is enabled for an entity. NULL value for status is allowed and it will reset the trigger value when status is enabled.

Return 0 - Success

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `writer`: The entity to get the status
- `status`: The pointer to `DCPS_Status_OfferedDeadlineMissed` to get the status

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

Return 0 - Success

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `writer`: The writer entity to get the status
- `status`: The pointer to `DCPS_Status_OfferedIncompatibleQoS` to get the status

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

`_Pre_satisfies_((reader &DDS_ENTITY_KIND_MASK) == DDS_KIND_READER)`

Get `SUBSCRIPTION_MATCHED` status.

Get `REQUESTED_INCOMPATIBLE_QOS` status.

Get `REQUESTED_DEADLINE_MISSED` status.

Get `SAMPLE_LOST` status.

Get `SAMPLE_REJECTED` status.

Get `LIVELINESS_CHANGED` status.

This operation gets the status value corresponding to `SUBSCRIPTION_MATCHED` and reset the status. The value can be obtained, only if the status is enabled for an entity. NULL value for status is allowed and it will reset the trigger value when status is enabled.

This operation gets the status value corresponding to `LIVELINESS_CHANGED` and reset the status. The value can be obtained, only if the status is enabled for an entity. NULL value for status is allowed and it will reset the trigger value when status is enabled.

Return 0 - Success

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader`: The reader entity to get the status
- `status`: The pointer to `DCPS_Status_SubscriptionMatched` to get the status

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation gets the status value corresponding to `SAMPLE_REJECTED` and reset the status. The value can be obtained, only if the status is enabled for an entity. NULL value for status is allowed and it will reset the trigger value when status is enabled.

Return 0 - Success

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader`: The entity to get the status
- `status`: The pointer to `DCPS_Status_LivelinessChanged` to get the status

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation gets the status value corresponding to `SAMPLE_LOST` and reset the status. The value can be obtained, only if the status is enabled for an entity. NULL value for status is allowed and it will reset the trigger value when status is enabled.

Return 0 - Success

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader`: The entity to get the status
- `status`: The pointer to `DCPS_Status_SampleRejected` to get the status

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation gets the status value corresponding to `REQUESTED_DEADLINE_MISSED` and reset the status. The value can be obtained, only if the status is enabled for an entity. NULL value for status is allowed and it will reset the trigger value when status is enabled.

Return A `dds_return_t` indicating success or failure

Parameters

- `reader`: The entity to get the status
- `status`: The pointer to `DCPS_Status_SampleLost` to get the status

Return Value

- `DDS_RETCODE_OK`: Success
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation gets the status value corresponding to `REQUESTED_INCOMPATIBLE_QOS` and reset the status. The value can be obtained, only if the status is enabled for an entity. NULL value for status is allowed and it will reset the trigger value when status is enabled.

Return A `dds_return_t` indicating success or failure

Parameters

- `reader`: The entity to get the status
- `status`: The pointer to `DCPS_Status_RequestedDeadlineMissed` to get the status

Return Value

- `DDS_RETCODE_OK`: Success
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

Return A `dds_return_t` indicating success or failure

Parameters

- `reader`: The entity to get the status
- `status`: The pointer to `DCPS_Status_RequestedIncompatibleQoS` to get the status

Return Value

- `DDS_RETCODE_OK`: Success
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

Variables

`_Out_opt_ dds_requested_incompatible_qos_status_t* status`

file `dds_public_stream.h`

`#include "os/os_public.h" #include <stdbool.h> #include "ddsc/dds_export.h"` DDS C Stream API.

This header file defines the public API of the Streams in the VortexDDS C language binding.

Defines

`DDS_STREAM_BE`

`DDS_STREAM_LE`

`dds_stream_read_char` (s)

`dds_stream_read_int8` (s)

`dds_stream_read_int16` (s)

`dds_stream_read_int32` (s)

`dds_stream_read_int64` (s)

`dds_stream_write_char` (s, v)

`dds_stream_write_int8` (s, v)

`dds_stream_write_int16` (s, v)

`dds_stream_write_int32` (s, v)


```
dds_stream_write_int64(s, v)
```

Typedefs

```
typedef struct dds_stream dds_stream_t
```

Functions

```
DDS_EXPORT dds_stream_t* dds_stream_create(size_t size)
DDS_EXPORT void dds_stream_delete(dds_stream_t * st)
DDS_EXPORT void dds_stream_fini(dds_stream_t * st)
DDS_EXPORT void dds_stream_reset(dds_stream_t * st)
DDS_EXPORT void dds_stream_init(dds_stream_t * st, size_t size)
DDS_EXPORT void dds_stream_grow(dds_stream_t * st, size_t size)
DDS_EXPORT bool dds_stream_endian(void)
DDS_EXPORT bool dds_stream_read_bool(dds_stream_t * is)
DDS_EXPORT uint8_t dds_stream_read_uint8(dds_stream_t * is)
DDS_EXPORT uint16_t dds_stream_read_uint16(dds_stream_t * is)
DDS_EXPORT uint32_t dds_stream_read_uint32(dds_stream_t * is)
DDS_EXPORT uint64_t dds_stream_read_uint64(dds_stream_t * is)
DDS_EXPORT float dds_stream_read_float(dds_stream_t * is)
DDS_EXPORT double dds_stream_read_double(dds_stream_t * is)
DDS_EXPORT char* dds_stream_read_string(dds_stream_t * is)
DDS_EXPORT void dds_stream_read_buffer(dds_stream_t * is, uint8_t * buffer, uint32_t len)
DDS_EXPORT void dds_stream_write_bool(dds_stream_t * os, bool val)
DDS_EXPORT void dds_stream_write_uint8(dds_stream_t * os, uint8_t val)
DDS_EXPORT void dds_stream_write_uint16(dds_stream_t * os, uint16_t val)
DDS_EXPORT void dds_stream_write_uint32(dds_stream_t * os, uint32_t val)
DDS_EXPORT void dds_stream_write_uint64(dds_stream_t * os, uint64_t val)
DDS_EXPORT void dds_stream_write_float(dds_stream_t * os, float val)
DDS_EXPORT void dds_stream_write_double(dds_stream_t * os, double val)
DDS_EXPORT void dds_stream_write_string(dds_stream_t * os, const char * val)
DDS_EXPORT void dds_stream_write_buffer(dds_stream_t * os, uint32_t len, uint8_t * buffer)
```

file `dds_public_time.h`

```
#include "os/os_public.h" #include "ddsc/dds_export.h" DDS C Time support API.
```

This header file defines the public API of the in the VortexDDS C language binding.

Macro definition for time units in nanoseconds.

DDS_NSECS_IN_SEC

DDS_NSECS_IN_MSEC

DDS_NSECS_IN_USEC

Infinite timeout for indicate absolute time

DDS_NEVER

Infinite timeout for relative time

DDS_INFINITY

Macro definition for time conversion from nanoseconds

DDS_SECS (n)

DDS_MSECS (n)

DDS_USECS (n)

Typedefs

typedef int64_t dds_time_t
Absolute Time definition

typedef int64_t dds_duration_t
Relative Time definition

Functions

DDS_EXPORT dds_time_t dds_time(void)
Description : This operation returns the current time (in nanoseconds)

Arguments :

1. Returns current time

DDS_EXPORT void dds_sleepfor(dds_duration_t n)
Description : This operation blocks the calling thread until the relative time n has elapsed

Arguments :

1. n Relative Time to block a thread

DDS_EXPORT void dds_sleepuntil(dds_time_t n)
Description : This operation blocks the calling thread until the absolute time n has elapsed

Arguments :

1. n absolute Time to block a thread

file **ddsv2.h**

```
#include "os/os_public.h"#include "ddsc/dds_export.h"#include "ddsc/dds_public_stream.h"#include
"ddsc/dds_public_impl.h"#include "ddsc/dds_public_alloc.h"#include "ddsc/dds_public_time.h"#include
"ddsc/dds_public_qos.h"#include "ddsc/dds_public_error.h"#include "ddsc/dds_public_status.h"#include
"ddsc/dds_public_listener.h"#include "ddsc/dds_public_log.h" C DDS header.
```

Communication Status definitions

```
DDS_INCONSISTENT_TOPIC_STATUS
DDS_OFFERED_DEADLINE_MISSED_STATUS
DDS_REQUESTED_DEADLINE_MISSED_STATUS
DDS_OFFERED_INCOMPATIBLE_QOS_STATUS
DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS
DDS_SAMPLE_LOST_STATUS
DDS_SAMPLE_REJECTED_STATUS
DDS_DATA_ON_READERS_STATUS
DDS_DATA_AVAILABLE_STATUS
DDS_LIVELINESS_LOST_STATUS
DDS_LIVELINESS_CHANGED_STATUS
DDS_PUBLICATION_MATCHED_STATUS
DDS_SUBSCRIPTION_MATCHED_STATUS
```

Typedefs

typedef enum *dds_sample_state* dds_sample_state_t

defines the state for a data value

1. DDS_SST_READ - DataReader has already accessed the sample by read
2. DDS_SST_NOT_READ - DataReader has not accessed that sample before

dds_sample_state_t

typedef enum *dds_view_state* dds_view_state_t

defines the view state of an instance relative to the samples

1. DDS_VST_NEW - DataReader is accessing the sample for the first time when the instance is alive
2. DDS_VST_OLD - DataReader has accessed the sample before

dds_view_state_t

typedef enum *dds_instance_state* dds_instance_state_t

defines the state of the instance

1. DDS_IST_ALIVE - Samples received for the instance from the live data writers
2. DDS_IST_NOT_ALIVE_DISPOSED - Instance was explicitly disposed by the data writer
3. DDS_IST_NOT_ALIVE_NO_WRITERS - Instance has been declared as not alive by data reader as there are no live data writers writing that instance

dds_instance_state_t

typedef struct *dds_sample_info* dds_sample_info_t

Structure dds_sample_info_t - contains information about the associated data value

1. sample_state - *dds_sample_state_t*
2. view_state - *dds_view_state_t*
3. instance_state - *dds_instance_state_t*
4. valid_data - indicates whether there is a data associated with a sample
 - true, indicates the data is valid
 - false, indicates the data is invalid, no data to read
5. source_timestamp - timestamp of a data instance when it is written
6. instance_handle - handle to the data instance
7. publication_handle - handle to the publisher
8. disposed_generation_count - count of instance state change from NOT_ALIVE_DISPOSED to ALIVE
9. no_writers_generation_count - count of instance state change from NOT_ALIVE_NO_WRITERS to ALIVE
10. sample_rank - indicates the number of samples of the same instance that follow the current one in the collection
11. generation_rank - difference in generations between the sample and most recent sample of the same instance that appears in the returned collection
12. absolute_generation_rank - difference in generations between the sample and most recent sample of the same instance when read/take was called
13. reception_timestamp - timestamp of a data instance when it is added to a read queue

typedef bool (*dds_querycondition_filter_fn) (const void *sample)

Creates a querycondition associated to the given reader.

The querycondition allows specifying which samples are of interest in a data reader's history, by means of a mask and a filter. The mask is or'd with the flags that are dds_sample_state_t, dds_view_state_t and dds_instance_state_t.

TODO: Explain the filter (aka expression & parameters) of the (to be implemented) new querycondition implementation.

Based on the mask value set and data that matches the filter, the querycondition gets triggered when data is available on the reader.

Waitsets allow waiting for an event on some of any set of entities. This means that the querycondition can be used to wake up a waitset when data is in the reader history with states that matches the given mask and filter.

