RTI Connext Python API

Version 7.3.0.

Contents

RTI® Connext® is a connectivity framework for building distributed applications with requirements for high performance and scalability.

The **RTI Connext Python API** provides access to most Connext features from Python. It is supported on Python 3.6 to 3.12, and on Linux® x64 and arm64, Windows® x64, and macOS® x64 and arm64.

Chapter 1

Getting Started

The RTI Connext Getting Started Guide helps you install the software and run your first RTI Connext Python application while learning general concepts of Connext.

To learn more about the Python API, see *API Overview*; you can start with the *Hello World* example.

If you're looking for specific documentation, check out the *Quick Reference*, or the full *API reference*.

If you're migrating from *RTI Connector*, the Connext Migration Guide on RTI Community provides guidance.

Chapter 2

Contents

2.1 API Overview

2.1.1 Hello World

This section shows the basic code to publish and subscribe to a topic defined by a simple data type. Similar code can be generated by *rtiddsgen* from an IDL or XML file defining your data types. The sections that follow expand on the concepts shown in this example.

Define your types:

```
# hello.py

import rti.types as idl

@idl.struct
class HelloWorld:
    message: str = ""
```

Create a *DataWriter* to publish the HelloWorld Topic:

```
# hello_publisher.py

import time
import rti.connextdds as dds
from hello import HelloWorld

participant = dds.DomainParticipant(domain_id=0)
topic = dds.Topic(participant, 'HelloWorld Topic', HelloWorld)
writer = dds.DataWriter(participant.implicit_publisher, topic)

for i in range(10):
    writer.write(HelloWorld(message=f'Hello World! #{i}'))
    time.sleep(1)
```

Create a *DataReader* to subscribe to the HelloWorld Topic:

```
# hello_subscriber.py

import rti.connextdds as dds
import rti.asyncio
from hello import HelloWorld

participant = dds.DomainParticipant(domain_id=0)
topic = dds.Topic(participant, 'HelloWorld Topic', HelloWorld)
reader = dds.DataReader(participant.implicit_subscriber, topic)

async def print_data():
    async for data in reader.take_data_async():
        print(f"Received: {data}")

rti.asyncio.run(print_data())
```

Continue to the next sections to learn about a *DomainParticipant*, *Topics*, *Publications*, *Subscriptions*, and *Data Types*.

2.1.2 DomainParticipant

DomainParticipants are the focal point for creating, destroying, and managing other *Connext* objects. A DDS domain is a logical network of applications: only applications that belong to the same DDS domain may communicate using *Connext*. A DDS domain is identified by a unique integer value known as a domain ID. An application participates in a DDS domain by creating a *DomainParticipant* for that domain ID.

The following code creates a *`DomainParticipant`* on domain *0*.

```
import rti.connextdds as dds
participant = dds.DomainParticipant(domain_id=0)
```

Like all *`IEntity`* types, *DomainParticipants* have QoS policies and listeners. The following example shows how to create a *DomainParticipant* with a specific QoS policy:

```
qos = dds.DomainParticipantQos()
qos.database.shutdown_cleanup_period = dds.Duration.from_milliseconds(10)
participant = dds.DomainParticipant(domain_id=0, qos=qos)
```

A *DomainParticipant* and its contained entities can also be created from an XML definition with the *`QosProvider.create_participant_from_config()`* function.

DomainParticipants (and all other *Entities*) get destroyed automatically when they are garbage collected; however, to ensure that they are destroyed at a certain point in your application, you can call `close()` or create them within a `with` block:

```
with dds.DomainParticipant(domain_id=0) as participant:
    print(participant.domain_id)
    # ...
```

2.1.3 Topics

Shared knowledge of the data types is a requirement for different applications to communicate with DDS. The applications must also share a way to identify what data is to be shared. Data (regardless of type) is uniquely distinguished by using a name called a *Topic*.

Topics allow for *Publishers* and *Subscribers* to communicate without knowing about each other. They dynamically discover each other through *Topics*. The data that the *DataReader* and *DataWriter* share is described by a *Topic*.

Topics are identified by a name, and they are associated with a type and a *DomainParticipant*.

The following code creates a *Topic* named "Car Position" for a type `Point`:

```
topic = dds.Topic(participant, "Car Position", Point)
```

Where `Point` can be defined in IDL as:

```
struct Point {
    int64 x;
    int64 y;
};
```

And in Python as follows:

```
import rti.types as idl

@idl.struct
class Point:
    x: int = 0
    y: int = 0
```

Data Types explains how to define your types in more detail.

Special Topics

In addition to the class *Topic*, there are a few separate Topic classes for certain types:

- For `DynamicData` *Topics*: *DynamicData.Topic* (see *DynamicType and DynamicData*)
- For the built-in discovery *Topics*: *ParticipantBuiltinTopicData.Topic*, *SubscriptionBuiltinTopicData.Topic*, *PublicationBuiltinTopicData.Topic*, *TopicBuiltinTopicData.Topic*

2.1.4 Publications

An application uses *DataWriters* to send data. A *DataWriter* is associated with a single *Topic*. You can have multiple *DataWriters* and *Topics* in a single application. In addition, you can have more than one *DataWriter* for a particular *Topic* in a single application. *Publishers* own and manage *DataWriters*.

To create a *DataWriter*, you need a *Topic* (see *Topics*) and a *DomainParticipant*. Additionally, you may add a QoS parameter and a listener.

The following code creates a `DataWriter` for the *Topic* we created in the *Topics* section:

```
publisher = dds.Publisher(participant)
writer = dds.DataWriter(publisher, topic)
```

To publish data, create a sample, set the values, and write it:

```
data = Point(x=1, y=2)
writer.write(data)
```

A special DataWriter type for DynamicData, `DynamicData.DataWriter` is also available. Find more information in *DynamicType and DynamicData*.

2.1.5 Subscriptions

An application uses *DataReaders* to access data received over DDS. A *DataReader* is associated with a single *Topic*. You can have multiple *DataReaders* and *Topics* in a single application. In addition, you can have more than one *DataReader* for a particular *Topic* in a single application. *Subscribers* own and manage *DataReaders*.

Creating a DataReader

To create a *DataReader*, you need to create a *Topic* (see *Topics*) and a *DomainParticipant*. Optionally, you can add a QoS parameter and a listener.

The following code creates a `DataReader` for the *Topic* we created in the *Topics* section:

```
subscriber = dds.Subscriber(participant)
reader = dds.DataReader(subscriber, topic)
```

A *DataReader* can be created with a `ContentFilteredTopic`, instead of a regular *Topic* to define a content-based subscription with a filter on the data type.

Reading data

When data is available, the *DataReader.read_data()* and *DataReader.take_data()* methods return a collection of data samples of the type specified in the *Topic*. (`take_data` removes the data from the reader, and `read_data` keeps it so it can be accessed again.)

```
for sample in reader.take_data():
    print(sample)
```

To also access the meta-data associated with each data sample, use *DataReader.take()* or *DataReader.read()*:

```
for data, info in reader.take():
    if info.valid:
        print(f"Data received: {data}")
    else:
        print(f"State changed: {info.state}")
```

Each call to these methods returns newly created objects (even `read()`), unlike other Connext language bindings, which return temporary loaned objects.

The *DataReader.select()* method allows selecting which data to read.

Being notified when data is available

There are a few ways to check if a reader has data available:

- Polling for data
- Using a Condition and a WaitSet
- Reading with an asynchronous generator
- Using the DataReaderListener

Polling for data means that you call the "read" or "take" methods described before at certain intervals to check if they return any data.