TODO: Update parameters when new querycondition is introduced.

Note The parent reader and every of its associated conditions (whether they are readconditions or queryconditions) share the same resources. This means that one of these entities reads or takes data, the states of the data will change for other entities automatically. For instance, if one reads a sample, then the sample state will become 'read' for all associated reader/conditions. Or if one takes a sample, then it's not available to any other associated reader/condition.

Return >0 - Success (valid condition).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *reader*: Reader to associate the condition to.
- *mask*: Interest (dds_sample_state_t | dds_view_state_t | dds_instance_state_t).
- *filter*: Callback that the application can use to filter specific samples.

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

```
typedef void *dds_attach_t
```

Waitset attachment argument.

Every entity that is attached to the waitset can be accompanied by such an attachment argument. When the waitset wait is unblocked because of an entity that triggered, then the returning array will be populated with these attachment arguments that are related to the triggered entity.

Enums

```
enum dds_sample_state
```

defines the state for a data value

1. DDS_SST_READ - DataReader has already accessed the sample by read
2. DDS_SST_NOT_READ - DataReader has not accessed that sample before

dds_sample_state_t

Values:

```
DDS_SST_READ = 1u
```

DataReader has already accessed the sample by read

```
DDS_SST_NOT_READ = 2u
```

DataReader has not accessed the sample before

```
DDS_SST_READ = 1u
```

```
DDS_SST_NOT_READ = 2u
```

```
enum dds_view_state
```

defines the view state of an instance relative to the samples

1. DDS_VST_NEW - DataReader is accessing the sample for the first time when the instance is alive
2. DDS_VST_OLD - DataReader has accessed the sample before

dds_view_state_t

Values:

```
DDS_VST_NEW = 4u
```

DataReader is accessing the sample for the first time when the instance is alive

DDS_VST_OLD = 8u

DataReader accessed the sample before

DDS_VST_NEW = 4u

DDS_VST_OLD = 8u

enum dds_instance_state

defines the state of the instance

1. **DDS_IST_ALIVE** - Samples received for the instance from the live data writers
2. **DDS_IST_NOT_ALIVE_DISPOSED** - Instance was explicitly disposed by the data writer
3. **DDS_IST_NOT_ALIVE_NO_WRITERS** - Instance has been declared as not alive by data reader as there are no live data writers writing that instance

dds_instance_state_t

Values:

DDS_IST_ALIVE = 16u

Samples received for the instance from the live data writers

DDS_IST_NOT_ALIVE_DISPOSED = 32u

Instance was explicitly disposed by the data writer

DDS_IST_NOT_ALIVE_NO_WRITERS = 64u

Instance has been declared as not alive by data reader as there are no live data writers writing that instance

DDS_IST_ALIVE = 16u

DDS_IST_NOT_ALIVE_DISPOSED = 32u

DDS_IST_NOT_ALIVE_NO_WRITERS = 64u

Functions

typedef _Return_type_success_(return >= 0)

typedef _Return_type_success_(return, 0)

DDS_EXPORT dds_domainid_t dds_get_default_domainid(void)

Description : Returns the default DDS domain id. This can be configured in xml or set as an environment variable (VORTEX_DOMAIN).

Arguments :

1. None
2. Returns the default domain id

_Pre_satisfies_(entity &0x7F000000)

Enable entity.

Get the domain id to which this entity is attached.

Get entity children.

Get entity participant.

Get entity parent.

Set entity listeners.

Get entity listeners.

Set entity QoS policies.

Get entity QoS policies.

Delete given entity.

This operation enables the `dds_entity_t`. Created `dds_entity_t` objects can start in either an enabled or disabled state. This is controlled by the value of the entityfactory policy on the corresponding parent entity for the given entity. Enabled entities are immediately activated at creation time meaning all their immutable QoS settings can no longer be changed. Disabled Entities are not yet activated, so it is still possible to change their immutable QoS settings. However, once activated the immutable QoS settings can no longer be changed. Creating disabled entities can make sense when the creator of the `DDS_Entity` does not yet know which QoS settings to apply, thus allowing another piece of code to set the QoS later on.

Note Delayed entity enabling is not supported yet (CHAM-96).

The default setting of `DDS_EntityFactoryQosPolicy` is such that, by default, entities are created in an enabled state so that it is not necessary to explicitly call `dds_enable` on newly-created entities.

The `dds_enable` operation produces the same results no matter how many times it is performed. Calling `dds_enable` on an already enabled `DDS_Entity` returns `DDS_RETCODE_OK` and has no effect.

If an Entity has not yet been enabled, the only operations that can be invoked on it are: the ones to set, get or copy the `QosPolicy` settings, the ones that set (or get) the Listener, the ones that get the Status and the `dds_get_status_changes` operation (although the status of a disabled entity never changes). Other operations will return the error `DDS_RETCODE_NOT_ENABLED`.

Entities created with a parent that is disabled, are created disabled regardless of the setting of the entity-factory policy.

Calling `dds_enable` on an Entity whose parent is not enabled will fail and return `DDS_RETCODE_PRECONDITION_NOT_MET`.

If the entityfactory policy has `autoenable_created_entities` set to `TRUE`, the `dds_enable` operation on the parent will automatically enable all child entities created with the parent.

The Listeners associated with an Entity are not called until the Entity is enabled. Conditions associated with an Entity that is not enabled are “inactive”, that is, have a `trigger_value` which is `FALSE`.

This operation will delete the given entity. It will also automatically delete all its children, childrens' children, etc entities.

Return 0 - Success (`DDS_RETCODE_OK`).

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `e`: The entity to enable.

Return Value

- `DDS_RETCODE_OK`: The listeners of to the entity have been successfully been copied into the specified listener parameter.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The parent of the given Entity is not enabled.

TODO: Link to generic dds entity relations documentation.

Description : Read the status(es) set for the entity based on the enabled status and mask set. This operation does not clear the read status(es).

Return 0 - Success (DDS_RETCODE_OK).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- `entity`: Entity from which to get its parent.

Return Value

- DDS_RETCODE_ERROR: The entity and its children (recursive are deleted).
- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

Arguments :

1. `e` Entity on which the status has to be read
2. `status` Returns the status set on the entity, based on the enabled status
3. `mask` Filter the status condition to be read (can be NULL)
4. Returns 0 on success, or a non-zero error value if the mask does not correspond to the entity

Description : Read the status(es) set for the entity based on the enabled status and mask set. This operation clears the status set after reading.

Arguments :

1. `e` Entity on which the status has to be read
2. `status` Returns the status set on the entity, based on the enabled status
3. `mask` Filter the status condition to be read (can be NULL)
4. Returns 0 on success, or a non-zero error value if the mask does not correspond to the entity

Description : Returns the status changes since they were last read.

Arguments :

1. `e` Entity on which the statuses are read
2. Returns the current set of triggered statuses.

Description : This operation returns the status enabled on the entity

Arguments :

1. `e` Entity to get the status
2. Returns the status that are enabled for the entity

Description : This operation enables the status(es) based on the mask set

Arguments :

1. `e` Entity to enable the status
2. `mask` Status value that indicates the status to be enabled

3. Returns 0 on success, or a non-zero error value indicating failure if the mask does not correspond to the entity.

This operation allows access to the existing set of QoS policies for the entity.

TODO: Link to generic QoS information documentation.

This operation replaces the existing set of QoS Policy settings for an entity. The parameter qos must contain the struct with the QoSPolicy settings which is checked for self-consistency.

Return 0 - Success (DDS_RETCODE_OK).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- e: Entity on which to get qos
- qos: Pointer to the qos structure that returns the set policies

Return Value

- DDS_RETCODE_OK: The existing set of QoS policy values applied to the entity has successfully been copied into the specified qos parameter.
- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER: The qos parameter is NULL.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

The set of QoSPolicy settings specified by the qos parameter are applied on top of the existing QoS, replacing the values of any policies previously set (provided, the operation returned DDS_RETCODE_OK).

Not all policies are changeable when the entity is enabled.

TODO: Link to generic QoS information documentation.

This operation allows access to the existing listeners attached to the entity.

Note Currently only Latency Budget and Ownership Strength are changeable QoS that can be set.

Return 0 - Success (DDS_RETCODE_OK).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- e: Entity from which to get qos
- qos: Pointer to the qos structure that provides the policies

Return Value

- DDS_RETCODE_OK: The new QoS policies are set.
- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER: The qos parameter is NULL.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.
- DDS_RETCODE_IMMUTABLE_POLICY: The entity is enabled and one or more of the policies of the QoS are immutable.

- `DDS_RETCODE_INCONSISTENT_POLICY`: A few policies within the QoS are not consistent with each other.

TODO: Link to (generic) Listener and status information.

This operation attaches a `dds_listener_t` to the `dds_entity_t`. Only one Listener can be attached to each Entity. If a Listener was already attached, this operation will replace it with the new one. In other words, all related callbacks are replaced (possibly with NULL).

Return 0 - Success (`DDS_RETCODE_OK`).

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `e`: Entity on which to get the listeners
- `listener`: Pointer to the listener structure that returns the set of listener callbacks.

Return Value

- `DDS_RETCODE_OK`: The listeners of to the entity have been successfully been copied into the specified listener parameter.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The listener parameter is NULL.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

When listener parameter is NULL, all listener callbacks that were possibly set on the Entity will be removed.

TODO: Link to (generic) Listener and status information.

Note Not all listener callbacks are related to all entities.

For each communication status, the `StatusChangedFlag` flag is initially set to FALSE. It becomes TRUE whenever that plain communication status changes. For each plain communication status activated in the mask, the associated Listener callback is invoked and the communication status is reset to FALSE, as the listener implicitly accesses the status which is passed as a parameter to that operation. The status is reset prior to calling the listener, so if the application calls the `get_<status_name>` from inside the listener it will see the status already reset.

In case a related callback within the Listener is not set, the Listener of the Parent entity is called recursively, until a Listener with the appropriate callback set has been found and called. This allows the application to set (for instance) a default behaviour in the Listener of the containing Publisher and a `DataWriter` specific behaviour when needed. In case the callback is not set in the Publishers' Listener either, the communication status will be propagated to the Listener of the `DomainParticipant` of the containing `DomainParticipant`. In case the callback is not set in the `DomainParticipants`' Listener either, the `Communication Status` flag will be set, resulting in a possible `WaitSet` trigger.

This operation returns the parent to which the given entity belongs. For instance, it will return the Participant that was used when creating a Publisher (when that Publisher was provided here).

Return 0 - Success (`DDS_RETCODE_OK`).

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `e`: Entity on which to get the listeners

- `listener`: Pointer to the listener structure that contains the set of listener callbacks (maybe NULL).

Return Value

- `DDS_RETCODE_OK`: The listeners of to the entity have been successfully been copied into the specified listener parameter.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

TODO: Link to generic dds entity relations documentation.

This operation returns the participant to which the given entity belongs. For instance, it will return the Participant that was used when creating a Publisher that was used to create a DataWriter (when that DataWriter was provided here).

Return >0 - Success (valid entity handle).

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `entity`: Entity from which to get its parent.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

TODO: Link to generic dds entity relations documentation.

This operation returns the children that the entity contains. For instance, it will return all the Topics, Publishers and Subscribers of the Participant that was used to create those entities (when that Participant is provided here).

Return >0 - Success (valid entity handle).

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `entity`: Entity from which to get its participant.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This functions takes a pre-allocated list to put the children in and will return the number of found children. It is possible that the given size of the list is not the same as the number of found children. If less children are found, then the last few entries in the list are untouched. When more children are found, then only 'size' number of entries are inserted into the list, but still complete count of the found children is returned. Which children are returned in the latter case is undefined.

When supplying NULL as list and 0 as size, you can use this to acquire the number of children without having to pre-allocate a list.

TODO: Link to generic dds entity relations documentation.

When creating a participant entity, it is attached to a certain domain. All the children (like Publishers) and childrens' children (like DataReaders), etc are also attached to that domain.

Return ≥ 0 - Success (number of found children, can be larger than 'size').

Return < 0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *entity*: Entity from which to get its children.
- *children*: Pre-allocated array to contain the found children.
- *size*: Size of the pre-allocated children's list.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The children parameter is NULL, while a size is provided.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This function will return the original domain ID when called on any of the entities within that hierarchy.

Description : Checks whether the entity has one of its enabled statuses triggered.

Return 0 - Success (`DDS_RETCODE_OK`).

Return < 0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *entity*: Entity from which to get its children.
- *id*: Pointer to put the domain ID in.

Return Value

- `DDS_RETCODE_OK`: Domain ID was returned.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The id parameter is NULL.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

Arguments :

1. *e* Entity for which to check for triggered status

`_Pre_satisfies_((writer & (0x7F000000)) == DDS_KIND_WRITER)`

Get entity publisher.

This operation returns the publisher to which the given entity belongs. For instance, it will return the Publisher that was used when creating a DataWriter (when that DataWriter was provided here).

TODO: Link to generic dds entity relations documentation.

Return > 0 - Success (valid entity handle).

Return < 0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *entity*: Entity from which to get its publisher.

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

_Pre_satisfies_(((entity &(0x7F000000))==DDS_KIND_READER) || ((entity &(0x7F000000))==DDS_KIND_COND_READ))
Get entity subscriber.

This operation returns the subscriber to which the given entity belongs. For instance, it will return the Subscriber that was used when creating a DataReader (when that DataReader was provided here).

TODO: Link to generic dds entity relations documentation.

Return >0 - Success (valid entity handle).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *entity*: Entity from which to get its subscriber.

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

_Pre_satisfies_(((condition &(0x7F000000))==DDS_KIND_COND_READ) || ((condition &(0x7F000000))==DDS_KIND_READER))
Get entity datareader.

Get the mask of a condition.

This operation returns the datareader to which the given entity belongs. For instance, it will return the DataReader that was used when creating a ReadCondition (when that ReadCondition was provided here).

TODO: Link to generic dds entity relations documentation.

This operation returns the mask that was used to create the given condition.

Return >0 - Success (valid entity handle).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *entity*: Entity from which to get its datareader.

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

Return 0 - Success (given mask is set).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *condition*: Read or Query condition that has a mask.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The mask arg is NULL.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

DDS_EXPORT Must_inspect_result_ dds_entity_t dds_create_participant(_In_ const dds_domainid_t domain_id, const dds_qos_t qos, const dds_listener_t listener)
Creates a new instance of a DDS participant in a domain.

If domain is set (not `DDS_DOMAIN_DEFAULT`) then it must match if the domain has also been configured or an error status will be returned. Currently only a single domain can be configured by setting the environment variable `VORTEX_DOMAIN`, if this is not set the the default domain is 0. Valid values for domain id are between 0 and 230.

Return >0 - Success (valid handle of a participant entity).

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `domain`: - The domain in which to create the participant (can be `DDS_DOMAIN_DEFAULT`)
- `qos`: - The QoS to set on the new participant (can be NULL)
- `listener`: - Any listener functions associated with the new participant (can be NULL)

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.

DDS_EXPORT Check_return_ dds_return_t dds_lookup_participant(_In_ dds_domainid_t domain_id, dds_participant_t participants, size_t size)
Get participants of a domain.

This operation acquires the participants created on a domain and returns the number of found participants.

This function takes a domain id with the size of pre-allocated participant's list in and will return the number of found participants. It is possible that the given size of the list is not the same as the number of found participants. If less participants are found, then the last few entries in an array stay untouched. If more participants are found and the array is too small, then the participants returned are undefined.

Return >=0 - Success (number of found participants).

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `domain_id`: The domain id
- `participants`: The participant for domain
- `size`: Size of the pre-allocated participant's list.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The participant parameter is NULL, while a size is provided.

_Pre_satisfies_ ((participant & (0x7F000000)) == DDS_KIND_PARTICIPANT)

Creates a new instance of a DDS subscriber.

Create a waitset and allocate the resources required.

Creates a new instance of a DDS publisher.

Description : Creates a new DDS topic. The type name for the topic is taken from the generated descriptor. Topic matching is done on a combination of topic name and type name.

Arguments :

1. pp The participant on which the topic is being created
2. descriptor The IDL generated topic descriptor
3. name The name of the created topic
4. qos The QoS to set on the new topic (can be NULL)
5. listener Any listener functions associated with the new topic (can be NULL)
6. Returns a status, 0 on success or non-zero value to indicate an error

Description : Finds a named topic. Returns NULL if does not exist. The returned topic should be released with `dds_delete`.

Arguments :

1. pp The participant on which to find the topic
2. name The name of the topic to find
3. Returns a topic, NULL if could not be found or error

A WaitSet object allows an application to wait until one or more of the conditions of the attached entities evaluates to TRUE or until the timeout expires.

Return >0 - Success (valid handle of a subscriber entity).

Return <0 - Failure (use `dds_err_nr()` to get error value).

Return >0 - Success (valid handle of a publisher entity).

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- participant: The participant on which the subscriber is being created
- qos: The QoS to set on the new subscriber (can be NULL)
- listener: Any listener functions associated with the new subscriber (can be NULL)

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER One of the parameters is invalid

Parameters

- participant: The participant to create a publisher for
- qos: The QoS to set on the new publisher (can be NULL)
- listener: Any listener functions associated with the new publisher (can be NULL)

Return >0 - Success (valid waitset).

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- participant: Domain participant which the WaitSet contains.

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

`_Pre_satisfies_((topic &(0x7F000000)) == DDS_KIND_TOPIC)`

Description : Returns a topic name.

Arguments :

1. topic The topic
2. Returns The topic name or NULL to indicate an error

Description : Returns a topic type name.

Arguments :

1. topic The topic
2. Returns The topic type name or NULL to indicate an error

`_Out_writes_z_ (size)`

`_Pre_satisfies_((publisher &(0x7F000000)) == DDS_KIND_PUBLISHER)`

Suspends the publications of the Publisher.

Resumes the publications of the Publisher.

This operation is a hint to the Service so it can optimize its performance by e.g., collecting modifications to DDS writers and then batching them. The Service is not required to use the hint.

Every invocation of this operation must be matched by a corresponding call to This operation is a hint to the Service to indicate that the application has completed changes initiated by a previous The call to resume_publications must match a previous call to

See `dds_resume` indicating that the set of modifications has completed.

Return >0 - Success.

Return <0 - Failure (use `dds_err_nr()` to get error value).

See `suspend`. The Service is not required to use the hint.

See `suspend_publications`.

Return >0 - Success.

Return <0 - Failure (use `dds_err_nr()` to get error value).

See `dds_suspend`.

Parameters

- publisher: The publisher for which all publications will be suspended

Return Value

- DDS_RETCODE_OK: Publications suspended successfully.
- DDS_RETCODE_BAD_PARAMETER: The pub parameter is not a valid publisher.
- DDS_RETCODE_UNSUPPORTED: Operation is not supported

Parameters

- publisher: The publisher for which all publications will be resumed

Return Value

- DDS_RETCODE_OK: Publications resumed successfully.
- DDS_RETCODE_BAD_PARAMETER: The pub parameter is not a valid publisher.
- DDS_RETCODE_PRECONDITION_NOT_MET: No previous matching

Return Value

- DDS_RETCODE_UNSUPPORTED: Operation is not supported.

`_Pre_satisfies_((publisher_or_writer &(0x7F000000))==DDS_KIND_WRITER) || (publisher_or`

Waits at most for the duration timeout for acks for data in the publisher or writer.

This operation blocks the calling thread until either all data written by the publisher or writer is acknowledged by all matched reliable reader entities, or else the duration specified by the timeout parameter elapses, whichever happens first.

Return >0 - Success.

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- pub_or_w: The publisher or writer whose acknowledgements must be waited for.

Return Value

- DDS_RETCODE_OK: All acknowledgements successfully received with the timeout.
- DDS_RETCODE_BAD_PARAMETER: The pub_or_w parameter is not a valid publisher or writer.
- DDS_RETCODE_TIMEOUT: Timeout expired before all acknowledgements from reliable reader entities were received.
- DDS_RETCODE_UNSUPPORTED: Operation is not supported.

`_Pre_satisfies_((participant_or_subscriber &(0x7F000000))==DDS_KIND_SUBSCRIBER) || (pa`

Creates a new instance of a DDS reader.

Return >0 - Success (valid handle of a reader entity)

Return <0 - Failure (use *dds_err_nr()* to get error value)

Parameters

- participant_or_subscriber: The participant or subscriber on which the reader is being created
- topic: The topic to read
- qos: The QoS to set on the new reader (can be NULL)
- listener: Any listener functions associated with the new reader (can be NULL)

`_Pre_satisfies_(reader &(0x7F000000)) == DDS_KIND_READER)`

Creates a readcondition associated to the given reader.

Description : The operation blocks the calling thread until either all “historical” data is received, or else the duration specified by the max_wait parameter elapses, whichever happens first. A return value of 0 indicates that all the “historical” data was received; a return value of TIMEOUT indicates that max_wait elapsed before all the data was received.

Arguments :

1. reader The reader on which to wait for historical data
2. max_wait How long to wait for historical data before time out

3. Returns a status, 0 on success, TIMEOUT on timeout or a negative value to indicate error

The readcondition allows specifying which samples are of interest in a data reader's history, by means of a mask. The mask is or'd with the flags that are `dds_sample_state_t`, `dds_view_state_t` and `dds_instance_state_t`.

Based on the mask value set, the readcondition gets triggered when data is available on the reader.

Waitsets allow waiting for an event on some of any set of entities. This means that the readcondition can be used to wake up a waitset when data is in the reader history with states that matches the given mask.

Note The parent reader and every of its associated conditions (whether they are readconditions or queryconditions) share the same resources. This means that one of these entities reads or takes data, the states of the data will change for other entities automatically. For instance, if one reads a sample, then the sample state will become 'read' for all associated reader/conditions. Or if one takes a sample, then it's not available to any other associated reader/condition.

Return >0 - Success (valid condition).

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader`: Reader to associate the condition to.
- `mask`: Interest (`dds_sample_state_t``dds_view_state_t``dds_instance_state_t`).

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

`_Pre_satisfies((participant_or_publisher & (0x7F000000)) == DDS_KIND_PUBLISHER) || ((part`
Creates a new instance of a DDS writer.

Return >0 - Success (valid handle of a writer entity)

Return <0 - Failure (use `dds_err_nr()` to get error value)

Parameters

- `participant_or_publisher`: The participant or publisher on which the writer is being created
- `topic`: The topic to write
- `qos`: The QoS to set on the new writer (can be NULL)
- `listener`: Any listener functions associated with the new writer (can be NULL)

`_Pre_satisfies((writer & (0x7F000000)) == DDS_KIND_WRITER)`

This operation modifies and disposes a data instance.

Write the value of a data instance along with the source timestamp passed.

Write a CDR serialized value of a data instance.

Write the value of a data instance.

This operation disposes an instance, identified by the instance handle.

This operation disposes an instance, identified by the data sample.