A *StatusCondition* and a *WaitSet* allows waiting synchronously until a DataReader status change triggers, including the `DATA_AVAILABLE` status:

```
def process_data(_):
    nonlocal reader
    for sample in reader.take_data():
        print(sample)

# Each Entity has a StatusCondition
status_condition = dds.StatusCondition(reader)

# Specify which status to get notified about and set the handler:
status_condition.enable_statuses = dds.StatusMask.DATA_AVAILABLE
status_condition.handler(process_data)
```

```
# Attach the condition to a waitset and call dispatch() to execute the
# condition handlers when they become active
waitset = dds.WaitSet()
waitset += status_condition
while True:
    waitset.dispatch(dds.Duration(4)) # Wait up to 4 seconds
```

The `async` versions of the "take" methods provide a simple way to write your subscriber application. The methods *`DataReader.take_data_async()`* or *`DataReader.take_async()`* work as asynchronous generators, returning data as it is received and awaiting as necessary.

To use these functions your application must `import rti.asyncio`, which requires Python 3.7+.

```
import rti.connextdds as dds
import rti.asyncio

async def print_infinite(reader: dds.DataReader):
    # Print data as it arrives, suspending the coroutine until data is
    # available.
    async for data in reader.take_data_async():
        print(data)

if __name__ == "__main__":
    # ... create the participant, topic, and subscriber as shown before
    reader = dds.DataReader(subscriber, topic)

    # you can use Python's asyncio.run() as well
    rti.asyncio.run(print_infinite(reader))
```

`take_data_async()` and `take_async()` receive an optional `condition` argument (a `dds.ReadCondition` or `dds.QueryCondition`) that can select data by state or content.

Finally, you can use a *`DataReaderListener`* to get notified of status updates, including new data. This method is only recommended for lightweight processing, since the listener callback is executed in an internal Connext thread, and should not block or perform CPU-heavy operations.

Special DataReaders

In addition to the class *`DataReader`*, there are a few separate *DataReader* classes for certain types:

- For `DynamicData` *DataReaders*: `DataReader.Topic` (see *DynamicType and DynamicData*)
- For the built-in discovery *Topics*: *`ParticipantBuiltinTopicData.DataReader`*, *`SubscriptionBuiltinTopicData.DataReader`*, *`PublicationBuiltinTopicData.DataReader`*, *`TopicBuiltinTopicData.DataReader`*

2.1.6 Data Types

IDL to Python mapping

If you're not familiar with IDL and DDS data types, you can read and complete the exercises in the RTI Connext Getting Started Guide, in the chapter on Data Types.

You can generate your Python data types from IDL (or XML) using **rtiddsgen**:

```
rtiddsgen -language python MyTypes.idl [-example universal]
```

(The optional `-example universal` option generates example publisher and subscriber scripts.)

You can also define your types directly in Python, without code generation, following a few conventions.

IDL structs map to Python dataclasses with a special decorator. For example, the following IDL struct:

```
struct Point {
    int64 x;
    int64 y;
};
```

Is defined in Python as follows:

```
import rti.types as idl

@idl.struct
class Point:
    x: int = 0
    y: int = 0
```

The `@idl.struct` decorator parses the type and generates the necessary type support routines that allow DDS to serialize and deserialize instances of this class. `@idl.struct` also applies `@dataclass` to the class, adding all its functionality (`__init__`, `__eq__`, `__repr__`, `__hash__`, `copy.deepcopy()` support, etc.). For example:

```
point1 = Point(x=1, y=2)
point2 = Point(y=2) # x=0, y=2
point3 = Point() # x=0, y=0

print(point1)
assert point1 != point2

point2.x = 1
assert point1 == point2

point4 = copy.deepcopy(point3)
assert point4 == point3
```

`Point` can be used to create a *`Topic`* as we explained in *Topics*.

There are a few pre-defined classes available for convenience (see *Built-in types* below).

The following sections explain how different IDL types map to Python.

Primitive types

The IDL types `int64`, `double`, and `bool` map directly to the Python built-in types `int`, `float`, and `bool`.

For IDL types of different sign or size, type annotations are provided. For example, the IDL types `int32`, `uint16`, and `float` map to the type annotations `idl.int32`, `idl.uint16`, and `idl.float32`, respectively.

For example, the following IDL type:

```
struct Foo {
    int32 a;
    int64 b;
    uint64 c;
    float d;
    double e;
    bool f;
};
```

Maps to the following Python class:

```
@idl.struct
class Foo:
    a: idl.int32
    b: int
    c: idl.uint64
    d: idl.float32
    e: float
    f: bool
```

(Note that IDL's `double` maps to Python's `float`, and IDL's `float` to `idl.float32`.)

Warning: The sign or size of the types is currently not enforced. If you write a value outside the expected range, the subscribers will receive an incorrect value.

Strings

IDL strings map to Python's built-in `str`.

IDL strings can be single (UTF-8) or wide (UTF-16) and bounded or unbounded. These options are passed to the `member_annotations` argument of the type decorator, if needed. By default strings are UTF-8 and unbounded.

For example, the following IDL type:

```
// MyTypes.idl
struct MyStrings {
    string unbounded_str;
    string<128> bounded_str;
```

```
    wstring<256> bounded_wstr;
};
```

Maps to the following Python dataclass:

```
@idl.struct(
    member_annotations = {
        'bounded_str': [idl.bound(128)],
        'bounded_wstr': [idl.bound(256), idl.utf16],
    }
)
class MyStrings:
    unbounded_str: str = ""
    bounded_str: str = ""
    bounded_wstr: str = ""
```

The type above can be generated with **rtiddsgen** as follows:

```
rtiddsgen -unboundedSupport -language python MyTypes.idl
```

Sequences

The mapping of IDL sequences depends on whether the element type is a primitive type or not.

Non-primitive sequences map to Python's `list`.

Primitive sequences map, by default, to efficient, compact collections in the `dds` module. For example, an IDL `sequence<int32>` maps to `dds.Int32Seq`.

IDL sequences can be bounded or unbounded. Bounded sequences may not exceed the number of elements indicated by the bound when the data is written. The bound is specified as part of the `member_annotations` argument to the type decorator.

For example, the following IDL type:

```
struct MySequences {
    sequence<int64, 100> bounded_int64_seq;
    sequence<uint32> unbounded_uint32_seq;
    sequence<Foo> unbounded_foo_seq;
};
```

Maps to the following Python type:

```
@idl.struct(
    member_annotations = {
        'bounded_int64_seq': [idl.bound(100)],
    }
)
class MySequences:
    bounded_int64_seq: Sequence[int] = field(default_factory = idl.array_
↪factory(int))
    unbounded_uint32_seq: Sequence[idl.uint32] = field(default_factory = idl.
```

```
↪array_factory(idl.uint32))
    unbounded_foo_seq: Sequence[Foo] = field(default_factory = list)
```

The `field` function and its `default_factory` argument indicate how the `dataclass` is created by default. When a instance of `MySequences` is created, all the sequences are empty. You can add elements or replace them altogether. For example:

```
my_sequences = MySequences()
my_sequences.bounded_int64_seq.append(1)
my_sequences.bounded_int64_seq.append(2)
my_sequences.unbounded_uint32_seq = dds.Uint32Seq([33] * 5)
my_sequences.unbounded_foo_seq = [Foo(a=x) for x in range(10)]
```

You're not restricted to `dds.Int64Seq` or `dds.Uint32Seq`; you can write a `list`, but the data serialization will be less efficient.

Arrays

The mapping for IDL arrays is similar to that of sequences, except that an array must always have the same number of elements.

The default creation of data samples with arrays populates them with the right number of elements.

The `write()` operation will fail if a sample with an array containing an incorrect number of elements is written.

> **Warning:** Multi-dimensional arrays are not fully supported in this release. They are flattened out and the number of elements is the product of the array's dimensions.