Description : Registers an instance with a key value to the data writer

Arguments :

1. wr The writer to which instance has be associated
2. data Instance with the key value
3. Returns an instance handle that could be used for successive write & dispose operations or NULL, if handle is not allocated

Description : Unregisters an instance with a key value from the data writer. Instance can be identified either from data sample or from instance handle (at least one must be provided).

Arguments :

1. wr The writer to which instance is associated
2. data Instance with the key value (can be NULL if handle set)
3. handle Instance handle (can be DDS_HANDLE_NIL if data set)
4. Returns 0 on success, or non-zero value to indicate an error

Note : If an unregistered key ID is passed as instance data, an error is logged and not flagged as return value

Description : Unregisters an instance with a key value from the data writer. Instance can be identified either from data sample or from instance handle (at least one must be provided).

Arguments :

1. wr The writer to which instance is associated
2. data Instance with the key value (can be NULL if handle set)
3. handle Instance handle (can be DDS_HANDLE_NIL if data set)
4. timestamp used at registration.
5. Returns 0 on success, or non-zero value to indicate an error

Note : If an unregistered key ID is passed as instance data, an error is logged and not flagged as return value

This operation requests the Data Distribution Service to modify the instance and mark it for deletion. Copies of the instance and its corresponding samples, which are stored in every connected reader and, dependent on the QoS policy settings (also in the Transient and Persistent stores) will be modified and marked for deletion by setting their `dds_instance_state_t` to `DDS_IST_NOT_ALIVE_DISPOSED`.

If the history QoS policy is set to `DDS_HISTORY_KEEP_ALL`, the `dds_writedispose` operation on the writer may block if the modification would cause data to be lost because one of the limits, specified in the `resource_limits` QoS policy, to be exceeded. In case the synchronous attribute value of the reliability QoS policy is set to true for communicating writers and readers then the writer will wait until all synchronous readers have acknowledged the data. Under these circumstances, the `max_blocking_time` attribute of the reliability QoS policy configures the maximum time the `dds_writedispose` operation may block. If `max_blocking_time` elapses before the writer is able to store the modification without exceeding the limits and all expected acknowledgements are received, the `dds_writedispose` operation will fail and returns `DDS_RETCODE_TIMEOUT`.

Description : This operation modifies and disposes a data instance with a specific timestamp.

Return 0 - Success.

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `writer`: The writer to dispose the data instance from.
- `data`: The data to be written and disposed.

Return Value

- `DDS_RETCODE_OK`: The sample is written and the instance is marked for deletion.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: At least one of the arguments is invalid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_TIMEOUT`: Either the current action overflowed the available resources as specified by the combination of the reliability QoS policy, history QoS policy and resource_limits QoS policy, or the current action was waiting for data delivery acknowledgement by synchronous readers. This caused blocking of this operation, which could not be resolved before `max_blocking_time` of the reliability QoS policy elapsed.

This operation performs the same functions as `dds_writedispose` except that the application provides the value for the `source_timestamp` that is made available to connected reader objects. This timestamp is important for the interpretation of the `destination_order` QoS policy.

This operation requests the Data Distribution Service to modify the instance and mark it for deletion. Copies of the instance and its corresponding samples, which are stored in every connected reader and, dependent on the QoS policy settings (also in the Transient and Persistent stores) will be modified and marked for deletion by setting their `dds_instance_state_t` to `DDS_IST_NOT_ALIVE_DISPOSED`.

Return 0 - Success.

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `writer`: The writer to dispose the data instance from.
- `data`: The data to be written and disposed.
- `timestamp`: The timestamp used as source timestamp.

Return Value

- `DDS_RETCODE_OK`: The sample is written and the instance is marked for deletion.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: At least one of the arguments is invalid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_TIMEOUT`: Either the current action overflowed the available resources as specified by the combination of the reliability QoS policy, history QoS policy and resource_limits QoS policy, or the current action was waiting for data delivery acknowledgement by synchronous readers. This caused blocking of this operation, which could not be resolved before `max_blocking_time` of the reliability QoS policy elapsed.

If the history QoS policy is set to `DDS_HISTORY_KEEP_ALL`, the `dds_writedispose` operation on the writer may block if the modification would cause data to be lost because one of the limits, specified in the `resource_limits` QoS policy, to be exceeded. In case the synchronous attribute value of the reliability QoS policy is set to true for communicating writers and readers then the writer will wait until all synchronous readers have acknowledged the data. Under these circumstances, the `max_blocking_time`

attribute of the reliability QoS policy configures the maximum time the `dds_writedispose` operation may block. If `max_blocking_time` elapses before the writer is able to store the modification without exceeding the limits and all expected acknowledgements are received, the `dds_writedispose` operation will fail and returns `DDS_RETCODE_TIMEOUT`.

Description : This operation disposes an instance with a specific timestamp, identified by the data sample.

Return 0 - Success.

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `writer`: The writer to dispose the data instance from.
- `data`: The data sample that identifies the instance to be disposed.

Return Value

- `DDS_RETCODE_OK`: The sample is written and the instance is marked for deletion.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: At least one of the arguments is invalid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_TIMEOUT`: Either the current action overflowed the available resources as specified by the combination of the reliability QoS policy, history QoS policy and resource_limits QoS policy, or the current action was waiting for data delivery acknowledgement by synchronous readers. This caused blocking of this operation, which could not be resolved before `max_blocking_time` of the reliability QoS policy elapsed.

This operation performs the same functions as `dds_dispose` except that the application provides the value for the `source_timestamp` that is made available to connected reader objects. This timestamp is important for the interpretation of the `destination_order` QoS policy.

This operation requests the Data Distribution Service to modify the instance and mark it for deletion. Copies of the instance and its corresponding samples, which are stored in every connected reader and, dependent on the QoS policy settings (also in the Transient and Persistent stores) will be modified and marked for deletion by setting their `dds_instance_state_t` to `DDS_IST_NOT_ALIVE_DISPOSED`.

Return 0 - Success.

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `writer`: The writer to dispose the data instance from.
- `data`: The data sample that identifies the instance to be disposed.
- `timestamp`: The timestamp used as source timestamp.

Return Value

- `DDS_RETCODE_OK`: The sample is written and the instance is marked for deletion.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: At least one of the arguments is invalid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

- `DDS_RETCODE_TIMEOUT`: Either the current action overflowed the available resources as specified by the combination of the reliability QoS policy, history QoS policy and resource_limits QoS policy, or the current action was waiting for data delivery acknowledgement by synchronous readers. This caused blocking of this operation, which could not be resolved before `max_blocking_time` of the reliability QoS policy elapsed.

The given instance handle must correspond to the value that was returned by either the `dds_register_instance` operation, `dds_register_instance_ts` or `dds_instance_lookup`. If there is no correspondence, then the result of the operation is unspecified.

Description : This operation disposes an instance with a specific timestamp, identified by the instance handle.

Return 0 - Success.

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `writer`: The writer to dispose the data instance from.
- `handle`: The handle to identify an instance.

Return Value

- `DDS_RETCODE_OK`: The sample is written and the instance is marked for deletion.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: At least one of the arguments is invalid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this writer.

This operation performs the same functions as `dds_dispose_ih` except that the application provides the value for the `source_timestamp` that is made available to connected reader objects. This timestamp is important for the interpretation of the `destination_order` QoS policy.

With this API, the value of the source timestamp is automatically made available to the data reader by the service.

Return 0 - Success.

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `writer`: The writer to dispose the data instance from.
- `handle`: The handle to identify an instance.
- `timestamp`: The timestamp used as source timestamp.

Return Value

- `DDS_RETCODE_OK`: The sample is written and the instance is marked for deletion.
- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: At least one of the arguments is invalid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this writer.

Untyped API, which take serialized blobs now. Whether they remain exposed like this with X-types isn't entirely clear yet. TODO: make a decide about `dds_takecdr`

Return - `dds_return_t` indicating success or failure

Parameters

- `writer`: The writer entity
- `data`: Value to be written

Return - A `dds_return_t` indicating success or failure

Return - A `dds_return_t` indicating success or failure

Parameters

- `writer`: The writer entity
- `cdr`: CDR serialized value to be written
- `size`: Size (in bytes) of CDR encoded data to be written

Parameters

- `writer`: The writer entity
- `data`: Value to be written
- `timestamp`: Source timestamp

`_In_reads_bytes_` (size) **const**

`_Pre_satisfies_` ((waitset & (0x7F000000)) == `DDS_KIND_WAITSET`)

Acquire previously attached entities.

This operation allows an application thread to wait for the a status change or other trigger on (one of) the entities that are attached to the WaitSet.

Sets the `trigger_value` associated with a waitset.

This operation detaches an Entity to the WaitSet.

This operation attaches an Entity to the WaitSet.

This functions takes a pre-allocated list to put the entities in and will return the number of found entities. It is possible that the given size of the list is not the same as the number of found entities. If less entities are found, then the last few entries in the list are untouched. When more entities are found, then only 'size' number of entries are inserted into the list, but still the complete count of the found entities is returned. Which entities are returned in the latter case is undefined.

This operation attaches an Entity to the WaitSet. The `dds_waitset_wait()` will block when none of the attached entities are triggered. 'Triggered' (`dds_triggered()`) doesn't mean the same for every entity:

- Reader/Writer/Publisher/Subscriber/Topic/Participant
 - These are triggered when their status changed.
- WaitSet
 - Triggered when trigger value was set to true by the application. It stays triggered until application sets the trigger value to false (`dds_waitset_set_trigger()`). This can be used to wake up an waitset for different reasons (f.i. termination) than the 'normal' status change (like new data).

- ReadCondition/QueryCondition
 - Triggered when data is available on the related Reader that matches the Condition.

Return ≥ 0 - Success (number of found children, can be larger than 'size').

Return < 0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- waitset: Waitset from which to get its attached entities.
- entities: Pre-allocated array to contain the found entities.
- size: Size of the pre-allocated entities' list.

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER: The entities parameter is NULL, while a size is provided.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The waitset has already been deleted.

Multiple entities can be attached to a single waitset. A particular entity can be attached to multiple waitsets. However, a particular entity can not be attached to a particular waitset multiple times.

When the waitset is attached to itself and the trigger value is set to 'true', then the waitset will wake up just like with an other status change of the attached entities.

Return 0 - Success (entity attached).

Return < 0 - Failure (use *dds_err_nr()* to get error value).

Return 0 - Success (entity attached).

Return < 0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- waitset: The waitset to attach the given entity to.
- entity: The entity to attach.
- x: Blob that will be supplied when the waitset wait is triggered by the given entity.

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER: The given waitset or entity are not valid.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The waitset has already been deleted.
- DDS_RETCODE_PRECONDITION_NOT_MET: The entity was already attached.

Parameters

- waitset: The waitset to detach the given entity from.
- entity: The entity to detach.

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.

- `DDS_RETCODE_BAD_PARAMETER`: The given waitset or entity are not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The waitset has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The entity is not attached.

This can be used to forcefully wake up a waitset, for instance when the application wants to shut down. So, when the trigger value is true, the waitset will wake up or not wait at all.

The trigger value will remain true until the application sets it false again deliberately.

The “`dds_waitset_wait`” operation blocks until the some of the attached entities have triggered or “retime-out” has elapsed. ‘Triggered’ (`dds_triggered()`) doesn’t mean the same for every entity:

- Reader/Writer/Publisher/Subscriber/Topic/Participant
 - These are triggered when their status changed.
- WaitSet
 - Triggered when trigger value was set to true by the application. It stays triggered until application sets the trigger value to false (`dds_waitset_set_trigger()`). This can be used to wake up an waitset for different reasons (f.i. termination) than the ‘normal’ status change (like new data).
- ReadCondition/QueryCondition
 - Triggered when data is available on the related Reader that matches the Condition.

Return 0 - Success (entity attached).

Return <0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `waitset`: The waitset to set the trigger value on.
- `trigger`: The trigger value to set.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The given waitset is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The waitset has already been deleted.

This functions takes a pre-allocated list to put the “xs” blobs in (that were provided during the attach of the related entities) and will return the number of triggered entities. It is possible that the given size of the list is not the same as the number of triggered entities. If less entities were triggered, then the last few entries in the list are untouched. When more entities are triggered, then only ‘size’ number of entries are inserted into the list, but still the complete count of the triggered entities is returned. Which “xs” blobs are returned in the latter case is undefined.

In case of a time out, the return value is 0.

Deleting the waitset while the application is blocked results in an error code (i.e. < 0) returned by “wait”.

Multiple threads may block on a single waitset at the same time; the calls are entirely independent.

An empty waitset never triggers (i.e., `dds_waitset_wait` on an empty waitset is essentially equivalent to a sleep).

The “dds_waitset_wait_until” operation is the same as the “dds_waitset_wait” except that it takes an absolute timeout.

The “dds_waitset_wait” operation blocks until the some of the attached entities have triggered or “abstime-out” has been reached. ‘Triggered’ (dds_triggered()) doesn’t mean the same for every entity:

- Reader/Writer/Publisher/Subscriber/Topic/Participant
 - These are triggered when their status changed.
- WaitSet
 - Triggered when trigger value was set to true by the application. It stays triggered until application sets the trigger value to false (dds_waitset_set_trigger()). This can be used to wake up an waitset for different reasons (f.i. termination) than the ‘normal’ status change (like new data).
- ReadCondition/QueryCondition
 - Triggered when data is available on the related Reader that matches the Condition.

Return >0 - Success (number of entities triggered).

Return 0 - Time out (no entities were triggered).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *waitset*: The waitset to set the trigger value on.
- *xs*: Pre-allocated list to store the ‘blobs’ that were provided during the attach of the triggered entities.
- *nxs*: The size of the pre-allocated blobs list.
- *reltimeout*: Relative timeout

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: The given waitset is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The waitset has already been deleted.

This functions takes a pre-allocated list to put the “xs” blobs in (that were provided during the attach of the related entities) and will return the number of triggered entities. It is possible that the given size of the list is not the same as the number of triggered entities. If less entities were triggered, then the last few entries in the list are untouched. When more entities are triggered, then only ‘size’ number of entries are inserted into the list, but still the complete count of the triggered entities is returned. Which “xs” blobs are returned in the latter case is undefined.

In case of a time out, the return value is 0.

Deleting the waitset while the application is blocked results in an error code (i.e. < 0) returned by “wait”.

Multiple threads may block on a single waitset at the same time; the calls are entirely independent.

An empty waitset never triggers (i.e., dds_waitset_wait on an empty waitset is essentially equivalent to a sleep).

The “dds_waitset_wait” operation is the same as the “dds_waitset_wait_until” except that it takes an relative timeout.

The “dds_waitset_wait” operation is the same as the “dds_wait” except that it takes an absolute timeout.

Return >0 - Success (number of entities triggered).

Return 0 - Time out (no entities were triggered).

Return <0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *waitset*: The waitset to set the trigger value on.
- *xs*: Pre-allocated list to store the ‘blobs’ that were provided during the attach of the triggered entities.
- *nxs*: The size of the pre-allocated blobs list.
- *abstimeout*: Absolute timeout

Return Value

- *DDS_RETCODE_ERROR*: An internal error has occurred.
- *DDS_RETCODE_BAD_PARAMETER*: The given waitset is not valid.
- *DDS_RETCODE_ILLEGAL_OPERATION*: The operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED*: The waitset has already been deleted.

_Out_writes_to_ (size)

_Out_writes_to_ (nxs)

_Pre_satisfies_ ((*(reader_or_condition &(0x7F000000)) == DDS_KIND_READER*) || (*(reader_or_co*

Access and read the collection of data values (of same type) and sample info from the data reader, readcondition or querycondition.

Return loaned samples to data-reader or condition associated with a data-reader.

Access loaned samples of data reader, readcondition or querycondition based on mask and scoped by the given instance handle.

Take the collection of data values (of same type) and sample info from the data reader, readcondition or querycondition based on mask and scoped by the given instance handle.

Access loaned samples of data reader, readcondition or querycondition, scoped by the given instance handle.

Access the collection of data values (of same type) and sample info from the data reader, readcondition or querycondition but scoped by the given instance handle.

Access loaned samples of data reader, readcondition or querycondition based on mask.

Take the collection of data values (of same type) and sample info from the data reader, readcondition or querycondition based on mask.

Access loaned samples of data reader, readcondition or querycondition.

Access the collection of data values (of same type) and sample info from the data reader, readcondition or querycondition.

Access and read loaned samples of data reader, readcondition or querycondition based on mask, scoped by the provided instance handle.

Read the collection of data values and sample info from the data reader, readcondition or querycondition based on mask and scoped by the provided instance handle.

Access and read loaned samples of data reader, readcondition or querycondition, scoped by the provided instance handle.

Access and read the collection of data values (of same type) and sample info from the data reader, readcondition or querycondition, copied by the provided instance handle.

Access and read loaned samples of data reader, readcondition or querycondition based on mask.

Read the collection of data values and sample info from the data reader, readcondition or querycondition based on mask.

Access and read loaned samples of data reader, readcondition or querycondition.

Return value provides information about number of samples read, which will be \leq maxs. Based on the count, the buffer will contain data to be read only when valid_data bit in sample info structure is set. The buffer required for data values, could be allocated explicitly or can use the memory from data reader to prevent copy. In the latter case, buffer and sample_info should be returned back, once it is no longer using the Data. Data values once read will remain in the buffer with the sample_state set to READ and view_state set to NOT_NEW.

After dds_read_wl function is being called and the data has been handled, dds_return_loan function must be called to possibly free memory

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- reader_or_condition: Reader, readcondition or querycondition entity
- buf: An array of pointers to samples into which data is read (pointers can be NULL)
- si: Pointer to an array of *dds_sample_info_t* returned for each data value
- bufisz: The size of buffer provided
- maxs: Maximum number of samples to read

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER: One of the given arguments is not valid.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

When using a readcondition or querycondition, their masks are or'd with the given mask.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- reader_or_condition: Reader, readcondition or querycondition entity
- buf: An array of pointers to samples into which data is read (pointers can be NULL)
- si: Pointer to an array of *dds_sample_info_t* returned for each data value
- maxs: Maximum number of samples to read

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.

- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

When using a readcondition or querycondition, their masks are or'd with the given mask.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value
- `bufsz`: The size of buffer provided
- `maxs`: Maximum number of samples to read
- `mask`: Filter the data based on `dds_sample_state_t`, `ldds_view_state_t`, `ldds_instance_state_t`.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

After `dds_read_mask_wl` function is being called and the data has been handled, `dds_return_loan` function must be called to possibly free memory

This operation implements the same functionality as `dds_read`, except that only data scoped to the provided instance handle is read.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value
- `maxs`: Maximum number of samples to read
- `mask`: Filter the data based on `dds_sample_state_t`, `ldds_view_state_t`, `ldds_instance_state_t`.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation implements the same functionality as `dds_read_wl`, except that only data scoped to the provided instance handle is read.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *reader_or_condition*: Reader, readcondition or querycondition entity
- *buf*: An array of pointers to samples into which data is read (pointers can be NULL)
- *si*: Pointer to an array of *dds_sample_info_t* returned for each data value
- *bufsz*: The size of buffer provided
- *maxs*: Maximum number of samples to read
- *handle*: Instance handle related to the samples to read

Return Value

- *DDS_RETCODE_ERROR*: An internal error has occurred.
- *DDS_RETCODE_BAD_PARAMETER*: One of the given arguments is not valid.
- *DDS_RETCODE_ILLEGAL_OPERATION*: The operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED*: The entity has already been deleted.
- *DDS_RETCODE_PRECONDITION_NOT_MET*: The instance handle has not been registered with this reader.

This operation implements the same functionality as *dds_read_mask*, except that only data scoped to the provided instance handle is read.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- *reader_or_condition*: Reader, readcondition or querycondition entity
- *buf*: An array of pointers to samples into which data is read (pointers can be NULL)
- *si*: Pointer to an array of *dds_sample_info_t* returned for each data value
- *maxs*: Maximum number of samples to read
- *handle*: Instance handle related to the samples to read

Return Value

- *DDS_RETCODE_ERROR*: An internal error has occurred.
- *DDS_RETCODE_BAD_PARAMETER*: One of the given arguments is not valid.
- *DDS_RETCODE_ILLEGAL_OPERATION*: The operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED*: The entity has already been deleted.
- *DDS_RETCODE_PRECONDITION_NOT_MET*: The instance handle has not been registered with this reader.

This operation implements the same functionality as *dds_read_mask_wl*, except that only data scoped to the provided instance handle is read.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of *dds_sample_info_t* returned for each data value
- `bufsz`: The size of buffer provided
- `maxs`: Maximum number of samples to read
- `handle`: Instance handle related to the samples to read
- `mask`: Filter the data based on `dds_sample_state_t``ldds_view_state_t``ldds_instance_state_t`.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

Description : Access the collection of data values (of same type) and sample info from the data reader based on the criteria specified in the read condition. Read condition must be attached to the data reader before associating with data read. Return value provides information about number of samples read, which will be \leq `maxs`. Based on the count, the buffer will contain data to be read only when `valid_data` bit in sample info structure is set. The buffer required for data values, could be allocated explicitly or can use the memory from data reader to prevent copy. In the latter case, buffer and sample_info should be returned back, once it is no longer using the Data. Data values once read will remain in the buffer with the `sample_state` set to `READ` and `view_state` set to `NOT_NEW`.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of *dds_sample_info_t* returned for each data value
- `maxs`: Maximum number of samples to read
- `handle`: Instance handle related to the samples to read
- `mask`: Filter the data based on `dds_sample_state_t``ldds_view_state_t``ldds_instance_state_t`.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

Arguments :

1. rd Reader entity
2. buf an array of pointers to samples into which data is read (pointers can be NULL)
3. maxs maximum number of samples to read
4. si pointer to an array of *dds_sample_info_t* returned for each data value
5. cond read condition to filter the data samples based on the content
6. Returns the number of samples read, 0 indicates no data to read. Data value once read is removed from the Data Reader cannot to 'read' or 'taken' again. Return value provides information about number of samples read, which will be \leq maxs. Based on the count, the buffer will contain data to be read only when valid_data bit in sample info structure is set. The buffer required for data values, could be allocated explicitly or can use the memory from data reader to prevent copy. In the latter case, buffer and sample_info should be returned back, once it is no longer using the Data.

After dds_take_wl function is being called and the data has been handled, dds_return_loan function must be called to possibly free memory

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- reader_or_condition: Reader, readcondition or querycondition entity
- buf: An array of pointers to samples into which data is read (pointers can be NULL)
- si: Pointer to an array of *dds_sample_info_t* returned for each data value
- bufisz: The size of buffer provided
- maxs: Maximum number of samples to read

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER: One of the given arguments is not valid.
- DDS_RETCODE_ILLEGAL_OPERATION: The operation is invoked on an inappropriate object.
- DDS_RETCODE_ALREADY_DELETED: The entity has already been deleted.