Nested collections

In IDL, you can define sequences of sequences, sequences of arrays, and arrays of sequences.

To do that, the inner collection must be aliased. For example:

```
typedef sequence<Point> PointSeq;
typedef int64 TenInts[10];

struct MySequences {
    sequence<PointSeq> sequence_of_point_sequences;
    sequence<TenInts> sequence_of_int_arrays;
    PointSeq five_point_sequences[5];
};
```

Optional members

By default, members of an IDL `struct` always contain a value and they are always published with a data sample. A member can be declared optional in IDL allowing it to not be sent with all data samples.

In Python, optional members receive the `None` value by default.

For example, the following IDL struct contains a required and an optional member:

```
struct MyOptionals {
    double required_value;
    @optional double optional_value;
};
```

This maps to the following Python dataclass:

```
@idl.struct
class MyOptionals:
    required_value: float = 0.0
    optional_value: Optional[float] = None
```

And a data sample is created by default as follows:

```
sample = MyOptionals()
assert sample.required_value == 0.0
assert sample.optional_value is None
```

Enumerations

IDL enumerations map to Python `IntEnum`-derived classes that are decorated with the `idl.enum` decorator.

```
enum Color {
    RED,
    GREEN,
    BLUE
};
```

Maps to:

```
@idl.enum
class Color(IntEnum):
    RED = 0
    GREEN = 1
    BLUE = 2
```

Unions

IDL unions define types in which only one member exists at a time. The selected member is identified by the "discriminator."

IDL unions map to decorated Python dataclasses with two members (`discriminator` and `value`) and one read/write property for each member that allows setting the value and the discriminator consistently.

For example, the following IDL union:

```
union MyUnion switch(int32) {
    case 0:
        string string_member;
    case 1:
        int64 int_member;
    case 2:
        Point point_member;
};
```

Maps to the following Python class:

```
@idl.union
class MyUnion:

    discriminator: idl.int32 = 0
    value: Union[str, int, Point] = ""

    string_member: str = idl.case(0)
    int_member: int = idl.case(1)
    point_member: Point = idl.case(2)
```

The `discriminator` and `value` members should be used as read-only. To modify the `union`, use the "cases" (read/write properties). For example:

```
sample = MyUnion()

# By default the case with the lowest discriminator value (0 in this case)
# is selected (unless a "default:" label is defined in IDL)
assert sample.discriminator == 0
assert sample.value == ""
assert sample.string_member == ""

# Select a different member:
sample.point_member = Point(1, 2)
assert sample.discriminator == 2
assert sample.value == Point(1, 2)
assert sample.point_member == Point(1, 2)

# Attempting to access member that is not selected raises a ValueError:
try:
    print(sample.string_member)
except ValueError:
    print("string_member is not selected")
```

Modules

Each IDL (or XML) file called **Foo.idl** generates a Python file with the same name, **Foo.py**.

This defines a python package you can import:

```
import Foo

my_type = Foo.MyType()
```

Additionally, in IDL you can define "modules." Similarly to C++ namespaces, an IDL module can be partially defined in several files. To allow for this capability, IDL modules map to Python's SimpleNamespace.

For example, assume the following IDL files:

```
# Foo.idl

module A {
    struct MyType1 { ... };
};

struct MyType2 { ... };
```

And:

```
# Bar.idl

module A {
    struct MyType3 { ... };
};

module B {
    struct MyType4 { ... };
};
```

This generates two Python packages, **Foo.py** and **Bar.py**. The module **A** is accessible from both packages as `Foo.A` and `Bar.A`. **Foo.idl** also defines a type without a module, and **Bar.idl** defines another module, **B**:

```
import Foo
import Bar

sample1 = Foo.A.MyType1()
sample2 = Foo.MyType2()
sample3 = Bar.A.MyType3()
sample4 = Bar.B.MyType4()

# You can create an alias:
MyType3 = Bar.A.MyType3

sample3 = MyType3()
```

IDL annotations

There are several IDL annotations that are passed to the `struct`, `union`, or `enum` decorators in the `type_annotations` or `member_annotations` arguments.

Examples are the `@key` and extensibility annotations (such as `@mutable`):

```
@mutable
struct MutableKeyedType {
    @key string id;
    string value;
};
```

The Python mapping is:

```
@idl.struct(
    type_annotations = [idl.mutable],
    member_annotations = {
        'id': [idl.key]
    }
)
class MutableKeyedType:
    id: str = ""
    value: str = ""
```

These annotations don't have a direct effect on how you use the classes in your application, but they may change how the data is internally processed or delivered.

Built-in types

For convenience, the following types are directly available in the `rti.types.builtin` package:

```
@idl.struct
class String:
    value: str = ""


@idl.struct(member_annotations={'key': [idl.key]})
class KeyedString:
    key: str = ""
    value: str = ""


@idl.struct
class Bytes:
    value: Sequence[idl.uint8] = field(default_factory=idl.array_factory(idl.
↪uint8))


@idl.struct(member_annotations={'key': [idl.key]})
class KeyedBytes:
    key: str = ""
```

```
    value: Sequence[idl.uint8] = field(default_factory=idl.array_factory(idl.
↪uint8))
```

You can directly use these types in your application:

```
import rti.connextdds as dds
from rti.types.builtin import String

participant = dds.DomainParticipant(domain_id=0)
topic = dds.Topic(participant, "HelloWorld", String)
writer = dds.DataWriter(participant, topic)
writer.write(String("Hello World!"))
```

Type support

Every `@idl.struct`-decorated class or `@idl.union`-decorated class has an associated `TypeSupport` object that can be obtained as follows:

```
import rti.types as idl

@idl.struct
class Foo:
    ...

foo_support = idl.get_type_support(Foo)
```

`TypeSupport` provides access to serialization functions:

```
foo = Foo()
buffer = foo_support.serialize(foo)
new_foo = foo_support.deserialize(buffer)
assert foo == new_foo
```

It also provides the property `max_serialized_sample_size`, and the method `get_serialized_sample_size()`.

`TypeSupport` allows converting data samples to and from JSON strings:

```
point_support = idl.get_type_support(Point)
point = Point(x=10, y=20)
assert point == point_support.from_json('{"x": 10, "y": 20}')
assert point_support.to_json(point) == '{"x": 10, "y": 20}'
```

`TypeSupport` also provides information about the type definition as a *`DynamicType`* (`dynamic_type` property) and helpers to convert to and from *`DynamicData`* (`to_dynamic_data()` and `from_dynamic_data()` methods).

DynamicType and DynamicData

The *Connext* Python API can dynamically load type definitions from XML and create `dds.DynamicData` samples.

```
import rti.connextdds as dds

provider = dds.QosProvider("your_types.xml")
my_type = provider.type("MyType")
```

You can now use `my_type` to create a *DynamicData.Topic* and to instantiate a *DynamicData* object:

```
topic = dds.DynamicData.Topic(participant, "Example MyType", my_type)
sample = dds.DynamicData(my_type)
sample["x"] = 42 # assuming MyType has an int32 field
```

You can use `topic` to create a *DynamicData.DataWriter* or a *DynamicData.DataReader*.

Types can also be defined dynamically in the application, using *DynamicType* and its derived classes.