When using a readcondition or querycondition, their masks are or'd with the given mask.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use *dds_err_nr()* to get error value).

Parameters

- reader_or_condition: Reader, readcondition or querycondition entity
- buf: An array of pointers to samples into which data is read (pointers can be NULL)
- si: Pointer to an array of *dds_sample_info_t* returned for each data value
- maxs: Maximum number of samples to read

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER: One of the given arguments is not valid.

- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

After `dds_take_mask_wl` function is being called and the data has been handled, `dds_return_loan` function must be called to possibly free memory

This operation implements the same functionality as `dds_take`, except that only data scoped to the provided instance handle is taken.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value
- `maxs`: Maximum number of samples to read
- `mask`: Filter the data based on `dds_sample_state_t`, `ldds_view_state_t`, `ldds_instance_state_t`.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.

This operation implements the same functionality as `dds_take_wl`, except that only data scoped to the provided instance handle is read.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value
- `bufsz`: The size of buffer provided
- `maxs`: Maximum number of samples to read
- `handle`: Instance handle related to the samples to read

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

This operation implements the same functionality as `dds_take_mask`, except that only data scoped to the provided instance handle is read.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value
- `maxs`: Maximum number of samples to read
- `handle`: Instance handle related to the samples to read

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

This operation implements the same functionality as `dds_take_mask_wl`, except that only data scoped to the provided instance handle is read.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value
- `bufsz`: The size of buffer provided
- `maxs`: Maximum number of samples to read
- `handle`: Instance handle related to the samples to read
- `mask`: Filter the data based on `dds_sample_state_t`, `ldds_view_state_t`, `ldds_instance_state_t`.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

Used to release sample buffers returned by a read/take operation. When the application provides an empty buffer, memory is allocated and managed by DDS. By calling `dds_return_loan`, the memory is released so that the buffer can be reused during a successive read/take operation. When a condition is provided, the reader to which the condition belongs is looked up.

Return ≥ 0 - Success (number of samples read).

Return < 0 - Failure (use `dds_err_nr()` to get error value).

Parameters

- `reader_or_condition`: Reader, readcondition or querycondition entity
- `buf`: An array of pointers to samples into which data is read (pointers can be NULL)
- `si`: Pointer to an array of `dds_sample_info_t` returned for each data value
- `maxs`: Maximum number of samples to read
- `handle`: Instance handle related to the samples to read
- `mask`: Filter the data based on `dds_sample_state_t`, `dds_view_state_t`, `dds_instance_state_t`.

Return Value

- `DDS_RETCODE_ERROR`: An internal error has occurred.
- `DDS_RETCODE_BAD_PARAMETER`: One of the given arguments is not valid.
- `DDS_RETCODE_ILLEGAL_OPERATION`: The operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED`: The entity has already been deleted.
- `DDS_RETCODE_PRECONDITION_NOT_MET`: The instance handle has not been registered with this reader.

Return A `dds_return_t` indicating success or failure

Parameters

- `rd_or_cnd`: Reader or condition that belongs to a reader
- `buf`: An array of (pointers to) samples
- `bufsz`: The number of (pointers to) samples stored in `buf`

DDS_EXPORT `int dds_takecdr(dds_entity_t reader_or_condition, struct serdata ** buf, unsigned int maxs, unsigned int mask)`

DDS_EXPORT `dds_return_t dds_take_next(_In_ dds_entity_t reader_or_condition, _Out_ void ** buf, unsigned int maxs, unsigned int mask)`

Description : This operation copies the next, non-previously accessed data value and corresponding sample info and removes from the data reader.

Arguments :

1. `rd` Reader entity
2. `buf` an array of pointers to samples into which data is read (pointers can be NULL)
3. `si` pointer to `dds_sample_info_t` returned for a data value
4. Returns 1 on successful operation, else 0 if there is no data to be read.

DDS_EXPORT `dds_return_t dds_take_next_wl(_In_ dds_entity_t reader_or_condition, _Out_ void ** buf, unsigned int maxs, unsigned int mask)`

DDS_EXPORT `dds_return_t dds_read_next(_In_ dds_entity_t reader_or_condition, _Out_ void ** buf, unsigned int maxs, unsigned int mask)`

Description : This operation copies the next, non-previously accessed data value and corresponding sample info.

Arguments :

1. rd Reader entity
2. buf an array of pointers to samples into which data is read (pointers can be NULL)
3. si pointer to *dds_sample_info_t* returned for a data value
4. Returns 1 on successful operation, else 0 if there is no data to be read.

DDS_EXPORT dds_return_t dds_read_next_wl(_In_ dds_entity_t reader_or_condition, _Out_ *dds_sample_info_t* *si, _Inout_updates_ (bufsz)

DDS_EXPORT dds_return_t dds_lookup_instance(_In_ dds_entity_t entity, _Out_ dds_instance_t *inst)
Description : This operation takes a sample and returns an instance handle to be used for subsequent operations.

Arguments :

1. e Reader or Writer entity
2. data sample with a key fields set
3. Returns instance handle or DDS_HANDLE_NIL if instance could not be found from key

DDS_EXPORT dds_return_t dds_instance_get_key(_In_ dds_entity_t entity, _In_ dds_instance_t inst, _Out_ *dds_key_value_t* *key)
Description : This operation takes an instance handle and return a key-value corresponding to it.

Arguments :

1. e Reader or Writer entity
2. inst Instance handle
3. data pointer to an instance, to which the key ID corresponding to the instance handle will be returned, the sample in the instance should be ignored.
4. Returns 0 on successful operation, or a non-zero value to indicate an error if the instance passed doesn't have a key-value

_Pre_satisfies_(((entity & (0x7F000000)) == DDS_KIND_READER) || ((entity & (0x7F000000)) == DDS_KIND_WRITER))
Begin coherent publishing or begin accessing a coherent set in a subscriber.

End coherent publishing or end accessing a coherent set in a subscriber.

Invoking on a Writer or Reader behaves as if dds_begin_coherent was invoked on its parent Publisher or Subscriber respectively.

Invoking on a Writer or Reader behaves as if dds_end_coherent was invoked on its parent Publisher or Subscriber respectively.

Return - A dds_return_t indicating success or failure

Parameters

- e: - The entity that is prepared for coherent access

Return Value

- DDS_RETCODE_ERROR: An internal error has occurred.
- DDS_RETCODE_BAD_PARAMETER The provided entity is invalid or not supported

Return - A dds_return_t indicating success or failure

Parameters

- e: - The entity on which coherent access is finished

Return Value

- DDS_RETCODE_OK: The operation was successful DDS_RETCODE_BAD_PARAMETER The provided entity is invalid or not supported

`_Pre_satisfies_((subscriber &(0x7F000000)) == DDS_KIND_SUBSCRIBER)`

Trigger DATA_AVAILABLE event on contained readers.

The DATA_AVAILABLE event is broadcast to all readers owned by this subscriber that currently have new data available. Any on_data_available listener callbacks attached to respective readers are invoked.

DDS_RETCODE_OK The operation was successful DDS_RETCODE_BAD_PARAMETER The provided subscriber is invalid

Return - A dds_return_t indicating success or failure

Parameters

- sub: A subscriber

DDS_EXPORT dds_entity_t dds_get_domain(_In_ dds_domainid_t id)

Description : Resolves the domain-entity identified by id if it exists

Arguments :

1. id

DDS_EXPORT dds_return_t dds_get_matched(_In_ dds_entity_t wr_or_r, _Out_writes_to_(nofHandles, 1) dds_entity_t handles)

Description : Retrieves the matched publications (for a given Reader) or subscriptions (for a given Writer)

Arguments :

1. wr_or_r Writer or Reader
2. handles Array of size nofHandles
3. nofHandles Number of elements that can be written to in handles
4. Returns the number of available matched publications or subscriptions. If return > nofHandles the resulting set is truncated. Handles are only initialized up to min(return, nofHandles).

DDS_EXPORT dds_return_t dds_assert_liveliness(_In_ dds_entity_t e)

Description : Asserts the liveliness of the entity

Arguments :

1. e Entity

DDS_EXPORT dds_return_t dds_contains(_In_ dds_entity_t e, _In_ dds_entity_t c)

Description : Checks whether entity c is contained in entity e

Containment is defined as follows: TODO

Arguments :

1. e Entity for which has to be determined whether c is contained within it
2. c Entity to check for being contained in e

DDS_EXPORT dds_time_t dds_time(void)

Description : Returns the current wall-clock as used for timestamps

DDS_EXPORT dds_entity_t dds_create_contentfilteredtopic(_In_ dds_entity_t pp, _In_z_ const char * name)

DDS_EXPORT dds_entity_t dds_lookup_topic(_In_ dds_entity_t pp, _In_z_ const char * name)

Description : Tries to find the topic with the supplied name.

Arguments :

1. pp Participant
2. name Topic-name to look for

DDS_EXPORT dds_return_t dds_ignore(_In_ dds_entity_t pp, _In_ dds_instance_handle_t handle)
Description : Ignore the entity described by handle.

Arguments :

1. pp Participant
2. handle Instance-handle of entity to be ignored.

DDS_EXPORT dds_entity_t dds_get_related_topic(_In_ dds_entity_t cft)
Description : Retrieve the topic on which the content-filtered-topic is based

TODO: Refactor CFT

Arguments :

1. cft ContentFilteredTopic

DDS_EXPORT dds_entity_t dds_get_query(_In_ dds_entity_t top_mt_qc)
Description : Retrieve the query underlying the entity

Arguments :

1. top_mt_qc Topic, MultiTopic, QueryCondition

DDS_EXPORT dds_return_t dds_get_query_parameters(_In_ dds_entity_t e, _Out_writes_to_ (, 1) dds_query_parameters_t* qp)
Description : Retrieve the query-parameters

Arguments :

1. top_mt_qc Topic, MultiTopic, QueryCondition

DDS_EXPORT dds_return_t dds_set_query_parameters(_In_ dds_entity_t e, _In_reads_opt_z_ (, 1) dds_query_parameters_t* qp)
Description : Set the query-parameters

Arguments :

1. top_mt_qc Topic, MultiTopic, QueryCondition

DDS_EXPORT dds_entity_t dds_get_topic(_In_ dds_entity_t e)

Variables

Out void _Out_ dds_sample_info_t _In_ uint32_t _In_ dds_instance_handle_t _In_ uint32_t
Out dds_instance_handle_t* ihdl
Out uint32_t* status
_In_opt_ const dds_qos_t* qos
_In_opt_ const dds_qos_t _In_opt_ const dds_listener_t* listener
_Out_opt_ dds_entity_t* children
In size_t size
Out dds_domainid_t* id
In const dds_topic_descriptor_t* descriptor
_In_z_ const char* name

```
_In_ dds_duration_t timeout
_In_ dds_duration_t max_wait
_Out_ void _Out_ dds_sample_info_t _In_ uint32_t _In_ dds_instance_handle_t handle
_In_ const void* data
_In_ const void _In_ dds_time_t timestamp
_In_ uint32_t _In_ dds_querycondition_filter_fn filter
_In_ dds_entity_t entity
_In_ dds_entity_t _In_ dds_attach_t x
_In_ bool trigger
_In_ size_t nxs
_In_ size_t _In_ dds_duration_t reltimeout
_In_ size_t _In_ dds_time_t abstimeout
_Out_ void** buf
_Out_ void _Out_ dds_sample_info_t* si
_In_ size_t bufsz
_Out_ void _Out_ dds_sample_info_t _In_ uint32_t maxs
dir /home/jenkins/workspace/BuildChameleonLinux64bit/cham/src/core
dir /home/jenkins/workspace/BuildChameleonLinux64bit/cham/src/core/ddsc/include/ddsc
dir /home/jenkins/workspace/BuildChameleonLinux64bit/cham/src/core/ddsc
dir /home/jenkins/workspace/BuildChameleonLinux64bit/cham/src/core/ddsc/include
dir /home/jenkins/workspace/BuildChameleonLinux64bit/cham/src
```


CHAPTER 7

Indices and tables

- `genindex`
- `modindex`
- `search`

-
- _In_reads_bytes_ (C++ function), 139
 _Inout_updates_ (C++ function), 86, 152
 _Out_writes_to_ (C++ function), 72, 143
 _Out_writes_to_opt_ (C++ function), 72
 _Out_writes_z_ (C++ function), 49, 132

 cdr (C++ member), 88

 dds_aligned_allocator (C++ class), 23
 dds_aligned_allocator::alloc (C++ member), 23
 dds_aligned_allocator::free (C++ member), 23
 dds_aligned_allocator_t (C++ type), 89
 DDS_ALIVE_INSTANCE_STATE (C macro), 92
 dds_alloc_fn_t (C++ type), 89
 dds_allocator (C++ class), 23
 dds_allocator::free (C++ member), 23
 dds_allocator::malloc (C++ member), 23
 dds_allocator::realloc (C++ member), 23
 dds_allocator_t (C++ type), 89
 DDS_ANY_INSTANCE_STATE (C macro), 92
 DDS_ANY_SAMPLE_STATE (C macro), 92
 DDS_ANY_STATE (C macro), 92
 DDS_ANY_VIEW_STATE (C macro), 92
 dds_attach_t (C++ type), 30, 121
 DDS_CHECK_EXIT (C macro), 90
 DDS_CHECK_FAIL (C macro), 90
 DDS_CHECK_REPORT (C macro), 90
 DDS_DATA_AVAILABLE_STATUS (C macro), 29, 119
 DDS_DATA_ON_READERS_STATUS (C macro), 29, 119
 DDS_DEADLINE_QOS_POLICY_ID (C macro), 101
 dds_destination_order_kind (C++ type), 103
 dds_destination_order_kind_t (C++ type), 102
 DDS_DESTINATIONORDER_BY_RECEPTION_TIMESTAMP (C++ class), 103
 DDS_DESTINATIONORDER_BY_SOURCE_TIMESTAMP (C++ class), 103
 DDS_DESTINATIONORDER_QOS_POLICY_ID (C macro), 101
 DDS_DOMAIN_DEFAULT (C macro), 92
 dds_domainid_t (C++ type), 94
 dds_durability_kind (C++ type), 102
 dds_durability_kind_t (C++ type), 102
 DDS_DURABILITY_PERSISTENT (C++ class), 102
 DDS_DURABILITY_QOS_POLICY_ID (C macro), 101
 DDS_DURABILITY_TRANSIENT (C++ class), 102
 DDS_DURABILITY_TRANSIENT_LOCAL (C++ class), 102
 DDS_DURABILITY_VOLATILE (C++ class), 102
 DDS_DURABILITYSERVICE_QOS_POLICY_ID (C macro), 101
 dds_duration_t (C++ type), 118
 dds_entity_kind (C++ type), 94
 DDS_ENTITY_KIND_MASK (C macro), 92
 dds_entity_kind_t (C++ type), 94
 DDS_ENTITY_NIL (C macro), 92
 DDS_ENTITYFACTORY_QOS_POLICY_ID (C macro), 101
 DDS_ERR_CHECK (C macro), 91
 dds_err_file_id (C macro), 91
 DDS_ERR_FILE_ID_MASK (C macro), 91
 dds_err_line (C macro), 91
 DDS_ERR_LINE_MASK (C macro), 91
 dds_err_nr (C macro), 91
 DDS_ERR_NR_MASK (C macro), 91
 DDS_FAIL (C macro), 91
 dds_fail_fn (C++ type), 91
 DDS_FREE_ALL (C++ class), 89
 DDS_FREE_ALL_BIT (C macro), 89
 DDS_FREE_CONTENTS (C++ class), 89
 DDS_FREE_CONTENTS_BIT (C macro), 89
 dds_free_fn_t (C++ type), 89
 DDS_FREE_KEY (C++ class), 89
 DDS_FREE_KEY_BIT (C macro), 89
 dds_free_op_t (C++ type), 89
 DDS_GROUPDATA_QOS_POLICY_ID (C macro), 101
 DDS_HANDLE_NIL (C macro), 92
 DDS_HISTORY_KEEP_ALL (C++ class), 102
 DDS_HISTORY_KEEP_LAST (C++ class), 102
 dds_history_kind (C++ type), 102

dds_history_kind_t (C++ type), 102
 DDS_HISTORY_QOS_POLICY_ID (C macro), 101
 dds_history_qospolicy (C++ class), 23
 dds_history_qospolicy::depth (C++ member), 23
 dds_history_qospolicy::kind (C++ member), 23
 dds_history_qospolicy_t (C++ type), 102
 DDS_INCONSISTENT_TOPIC_STATUS (C macro), 29, 119
 dds_inconsistent_topic_status (C++ class), 23
 dds_inconsistent_topic_status::total_count (C++ member), 24
 dds_inconsistent_topic_status::total_count_change (C++ member), 24
 dds_inconsistent_topic_status_t (C++ type), 111
 DDS_INFINITY (C macro), 118
 dds_instance_handle_t (C++ type), 94
 dds_instance_state (C++ type), 30, 122
 dds_instance_state_t (C++ type), 30, 119
 DDS_INT_TO_STRING (C macro), 91
 DDS_INVALID_QOS_POLICY_ID (C macro), 101
 DDS_IST_ALIVE (C++ class), 30, 31, 122
 DDS_IST_NOT_ALIVE_DISPOSED (C++ class), 31, 122
 DDS_IST_NOT_ALIVE_NO_WRITERS (C++ class), 31, 122
 dds_key_descriptor (C++ class), 24
 dds_key_descriptor::m_index (C++ member), 24
 dds_key_descriptor::m_name (C++ member), 24
 dds_key_descriptor_t (C++ type), 94
 DDS_KIND_COND_QUERY (C++ class), 94
 DDS_KIND_COND_READ (C++ class), 94
 DDS_KIND_DONTCARE (C++ class), 94
 DDS_KIND_INTERNAL (C++ class), 94
 DDS_KIND_PARTICIPANT (C++ class), 94
 DDS_KIND_PUBLISHER (C++ class), 94
 DDS_KIND_READER (C++ class), 94
 DDS_KIND_SUBSCRIBER (C++ class), 94
 DDS_KIND_TOPIC (C++ class), 94
 DDS_KIND_WAITSET (C++ class), 94
 DDS_KIND_WRITER (C++ class), 94
 DDS_LATENCYBUDGET_QOS_POLICY_ID (C macro), 101
 DDS_LENGTH_UNLIMITED (C macro), 92
 DDS_LIFESPAN_QOS_POLICY_ID (C macro), 101
 dds_listener_t (C++ type), 96
 DDS_LIVELINESS_AUTOMATIC (C++ class), 103
 DDS_LIVELINESS_CHANGED_STATUS (C macro), 29, 119
 dds_liveliness_changed_status (C++ class), 24
 dds_liveliness_changed_status::alive_count (C++ member), 24
 dds_liveliness_changed_status::alive_count_change (C++ member), 24
 dds_liveliness_changed_status::last_publication_handle (C++ member), 24
 dds_liveliness_changed_status::not_alive_count (C++ member), 24
 dds_liveliness_changed_status::not_alive_count_change (C++ member), 24
 dds_liveliness_changed_status_t (C++ type), 111
 dds_liveliness_kind (C++ type), 103
 dds_liveliness_kind_t (C++ type), 102
 DDS_LIVELINESS_LOST_STATUS (C macro), 29, 119
 dds_liveliness_lost_status (C++ class), 24
 dds_liveliness_lost_status::total_count (C++ member), 24
 dds_liveliness_lost_status::total_count_change (C++ member), 24
 dds_liveliness_lost_status_t (C++ type), 111
 DDS_LIVELINESS_MANUAL_BY_PARTICIPANT (C++ class), 103
 DDS_LIVELINESS_MANUAL_BY_TOPIC (C++ class), 103
 DDS_LIVELINESS_QOS_POLICY_ID (C macro), 101
 DDS_LUNSET (C macro), 95
 DDS_MSECS (C macro), 118
 DDS_NEVER (C macro), 118
 DDS_NEW_VIEW_STATE (C macro), 92
 DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE (C macro), 92
 DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE (C macro), 92
 DDS_NOT_NEW_VIEW_STATE (C macro), 92
 DDS_NOT_READ_SAMPLE_STATE (C macro), 92
 DDS_NOT_REJECTED (C++ class), 112
 DDS_NSECS_IN_MSEC (C macro), 118
 DDS_NSECS_IN_SEC (C macro), 118
 DDS_NSECS_IN_USEC (C macro), 118
 DDS_OFFERED_DEADLINE_MISSED_STATUS (C macro), 29, 119
 dds_offered_deadline_missed_status (C++ class), 24
 dds_offered_deadline_missed_status::last_instance_handle (C++ member), 24
 dds_offered_deadline_missed_status::total_count (C++ member), 24
 dds_offered_deadline_missed_status::total_count_change (C++ member), 24
 dds_offered_deadline_missed_status_t (C++ type), 111
 DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (C macro), 29, 119
 dds_offered_incompatible_qos_status (C++ class), 24
 dds_offered_incompatible_qos_status::last_policy_id (C++ member), 25
 dds_offered_incompatible_qos_status::total_count (C++ member), 25
 dds_offered_incompatible_qos_status::total_count_change (C++ member), 25
 dds_offered_incompatible_qos_status_t (C++ type), 111