The following example creates a type and instantiates a data sample:

```
# struct Point {
#     double x, y;
# };
point_type = dds.StructType("Point")
point_type.add_member(dds.Member("x", dds.Float64Type()))
point_type.add_member(dds.Member("y", dds.Float64Type()))

# struct MyType {
#     @key string<128> id;
#     Point location;
#     int32 int_array[5];
#     sequence<Point, 10> path;
# };
my_type = dds.StructType("MyType")
my_type.add_member(dds.Member(name="id", data_type=dds.StringType(128), is_
↪key=True))
my_type.add_member(dds.Member(name="location", data_type=point_type))
my_type.add_member(dds.Member(name="int_array", data_type=dds.ArrayType(dds.
↪Int32Type(), 5)))
my_type.add_member(dds.Member(name="path", data_type=dds.SequenceType(point_
↪type, 10)))

# Instantiate the type
sample = dds.DynamicData(my_type)
sample["id"] = "object1"
```

Accessing Nested Members

There are a few different ways to manipulate data with nested types. The `.` notation allows accessing nested primitive members at any level:

```
sample = dds.DynamicData(my_type)
sample["location.x"] = 1.5
sample["location.y"] = 2.5
```

To make multiple modifications to a complex member, you can get a temporary reference (a loan) to the member:

```
with sample.loan_value("location") as location:
    location.data["x"] = 11.5
    location.data["y"] = 12.5
```

A nested member can be assigned from a dictionary, too:

```
sample["location"] = {"x": 4.5, "y": 5.5}
print(sample["location"])
```

Accessing Sequences and Arrays

Sequences and arrays can be retrieved or set using Python lists:

```
# We're using the type we created before
sample = dds.DynamicData(my_type)

# Set the array field with the values of a python list
sample["int_array"] = [1, 2, 3, 4, 5]

# Get all the array elements in a python list
lst = list(sample["int_array"])

# Set and get a single element:
sample["int_array[1]"] = 4
value = sample["int_array[1]"]
```

Lists of structures can be accessed using lists of dictionaries:

```
sample["path"] = [{"x": 1, "y": 2}, {"x": 3, "y": 4}]
path = list(sample["path"])
```

If you only need to set a few elements or fields, you can loan the sequence and its elements. Sequences are automatically resized when you access and index above the current length:

```
with sample.loan_value("path") as path:
    with path.data.loan_value(2) as point:
        point.data["x"] = 111
        point.data["y"] = 222
print(sample["path[2].x"]) # prints 111
```

2.1.7 XML Application Creation

RTI Connext supports the use of XML for the complete system definition. This includes not only the definition of the data types and Quality of Service settings, but also the definition of the Topics, DomainParticipants, and all the Entities they contain (Publishers, Subscribers, DataWriters and DataReaders).

The application calls *QosProvider.create_participant_from_config()* to indicate the participant configuration name of the DomainParticipant that the application wants to create. This method takes care of the rest: creating the DomainParticipant, registering the types and creating Topics, and populating all the configured Entities. When the application needs to read or write data, register listeners, or perform any other action, it simply looks up the appropriate Entity by name and uses it.

Defining your system in a configuration file

The following example configures types and DDS Entities in XML:

```
<!-- my_dds_system.xml -->

<dds>
    <types>
        <struct name="Foo">
            <member name="x" type="int32"/>
        </struct>
    </types>

    <domain_library name="ExampleDomainLibrary" >
        <domain name="ExampleDomain" domain_id="0">
        <register_type name="Foo" type_ref="Foo"/>
        <topic name="ExampleTopic" register_type_ref="Foo"/>
        </domain>
    </domain_library>

    <domain_participant_library name="ExampleParticipantLibrary">
        <domain_participant name="ExamplePublicationParticipant" domain_ref=
↪"ExampleDomainLibrary::ExampleDomain">
            <publisher name="ExamplePublisher">
                <data_writer name="ExampleWriter" topic_ref="ExampleTopic"/>
            </publisher>
        </domain_participant>
        <domain_participant name="ExampleSubscriptionParticipant" domain_ref=
↪"ExampleDomainLibrary::ExampleDomain">
            <data_reader name="ExampleReader" topic_ref="ExampleTopic"/>
        </domain_participant>
    </domain_participant_library>
</dds>
```

See the XML-Based Application Creation Getting Started Guide for more information.

Creating the Entities from the configuration

To load one of the DomainParticipants defined above, use the following code:

```
qos_provider = dds.QosProvider("my_dds_system.xml")
participant = qos_provider.create_participant_from_config(
    "ExampleParticipantLibrary::ExamplePublicationParticipant"
)
```

This creates all the participant's contained entities as well.

The type in this example was also defined in XML, and therefore the DataReaders and DataWriters that use it are *DynamicData.DataReader* and *DynamicData.DataWriter*. The following code looks up the DataWriter, creates a data sample, and writes it:

```
writer = dds.DynamicData.DataWriter(
    participant.find_datawriter("ExamplePublisher::ExampleWriter")
)
sample = writer.create_data()
sample["x"] = 10
writer.write(sample)
```

You can look up the `DataReader` with *DomainParticipant.find_datareader()*.

Using IDL and Python data types

You can also define your types in IDL and Python (as described in *Data Types*) and refer to them in the XML configuration. To use a Python type (whether it is defined directly in Python or generated from IDL), the application must previously register it with the same name used in the XML configuration file.

For example, given the following Python type:

```
@idl.struct
class Point:
    x: int = 0
    y: int = 0
```

We can rewrite the `<domain_library>` in the previous XML configuration to refer to `Point` as follows:

```
<domain_library name="ExampleDomainLibrary" >
    <domain name="ExampleDomain" domain_id="0">
        <register_type name="Point"/>
        <topic name="ExampleTopic" register_type_ref="Point"/>
    </domain>
</domain_library>
```

And then register the type before creating the participant:

```
dds.DomainParticipant.register_idl_type(Point, "Point")

qos_provider = dds.QosProvider("my_dds_system.xml")
```

```
participant = qos_provider.create_participant_from_config(
↪"ExamplePublicationParticipant")
```

Now the writer writes `Point` objects instead of `DynamicData`:

```
writer = dds.DataWriter(
    participant.find_datawriter("ExamplePublisher::ExampleWriter")
)
writer.write(Point(x=10, y=20))
```

2.1.8 Request-Reply and Remote Procedure Calls

As applications become more complex, it often becomes necessary to use other communication patterns in addition to publish-subscribe. Sometimes an application needs to get a one-time snapshot of information; for example, to make a query into a database or retrieve configuration parameters that never change. Other times an application needs to ask a remote application to perform an action on its behalf; for example, to invoke a remote procedure call or a service.

The request-reply pattern has two roles: the *`rti.rpc.Requester`* (service consumer or client) sends a request message and waits for a reply message. The *`rti.rpc.Replier`* (service provider) receives the request message and responds with a reply message.

The Remote Procedure Call pattern is a specialization of the Request-Reply one. In this pattern a client and a server communicate using a functional interface. The client application calls functions on the interface, and the server application implements the functions.

The Connext Core Libraries User's Manual provides more information about these communication patterns in Alternative Communication Models.

A full request-reply example application is available in the Connext Examples repository.

Requesters and Repliers

To create a *Requester* or a *Replier*, you need the following:

- A *DomainParticipant* (see *DomainParticipant*)
- The *request type* and the *reply type* (see *Data Types*)
- A *service name*, or alternatively, you can specify the *request topic name* and the *reply topic name*.

A *Requester* and a *Replier* will communicate when they join the same *domain id* and use the same *service name* (or same topic names), compatible *request* and *reply* types, and compatible *QoS*.

The following code creates a *`rti.rpc.Requester`*:

```
import rti.connextdds as dds
from rti.rpc import Requester

participant = dds.DomainParticipant(domain_id=0)
```

```
requester = Requester(
    request_type=MyRequestType,
    reply_type=MyReplyType,
    participant=participant,
    service_name="MyServiceName"
)
```

The same or a different application can create a *rti.rpc.Replier* as follows:

```
from rti.rpc import Replier

replier = Replier(
    request_type=MyRequestType,
    reply_type=MyReplyType,
    participant=participant,
    service_name="MyServiceName"
)
```

For this example `MyRequestType` and `MyReplyType` are defined as follows:

```
import rti.types as idl

@idl.struct
class MyRequestType:
    x: int = 0
    y: int = 0

@idl.struct
class MyReplyType:
    result: int = 0
```

The `request_type` or `reply_type` arguments can also be instances of *rti.connextdds.DynamicType* or one of the built-in types (*rti.types.builtin*).

There also are a number of optional parameters you can specify, including the quality of service.

Requester: sending requests and receiving replies

To send a request, create an instance of the request type and send it:

```
request = MyRequestType(x=1, y=2)
requester.send_request(request)
```

After sending a request, the Requester will typically wait for one or more replies:

```
replies = requester.receive_replies(dds.Duration(seconds=20))
for data, info in replies:
    print(data)
```

The `receive_replies` operation is equivalent to `wait_for_replies` followed by `take_replies`.

Replier: receiving requests and sending replies

The replier will typically process requests in a loop. The following example receives requests, processes each of them to produce a reply, and sends it.

```
while True:
    requests = replier.receive_requests(dds.Duration(seconds=20))
    for data, info in requests:
        reply = MyReplyType(result=data.x + data.y)
        replier.send_reply(reply, info)
```

Note how `send_reply` expects the `SampleInfo` from the request. This will allow delivering the reply only to the requester that sent the request. Using the same `info` object, multiple replies can be sent for the same request; and replies can be sent in any order: the `data` and `info` objects can be saved for later use.

SimpleReplier

The *`rti.rpc.SimpleReplier`* class is a convenience class that allows for an easy implementation of the replier application using a callback function.

The following example shows how to create a SimpleReplier that implements the same behavior as the Replier above:

```
from rti.rpc import SimpleReplier

simple_replier = SimpleReplier(
    request_type=MyRequestType,
    reply_type=MyReplyType,
    participant=participant,
    service_name="MyServiceName",
    handler=lambda request: MyReplyType(result=request.x + request.y)
)
```

The `handler` argument is a function that receives a single argument of the type `request_type` and must return an instance of the type `reply_type`. A SimpleReplier is only suitable for applications where each request generates exactly one reply and when the process to generate the reply is fast (otherwise the `handler` function would hold internal Connext threads).

Waiting for a reply to a specific request

A *Requester* can send multiple requests and then wait for replies to any of them or to a specific one.

When sending requests, save the returned request id:

```
request_id1 = requester.send_request(MyRequestType(x=1, y=1))
request_id2 = requester.send_request(MyRequestType(x=2, y=2))
```

Then, when waiting for and taking the replies, you can specify the request id:

```
if not requester.wait_for_replies(
    related_request_id=request_id2,
    max_wait=dds.Duration(seconds=20)):

    raise Exception("Timeout waiting for replies")

replies = requester.take_replies(related_request_id=request_id2)
for data, info in replies:
    print(data) # Prints MyReplyType(result=4)
```

The first operation, `wait_for_replies`, will wait until at least one reply for `request_id2` is received. The second operation, `take_replies`, will take all replies for `request_id2`.

Accessing the underlying readers and writers

Requesters and *Repliers* each contain one *DataWriter* and one *DataReader* for the request and reply topics. These can be accessed for advanced use cases.

For example, the *Replier's* reader can be used to asynchronously read requests, which allows simplifying the replier loop as follows:

```
async def process_requests(replier: Replier):
    async for data, info in replier.request_datareader.take_async():
        reply = MyReplyType(result=data.x + data.y)
        replier.send_reply(reply, info)
```

For more information, see Accessing Underlying DataWriters and DataReaders in the Connext Core Libraries User's Manual.

Remote Procedure Calls

> **Warning:** This feature is experimental and subject to change in future releases. It is included in this release to gather customer interest and feedback. For this reason, do not deploy any applications using Remote Procedure Calls in production. The Requester and Replier APIs described above can be used in production.
>
> For support, you may contact support@rti.com.

With this pattern, a service interface allows a client to make remote function calls into a service. A service interface can be defined as follows:

```
import abc
import rti.types as idl
import rti.rpc as rpc

@idl.struct
class Coordinates:
```

```
    x: int = 0
    y: int = 0

@rpc.service
class RobotControl(abc.ABC):
    @rpc.operation
    async def walk_to(self, destination: Coordinates,
                      speed: idl.float32) -> Coordinates: ...

    @rpc.operation
    async def get_speed(self) -> idl.float32: ...
```

A service interface such as `RobotControl` is an abstract base class (ABC) decorated with the `@rpc.service` decorator and with a set of `async` abstract methods decorated with the `@rpc.operation` decorator. The names and types of each operation and its parameters are used to synthesize the DDS topic-types that allow the client and service to communicate. The parameters and return values of an operation can be other IDL-based types (see *Data Types*).

The Python class `RobotControl` is derived from this IDL definition:

```
struct Coordinates {
    int64 x;
    int64 y;
};


@service
interface RobotControl {
    Coordinates walk_to(Coordinates destination, float speed);
    float get_speed();
};
```

Warning: Code generation from IDL `interface` or `exception` definitions to Python is not supported in this release. Other features, such as out parameters or attributes are not supported either.

From the `RobotControl` class you can define a client and a service.

The client class is defined simply as:

```
class RobotControlClient(RobotControl, rpc.ClientBase): ...
```

The base class `rpc.ClientBase` provides the implementation of all the methods decorated with `@rpc.operation`. The `RobotControlClient` class doesn't have any implementation of its own.

A `RobotControlClient` can be instantiated in a DomainParticipant with a service name:

```
import rti.connextdds as dds

participant = dds.DomainParticipant(domain_id=0)
client = RobotControlClient(participant, "My RobotControl")
```

And then used to make remote calls:

```
result = await client.walk_to(destination=Coordinates(x=10, y=10), speed=50.0)
print(result)
```

The call to `walk_to` will send a request to the server and return a coroutine that will complete when the reply is received, providing the return value.

On the service side, an implementation of `RobotControl`, with the server-side logic must be implemented. For example, we can define `RobotControlExample` as follows:

```
class RobotControlExample(RobotControl):
    def __init__(self):
        self.position = Coordinates(0, 0)
        self.speed = 0.0

    async def walk_to(self, destination: Coordinates, speed: idl.float32) ->␣
↪Coordinates:
        if self.speed <= 0.0:
            return self.position # return previously known position

        if speed > 100.0:
            self.speed = 100 # maximum speed
        else:
            self.speed = speed

        # sleep for 0 to 10 seconds, depending on the speed
        await asyncio.sleep(10.0 - 0.1 * self.speed)

        self.speed = 0.0
        self.position = destination
        return destination

    def get_speed(self) -> idl.float32:
        return self.speed
```

The implementation of each method can be asynchronous or synchronous.

To start receiving function calls into `RobotControlExample` we need to create a `rpc.Service` with an instance of `RobotControlExample`, a `DomainParticipant` and a service name:

```
participant = dds.DomainParticipant(domain_id=0)
robot_control = RobotControlExample()
service = rpc.Service(robot_control, participant, "My RobotControl")
await service.run() # runs forever
```

While `service` is running it will receive calls from matching clients (those with the same service name and compatible interfaces and QoS), call The corresponding method in `robot_control` and send the result of the operation back to the client that called it.

Concurrency model

Both the service and the client provide `async` methods and therefore need to be run in an `asyncio` event loop.

All of the client methods derived from the service interface are `async`; for each operation (such as `walk_to`) the client uses a `Requester` to send a request and returns a coroutine that completes when the reply from the service is received, providing the return value.

The server uses the `asyncio` event loop and a `Replier` to run the following tasks when `run()` is called:

- A *read* task that receives requests using the `Replier`.
- One or more *process* tasks that call the methods in the service implementation and send the result back using the `Replier`.