dds_on_data_available_fn (C++ type), 95
dds_on_data_on_readers_fn (C++ type), 95
dds_on_inconsistent_topic_fn (C++ type), 95
dds_on_liveliness_changed_fn (C++ type), 95
dds_on_liveliness_lost_fn (C++ type), 95
dds_on_offered_deadline_missed_fn (C++ type), 95
dds_on_offered_incompatible_qos_fn (C++ type), 95
dds_on_publication_matched_fn (C++ type), 95
dds_on_requested_deadline_missed_fn (C++ type), 95
dds_on_requested_incompatible_qos_fn (C++ type), 95
dds_on_sample_lost_fn (C++ type), 95
dds_on_sample_rejected_fn (C++ type), 95
dds_on_subscription_matched_fn (C++ type), 95
DDS_OP_ADR (C macro), 92
DDS_OP_FLAG_DEF (C macro), 93
DDS_OP_FLAG_KEY (C macro), 93
DDS_OP_JEQ (C macro), 93
DDS_OP_JSR (C macro), 93
DDS_OP_RTS (C macro), 92
DDS_OP_SUBTYPE_1BY (C macro), 93
DDS_OP_SUBTYPE_2BY (C macro), 93
DDS_OP_SUBTYPE_4BY (C macro), 93
DDS_OP_SUBTYPE_8BY (C macro), 93
DDS_OP_SUBTYPE_ARR (C macro), 93
DDS_OP_SUBTYPE_BOO (C macro), 93
DDS_OP_SUBTYPE_BST (C macro), 93
DDS_OP_SUBTYPE_SEQ (C macro), 93
DDS_OP_SUBTYPE_STR (C macro), 93
DDS_OP_SUBTYPE_STU (C macro), 93
DDS_OP_SUBTYPE_UNI (C macro), 93
DDS_OP_TYPE_1BY (C macro), 93
DDS_OP_TYPE_2BY (C macro), 93
DDS_OP_TYPE_4BY (C macro), 93
DDS_OP_TYPE_8BY (C macro), 93
DDS_OP_TYPE_ARR (C macro), 93
DDS_OP_TYPE_BOO (C macro), 93
DDS_OP_TYPE_BST (C macro), 93
DDS_OP_TYPE_SEQ (C macro), 93
DDS_OP_TYPE_STR (C macro), 93
DDS_OP_TYPE_STU (C macro), 93
DDS_OP_TYPE_UNI (C macro), 93
DDS_OP_VAL_1BY (C macro), 93
DDS_OP_VAL_2BY (C macro), 93
DDS_OP_VAL_4BY (C macro), 93
DDS_OP_VAL_8BY (C macro), 93
DDS_OP_VAL_ARR (C macro), 93
DDS_OP_VAL_BST (C macro), 93
DDS_OP_VAL_SEQ (C macro), 93
DDS_OP_VAL_STR (C macro), 93
DDS_OP_VAL_STU (C macro), 93
DDS_OP_VAL_UNI (C macro), 93
DDS_OWNERSHIP_EXCLUSIVE (C++ class), 103
dds_ownership_kind (C++ type), 102
dds_ownership_kind_t (C++ type), 102
DDS_OWNERSHIP_QOS_POLICY_ID (C macro), 101
DDS_OWNERSHIP_SHARED (C++ class), 102
DDS_OWNERSHIPSTRENGTH_QOS_POLICY_ID (C macro), 101
DDS_PARTITION_QOS_POLICY_ID (C macro), 101
dds_presentation_access_scope_kind (C++ type), 103
dds_presentation_access_scope_kind_t (C++ type), 102
DDS_PRESENTATION_GROUP (C++ class), 103
DDS_PRESENTATION_INSTANCE (C++ class), 103
DDS_PRESENTATION_QOS_POLICY_ID (C macro), 101
DDS_PRESENTATION_TOPIC (C++ class), 103
DDS_PUBLICATION_MATCHED_STATUS (C macro), 29, 119
dds_publication_matched_status (C++ class), 25
dds_publication_matched_status::current_count (C++ member), 25
dds_publication_matched_status::current_count_change (C++ member), 25
dds_publication_matched_status::last_subscription_handle (C++ member), 25
dds_publication_matched_status::total_count (C++ member), 25
dds_publication_matched_status::total_count_change (C++ member), 25
dds_publication_matched_status_t (C++ type), 111
dds_qos_t (C++ type), 102
dds_querycondition_filter_fn (C++ type), 30, 120
DDS_READ_SAMPLE_STATE (C macro), 92
DDS_READERDATA_LIFECYCLE_QOS_POLICY_ID (C macro), 101
dds_realloc_fn_t (C++ type), 89
DDS_REJECTED_BY_INSTANCES_LIMIT (C++ class), 112
DDS_REJECTED_BY_SAMPLES_LIMIT (C++ class), 112
DDS_REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT (C++ class), 112
DDS_RELIABILITY_BEST_EFFORT (C++ class), 103
dds_reliability_kind (C++ type), 103
dds_reliability_kind_t (C++ type), 102
DDS_RELIABILITY_QOS_POLICY_ID (C macro), 101
DDS_RELIABILITY_RELIABLE (C++ class), 103
DDS_REQUESTED_DEADLINE_MISSED_STATUS (C macro), 29, 119
dds_requested_deadline_missed_status (C++ class), 25
dds_requested_deadline_missed_status::last_instance_handle (C++ member), 25
dds_requested_deadline_missed_status::total_count (C++ member), 25
dds_requested_deadline_missed_status::total_count_change (C++ member), 25
dds_requested_deadline_missed_status_t (C++ type), 111

DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS (C macro), 29, 119
 dds_requested_incompatible_qos_status (C++ class), 25
 dds_requested_incompatible_qos_status::last_policy_id (C++ member), 25
 dds_requested_incompatible_qos_status::total_count (C++ member), 25
 dds_requested_incompatible_qos_status::total_count_change (C++ member), 25
 dds_requested_incompatible_qos_status_t (C++ type), 111
 dds_resource_limits_qospolicy (C++ class), 25
 dds_resource_limits_qospolicy::max_instances (C++ member), 25
 dds_resource_limits_qospolicy::max_samples (C++ member), 25
 dds_resource_limits_qospolicy::max_samples_per_instance (C++ member), 25
 dds_resource_limits_qospolicy_t (C++ type), 102
 DDS_RESOURCELIMITS_QOS_POLICY_ID (C macro), 101
 DDS_RETCODE_ALREADY_DELETED (C macro), 90
 DDS_RETCODE_BAD_PARAMETER (C macro), 90
 DDS_RETCODE_ERROR (C macro), 90
 DDS_RETCODE_ILLEGAL_OPERATION (C macro), 90
 DDS_RETCODE_IMMUTABLE_POLICY (C macro), 90
 DDS_RETCODE_INCONSISTENT_POLICY (C macro), 90
 DDS_RETCODE_NO_DATA (C macro), 90
 DDS_RETCODE_NOT_ALLOWED_BY_SECURITY (C macro), 90
 DDS_RETCODE_NOT_ENABLED (C macro), 90
 DDS_RETCODE_OK (C macro), 90
 DDS_RETCODE_OUT_OF_RESOURCES (C macro), 90
 DDS_RETCODE_PRECONDITION_NOT_MET (C macro), 90
 DDS_RETCODE_TIMEOUT (C macro), 90
 DDS_RETCODE_UNSUPPORTED (C macro), 90
 dds_sample_info (C++ class), 25
 dds_sample_info::absolute_generation_rank (C++ member), 27
 dds_sample_info::disposed_generation_count (C++ member), 26
 dds_sample_info::generation_rank (C++ member), 27
 dds_sample_info::instance_handle (C++ member), 26
 dds_sample_info::instance_state (C++ member), 26
 dds_sample_info::no_writers_generation_count (C++ member), 27
 dds_sample_info::publication_handle (C++ member), 26
 dds_sample_info::reception_timestamp (C++ member), 27
 dds_sample_info::sample_rank (C++ member), 27
 dds_sample_info::sample_state (C++ member), 26
 dds_sample_info::source_timestamp (C++ member), 26
 dds_sample_info::valid_data (C++ member), 26
 dds_sample_info::view_state (C++ member), 26
 dds_sample_info_t (C++ type), 30, 120
 DDS_SAMPLE_LOST_STATUS (C macro), 29, 119
 dds_sample_lost_status (C++ class), 27
 dds_sample_lost_status::total_count (C++ member), 27
 dds_sample_lost_status::total_count_change (C++ member), 27
 dds_sample_lost_status_t (C++ type), 111
 DDS_SAMPLE_REJECTED_STATUS (C macro), 29, 119
 dds_sample_rejected_status (C++ class), 27
 dds_sample_rejected_status::last_instance_handle (C++ member), 27
 dds_sample_rejected_status::last_reason (C++ member), 27
 dds_sample_rejected_status::total_count (C++ member), 27
 dds_sample_rejected_status::total_count_change (C++ member), 27
 dds_sample_rejected_status_kind (C++ type), 112
 dds_sample_rejected_status_t (C++ type), 111
 dds_sample_state (C++ type), 30, 121
 dds_sample_state_t (C++ type), 30, 119
 DDS_SECS (C macro), 118
 dds_sequence (C++ class), 27
 dds_sequence::buffer (C++ member), 27
 dds_sequence::length (C++ member), 27
 dds_sequence::maximum (C++ member), 27
 dds_sequence::release (C++ member), 27
 dds_sequence_t (C++ type), 94
 DDS_SST_NOT_READ (C++ class), 30, 121
 DDS_SST_READ (C++ class), 30, 121
 dds_stream (C++ class), 27
 dds_stream::m_buffer (C++ member), 28
 dds_stream::m_endian (C++ member), 28
 dds_stream::m_failed (C++ member), 28
 dds_stream::m_index (C++ member), 28
 dds_stream::m_size (C++ member), 28
 DDS_STREAM_BE (C macro), 116
 DDS_STREAM_LE (C macro), 116
 dds_stream_read_char (C macro), 116
 dds_stream_read_int16 (C macro), 116
 dds_stream_read_int32 (C macro), 116
 dds_stream_read_int64 (C macro), 116
 dds_stream_read_int8 (C macro), 116
 dds_stream_t (C++ type), 117
 dds_stream_write_char (C macro), 116
 dds_stream_write_int16 (C macro), 116
 dds_stream_write_int32 (C macro), 116

dds_stream_write_int64 (C macro), 116
dds_stream_write_int8 (C macro), 116
DDS_SUBSCRIPTION_MATCHED_STATUS (C macro), 29, 119
dds_subscription_matched_status (C++ class), 28
dds_subscription_matched_status::current_count (C++ member), 28
dds_subscription_matched_status::current_count_change (C++ member), 28
dds_subscription_matched_status::last_publication_handle (C++ member), 28
dds_subscription_matched_status::total_count (C++ member), 28
dds_subscription_matched_status::total_count_change (C++ member), 28
dds_subscription_matched_status_t (C++ type), 111
DDS_SUCCESS (C macro), 91
dds_time_t (C++ type), 118
DDS_TIMEBASEDFILTER_QOS_POLICY_ID (C macro), 101
DDS_TO_STRING (C macro), 91
dds_topic_descriptor (C++ class), 28
dds_topic_descriptor::m_align (C++ member), 28
dds_topic_descriptor::m_flagset (C++ member), 28
dds_topic_descriptor::m_keys (C++ member), 28
dds_topic_descriptor::m_meta (C++ member), 28
dds_topic_descriptor::m_nkeys (C++ member), 28
dds_topic_descriptor::m_nops (C++ member), 28
dds_topic_descriptor::m_ops (C++ member), 28
dds_topic_descriptor::m_size (C++ member), 28
dds_topic_descriptor::m_typename (C++ member), 28
dds_topic_descriptor_t (C++ type), 94
dds_topic_filter_fn (C++ type), 30
DDS_TOPIC_FIXED_KEY (C macro), 92
DDS_TOPIC_NO_OPTIMIZE (C macro), 92
DDS_TOPICDATA_QOS_POLICY_ID (C macro), 101
DDS_TRANSPORTPRIORITY_QOS_POLICY_ID (C macro), 101
dds_uptr_t (C++ type), 28
dds_uptr_t::p16 (C++ member), 28
dds_uptr_t::p32 (C++ member), 28
dds_uptr_t::p64 (C++ member), 28
dds_uptr_t::p8 (C++ member), 28
dds_uptr_t::pd (C++ member), 28
dds_uptr_t::pf (C++ member), 28
dds_uptr_t::pv (C++ member), 29
DDS_USECS (C macro), 118
DDS_USERDATA_QOS_POLICY_ID (C macro), 101
dds_view_state (C++ type), 30, 121
dds_view_state_t (C++ type), 30, 119
DDS_VST_NEW (C++ class), 30, 121, 122
DDS_VST_OLD (C++ class), 30, 121, 122
DDS_WRITERDATA_LIFECYCLE_QOS_POLICY_ID (C macro), 101
inst (C++ member), 89
max_wait (C++ member), 88