If a method is asynchronous (for example `RobotControlExample.walk_to` in the example above), the *process* task will await for the method to complete. If the method is not asynchronous (`RobotControlExample.get_speed`), it should return a value immediately to not block the event loop.

When all the *process* tasks are busy, the *read* task will stop receiving requests. The number of *process* tasks can be configured with the `task_count` Service parameter:

```
server = rpc.Service(robot_control, participant, "My RobotControl",
                     task_count=10)
```

The default value is 4. Note that with a value of 1, the server will be forced to process requests sequentially in the order they are received.

The service's `run()` method runs until it is cancelled. For example:

```
service = rpc.Service(robot_control, participant, "My RobotControl")
await service.run() # runs forever

# or

service = rpc.Service(robot_control, participant, "My RobotControl")
service_task = asyncio.create_task(service.run())
await asyncio.sleep(10.0) # run for 10 seconds
service_task.cancel()
await service_task
```

Note that `asyncio` event loops are single-threaded. If your service implementation needs to run an operation in a separate thread, you can use the `asyncio.run_in_executor` function. For example, the following implementation of `walk_to` runs the synchronous function `move_robot_impl` in a separate thread:

```
class RobotControlExample(RobotControl):
    async def walk_to(self, destination: Coordinates, speed: idl.float32) ->␣
↪Coordinates:
        # ...
        self.speed = speed

        # run move_robot_impl in a separate thread
        loop = asyncio.get_running_loop()
```

```
        await loop.run_in_executor(None, move_robot_impl, destination, speed)

        self.speed = 0.0
        self.position = destination
        return destination
```

If this concurrency model doesn't fit your needs, you can use the `Requester` and `Replier` classes directly, which provide more flexibility.

Exceptions

Operations in a service interface may throw exceptions. Exceptions are `@idl.struct` classes that inherit from `Exception`. Operations need to declare all the exceptions that they may throw. When a service implementation throws an exception, the `rpc.Service` will catch it and send it to the client, which will rethrow it.

For example, to have the `walk_to` operation throw a `TooFast` exception when the speed is too high, we can define the exception as follows:

```
@idl.struct
class TooFast(Exception):
    speed: idl.float32
```

And declare it in the `walk_to` operation:

```
@rpc.service
class RobotControl:
    @rpc.operation(raises=[TooFast])
    async def walk_to(self, destination: Coordinates, speed: idl.float32) ->␣
↪Coordinates: ...
```

We can now modify `ExampleRobotControl` to throw `TooFast`:

```
class RobotControlExample(RobotControl):
    async def walk_to(self, destination: Coordinates, speed: idl.float32) ->␣
↪Coordinates:
        if speed > 100.0:
            raise TooFast(speed)

        # ...
```

The client can now catch the exception:

```
try:
    await client.walk_to(destination, speed)
except TooFast as e:
    print(f"Too fast: {e.speed}")
```

Exceptions that are not declared in the `raises` parameter of an operation will be propagated and raised in the client as `rpc.RemoteUnknownExceptionError`.

2.2 Quick Reference

This section includes links to common `rti.connextdds` types.

See also *API Overview*

DDS Entities:

- *DomainParticipant*
- *Publisher*
- *Subscriber*
- *Topic*, *ContentFilteredTopic*
- *DataReader*
- *DataWriter*

Data types:

- User types: *Data Types*
- *rti.types.builtin*
- *DynamicData*

Quality of Service (QoS):

- *QosProvider*
- *DomainParticipantQos*
- *TopicQos*
- *PublisherQos*
- *SubscriberQos*
- *DataReaderQos*
- *DataWriterQos*

Listeners:

- *DomainParticipantListener*
- *TopicListener*
- *PublisherListener*
- *SubscriberListener*
- *DataReaderListener*
- *DataWriterListener*

Conditions:

- *WaitSet*

- *Condition*
- *GuardCondition*
- *StatusCondition*
- *ReadCondition*
- *QueryCondition*

Full module documentation: *rti.connextdds*

2.3 rti package

The rti package contains the following packages and modules:

2.3.1 rti.connextdds

rti.connextdds (often aliased as dds) is the main package and contains the DDS API.

The following is the alphabetical list of all classes in the module. For a list of the most relevant ones, see *Quick Reference*.

class rti.connextdds.**ACTEnumMember**

Bases: *DynamicType*

INVALID_INDEX = 4294967295

__eq__(*self:* rti.connextdds.ACTEnumMember, *arg0:* rti.connextdds.ACTEnumMember) → bool

Test for equality.

__getattr__(*self:* rti.connextdds.ACTEnumMember, *arg0: str*) → *rti.connextdds.EnumMember*

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.ACTEnumMember, arg0: str) -> rti.connextdds.EnumMember
2. __getitem__(self: rti.connextdds.ACTEnumMember, arg0: int) -> rti.connextdds.EnumMember

__hash__ = None

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.ACTEnumMember, *arg0:* rti.connextdds.ACTEnumMember) → bool

Test for inequality.

cdr_serialized_sample_max_size (*self:* rti.connextdds.ACTEnumMember, *arg0: int*) → int

Gets the maximum serialized size of samples of this type.

cdr_serialized_sample_min_size (*self:* rti.connextdds.ACTEnumMember, *arg0: int*) → int

Gets the minimum serialized size of samples of this type.

extensibility_kind (*self:* rti.connextdds.ACTEnumMember) → *rti.connextdds.ExtensibilityKind*

Gets the extensibility kind.

find_member_by_name (*self:* rti.connextdds.ACTEnumMember, *arg0: str*) → int

Obtains the member index from its name.

member (**args*, ***kwargs*)

Overloaded function.

1. member(self: rti.connextdds.ACTEnumMember, arg0: int) -> rti.connextdds.EnumMember

Gets a member by its index.

2. member(self: rti.connextdds.ACTEnumMember, arg0: str) -> rti.connextdds.EnumMember

Gets a member by its name.

property member_count

Gets the number of members.

members (*self:* rti.connextdds.ACTEnumMember) → *rti.connextdds.EnumMemberSeq*

Gets a copy of all the members

class rti.connextdds.**ACTMember**

Bases: *DynamicType*

INVALID_INDEX = 4294967295

__eq__ (*self:* rti.connextdds.ACTMember, *arg0:* rti.connextdds.ACTMember) → bool

Test for equality.

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.ACTMember, arg0: str) -> rti.connextdds.Member

2. __getitem__(self: rti.connextdds.ACTMember, arg0: int) -> rti.connextdds.Member

__hash__ = None

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ACTMember, *arg0:* rti.connextdds.ACTMember) → bool
Test for inequality.

cdr_serialized_sample_max_size (*self:* rti.connextdds.ACTMember, *arg0: int*) → int
Gets the maximum serialized size of samples of this type.

cdr_serialized_sample_min_size (*self:* rti.connextdds.ACTMember, *arg0: int*) → int
Gets the minimum serialized size of samples of this type.

extensibility_kind (*self:* rti.connextdds.ACTMember) → *rti.connextdds.ExtensibilityKind*
Gets the extensibility kind.

find_member_by_name (*self:* rti.connextdds.ACTMember, *arg0: str*) → int
Obtains the member index from its name.

member (**args*, ***kwargs*)
Overloaded function.

1. member(self: rti.connextdds.ACTMember, arg0: int) -> rti.connextdds.Member

Gets a member by its index.

2. member(self: rti.connextdds.ACTMember, arg0: str) -> rti.connextdds.Member

Gets a member by its name.

property member_count
Gets the number of members.

members (*self:* rti.connextdds.ACTMember) → *rti.connextdds.MemberSeq*
Gets a copy of all the members

class rti.connextdds.**ACTUnionMember**
Bases: *DynamicType*

INVALID_INDEX = 4294967295

__eq__ (*self:* rti.connextdds.ACTUnionMember, *arg0:* rti.connextdds.ACTUnionMember) → bool
Test for equality.

__getitem__ (**args*, ***kwargs*)
Overloaded function.

1. __getitem__(self: rti.connextdds.ACTUnionMember, arg0: str) -> rti.connextdds.UnionMember
2. __getitem__(self: rti.connextdds.ACTUnionMember, arg0: int) -> rti.connextdds.UnionMember

__hash__ = None

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ACTUnionMember, *arg0:* rti.connextdds.ACTUnionMember) → bool

Test for inequality.

cdr_serialized_sample_max_size (*self:* rti.connextdds.ACTUnionMember, *arg0: int*) → int

Gets the maximum serialized size of samples of this type.

cdr_serialized_sample_min_size (*self:* rti.connextdds.ACTUnionMember, *arg0: int*) → int

Gets the minimum serialized size of samples of this type.

extensibility_kind (*self:* rti.connextdds.ACTUnionMember) → *rti.connextdds.ExtensibilityKind*

Gets the extensibility kind.

find_member_by_name (*self:* rti.connextdds.ACTUnionMember, *arg0: str*) → int

Obtains the member index from its name.

member (**args*, ***kwargs*)

Overloaded function.

1. member(self: rti.connextdds.ACTUnionMember, arg0: int) -> rti.connextdds.UnionMember

Gets a member by its index.

2. member(self: rti.connextdds.ACTUnionMember, arg0: str) -> rti.connextdds.UnionMember

Gets a member by its name.

property member_count

Gets the number of members.

members (*self:* rti.connextdds.ACTUnionMember) → *rti.connextdds.UnionMemberSeq*

Gets a copy of all the members

class rti.connextdds.**AckResponseData**

Bases: pybind11_object

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.AckResponseData) -> None

Create an empty ack response.

2. __init__(self: rti.connextdds.AckResponseData, sequence: rti.connextdds.Uint8Seq) -> None

Create an instance with a byte sequence.

__module__ = 'rti.connextdds'

property value

Gets the subscription handle of the acknowledging DataReader

class rti.connextdds.**AcknowledgmentInfo**

Bases: pybind11_object

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property cookie

Cookie value of the acknowledgment.

property response_data

User data payload of application-level acknowledgment.

property sample_identity

Get the identity of the sample being acknowledged.

property subscription_handle

Gets the subscription handle of the acknowledging DataReader

property valid_response_data

Flag indicating validity of the use response data in the acknowledgment.

class rti.connextdds.**AcknowledgmentKind**

Bases: pybind11_object

APPLICATION_AUTO = <AcknowledgmentKind.APPLICATION_AUTO: 1>

APPLICATION_EXPLICIT = <AcknowledgmentKind.APPLICATION_EXPLICIT: 3>

APPLICATION_ORDERED = <AcknowledgmentKind.APPLICATION_ORDERED: 2>

class AcknowledgmentKind

Bases: pybind11_object

Members:

PROTOCOL : Samples are acknowledged by RTPS protocol.

Samples are acknowledged according to the Real-Time Publish-Subscribe (RTPS) interoperability protocol.

APPLICATION_AUTO : Samples are acknowledged automatically after a subscribing application has accessed them.

The DataReader automatically acknowledges a sample after it has been taken and the loan returned.

APPLICATION_ORDERED : Samples up to a specified sequence number are acknowledged.

APPLICATION_EXPLICIT : Samples are acknowledged after the subscribing application explicitly calls acknowledge on the samples.

Samples received by a DataReader are explicitly acknowledged by the subscribing application, after it calls either DataReader.acknowledge_all or DataReader.acknowledge_sample.

APPLICATION_AUTO = <AcknowledgmentKind.APPLICATION_AUTO: 1>

APPLICATION_EXPLICIT = <AcknowledgmentKind.APPLICATION_EXPLICIT: 3>

APPLICATION_ORDERED = <AcknowledgmentKind.APPLICATION_ORDERED: 2>

PROTOCOL = <AcknowledgmentKind.PROTOCOL: 0>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.AcknowledgmentKind.AcknowledgmentKind) → int

__init__(*self:* rti.connextdds.AcknowledgmentKind.AcknowledgmentKind, *value: int*) → None

__int__(*self:* rti.connextdds.AcknowledgmentKind.AcknowledgmentKind) → int

__members__ = {'APPLICATION_AUTO': <AcknowledgmentKind.APPLICATION_AUTO: 1>, 'APPLICATION_EXPLICIT': <AcknowledgmentKind.APPLICATION_EXPLICIT: 3>, 'APPLICATION_ORDERED': <AcknowledgmentKind.APPLICATION_ORDERED: 2>, 'PROTOCOL': <AcknowledgmentKind.PROTOCOL: 0>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.AcknowledgmentKind.AcknowledgmentKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

PROTOCOL = <AcknowledgmentKind.PROTOCOL: 0>

__eq__(*self:* rti.connextdds.AcknowledgmentKind, *arg0:* rti.connextdds.AcknowledgmentKind) → bool

Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.AcknowledgmentKind, *arg0:* rti.connextdds.AcknowledgmentKind) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.AcknowledgmentKind, *arg0:* rti.connextdds.AcknowledgmentKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.AcknowledgmentKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.AcknowledgmentKind, arg0: rti.connextdds.AcknowledgmentKind.AcknowledgmentKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.AcknowledgmentKind) → *rti.connextdds.AcknowledgmentKind.AcknowledgmentKind*

Int conversion.

__le__ (*self:* rti.connextdds.AcknowledgmentKind, *arg0:* rti.connextdds.AcknowledgmentKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.AcknowledgmentKind, *arg0:* rti.connextdds.AcknowledgmentKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.AcknowledgmentKind, *arg0:* rti.connextdds.AcknowledgmentKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.AcknowledgmentKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**ActivityContextMask**

Bases: pybind11_object

ALL = 00000000000000000000000000111111

DEFAULT = 00000000000000000000000000111111

DOMAIN_ID = 00000000000000000000000000010000

ENTITY_KIND = 00000000000000000000000000001000

ENTITY_NAME = 00000000000000000000000000100000

GUID_PREFIX = 00000000000000000000000000000001

NONE = 00000000000000000000000000000000

TOPIC = 00000000000000000000000000000010

TYPE = 00000000000000000000000000000100

__and__ (*self:* rti.connextdds.ActivityContextMask, *arg0:* rti.connextdds.ActivityContextMask) → *rti.connextdds.ActivityContextMask*

Bitwise logical AND of masks.

__bool__ (*self:* rti.connextdds.ActivityContextMask) → bool

Test if any bits are set.

__contains__ (*self:* rti.connextdds.ActivityContextMask, *arg0:* rti.connextdds.ActivityContextMask) → bool

__eq__ (*self:* rti.connextdds.ActivityContextMask, *arg0:* rti.connextdds.ActivityContextMask) → bool

Compare masks for equality.

__getitem__ (*self:* rti.connextdds.ActivityContextMask, *arg0: int*) → bool

Get individual mask bit.

__hash__ = None

__iand__ (*self:* rti.connextdds.ActivityContextMask, *arg0:* rti.connextdds.ActivityContextMask) → *rti.connextdds.ActivityContextMask*

Set mask to logical AND with another mask.

__ilshift__ (*self:* rti.connextdds.ActivityContextMask, *arg0: int*) → *rti.connextdds.ActivityContextMask*

Left shift bits in mask.

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ActivityContextMask) -> None

Create an ActivityContextMask with no bits set.

2. __init__(self: rti.connextdds.ActivityContextMask, value: int) -> None

Creates a mask from the bits in an integer.

__int__ (*self:* rti.connextdds.ActivityContextMask) → int

Convert mask to int.

__ior__ (*self:* rti.connextdds.ActivityContextMask, *arg0:* rti.connextdds.ActivityContextMask) → *rti.connextdds.ActivityContextMask*

Set mask to logical OR with another mask.

__irshift__ (*self:* rti.connextdds.ActivityContextMask, *arg0: int*) → *rti.connextdds.ActivityContextMask*

Right shift bits in mask.

__ixor__ (*self:* rti.connextdds.ActivityContextMask, *arg0:* rti.connextdds.ActivityContextMask) → *rti.connextdds.ActivityContextMask*

Set mask to logical XOR with another mask.

__lshift__ (*self:* rti.connextdds.ActivityContextMask, *arg0: int*) → *rti.connextdds.ActivityContextMask*

Left shift bits in mask.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ActivityContextMask, *arg0:* rti.connextdds.ActivityContextMask) → bool

Compare masks for inequality.

__or__ (*self:* rti.connextdds.ActivityContextMask, *arg0:* rti.connextdds.ActivityContextMask) → *rti.connextdds.ActivityContextMask*

Bitwise logical OR of masks.

__repr__ (*self:* rti.connextdds.ActivityContextMask) → str

Convert mask to string.

__rshift__ (*self:* rti.connextdds.ActivityContextMask, *arg0: int*) → *rti.connextdds.ActivityContextMask*

Right shift bits in mask.

__setitem__ (*self:* rti.connextdds.ActivityContextMask, *arg0: int*, *arg1: bool*) → None

Set individual mask bit

__str__ (*self:* rti.connextdds.ActivityContextMask) → str

Convert mask to string.

__xor__ (*self:* rti.connextdds.ActivityContextMask, *arg0:* rti.connextdds.ActivityContextMask) → *rti.connextdds.ActivityContextMask*

Bitwise logical XOR of masks.

property count

Returns the number of bits set in the mask.

flip (**args*, ***kwargs*)

Overloaded function.

1. flip(self: rti.connextdds.ActivityContextMask) -> rti.connextdds.ActivityContextMask

Flip all bits in the mask.

2. flip(self: rti.connextdds.ActivityContextMask, pos: int) -> rti.connextdds.ActivityContextMask

Flip the mask bit at the specified position.

reset (**args*, ***kwargs*)

Overloaded function.

1. reset(self: rti.connextdds.ActivityContextMask) -> rti.connextdds.ActivityContextMask

Clear all bits in the mask.

2. reset(self: rti.connextdds.ActivityContextMask, pos: int) -> rti.connextdds.ActivityContextMask

Clear the mask bit at the specified position.

set (**args*, ***kwargs*)

Overloaded function.

1. set(self: rti.connextdds.ActivityContextMask) -> rti.connextdds.ActivityContextMask

Set all bits in the mask.

2. set(self: rti.connextdds.ActivityContextMask, pos: int, value: bool = True) -> rti.connextdds.ActivityContextMask

Set the mask bit at the specified position to the provided value (default: true).

property size

Returns the number of bits in the mask type.

test (*self:* rti.connextdds.ActivityContextMask, *pos: int*) → bool

Test whether the mask bit at position "pos" is set.

test_all (*self:* rti.connextdds.ActivityContextMask) → bool

Test if all bits are set.

test_any (*self:* rti.connextdds.ActivityContextMask) → bool

Test if any bits are set.

test_none (*self:* rti.connextdds.ActivityContextMask) → bool

Test if none of the bits are set.

class rti.connextdds.**AliasType**

Bases: *DynamicType*

Represents a typedef.

__eq__ (*self:* rti.connextdds.AliasType, *arg0:* rti.connextdds.AliasType) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.AliasType, *name: str*, *type_name:* rti.connextdds.DynamicType, *is_pointer: bool = False*) → None

Creates an alias with a name and a related type.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.AliasType, *arg0:* rti.connextdds.AliasType) → bool

Test for inequality.

property is_pointer

Gets whether this alias makes related_type a pointer

related_type (*self:* rti.connextdds.AliasType) → *rti.connextdds.DynamicType*

Gets the related type.

resolve (*self:* rti.connextdds.DynamicType) → *rti.connextdds.DynamicType*

Resolves an AliasType recursively to get the final underlying type.

static resolve_type (*alias_type:* rti.connextdds.DynamicType) → *rti.connextdds.DynamicType*

Resolves an AliasType recursively to get the final underlying type.

class rti.connextdds.**AllocationSettings**

Bases: pybind11_object

AUTO_COUNT = -2

__eq__ (*self:* rti.connextdds.AllocationSettings, *arg0:* rti.connextdds.AllocationSettings) → bool

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.AllocationSettings) -> None

Creates an instance with an initial, max and incremental count set to zero.

2. __init__(self: rti.connextdds.AllocationSettings, initial_count: int, max_count: int, incremental_count: int) -> None

Creates an instance with the given initial, maximum and incremental values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.AllocationSettings, *arg0:* rti.connextdds.AllocationSettings) → bool

property incremental_count

Incremental count of resources.

property initial_count

Initial count of resources.

property max_count

Max count of resources.

exception rti.connextdds.**AlreadyClosedError**

Bases: *Exception*

__module__ = 'rti.connextdds'

class rti.connextdds.**AnyDataReader**

Bases: *IAnyDataReader*

__init__(*self:* rti.connextdds.AnyDataReader, *reader:* rti.connextdds.IAnyDataReader) → None

Create an AnyDataReader instance from a object that conforms to the IAnyDataReader interface.

__module__ = 'rti.connextdds'

class rti.connextdds.**AnyDataReaderListener**

Bases: pybind11_object

__init__(*self:* rti.connextdds.AnyDataReaderListener) → None

__module__ = 'rti.connextdds'

on_data_available(*self:* rti.connextdds.AnyDataReaderListener, *arg0:* rti.connextdds.AnyDataReader) → None

Data available callback.

on_liveliness_changed(*self:* rti.connextdds.AnyDataReaderListener, *arg0:* rti.connextdds.AnyDataReader, *arg1:* rti.connextdds.LivelinessChangedStatus) → None

Liveliness changed callback.

on_requested_deadline_missed(*self:* rti.connextdds.AnyDataReaderListener, *arg0:* rti.connextdds.AnyDataReader, *arg1:* rti.connextdds.RequestedDeadlineMissedStatus) → None

Requested deadline missed callback.

on_requested_incompatible_qos(*self:* rti.connextdds.AnyDataReaderListener, *arg0:* rti.connextdds.AnyDataReader, *arg1:* rti.connextdds.RequestedIncompatibleQosStatus) → None

Requested incompatible QoS callback.

on_sample_lost(*self:* rti.connextdds.AnyDataReaderListener, *arg0:* rti.connextdds.AnyDataReader, *arg1:* rti.connextdds.SampleLostStatus) → None

Sample lost callback.

on_sample_rejected(*self:* rti.connextdds.AnyDataReaderListener, *arg0:* rti.connextdds.AnyDataReader, *arg1:* rti.connextdds.SampleRejectedStatus) → None

Sample rejected callback.

on_subscription_matched(*self:* rti.connextdds.AnyDataReaderListener, *arg0:* rti.connextdds.AnyDataReader, *arg1:* rti.connextdds.SubscriptionMatchedStatus) → None

Subscription matched callback.

class rti.connextdds.**AnyDataReaderSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.AnyDataReaderSeq, *arg0:* rti.connextdds.AnyDataReaderSeq) → *rti.connextdds.AnyDataReaderSeq*

__bool__(*self:* rti.connextdds.AnyDataReaderSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.AnyDataReaderSeq, *x:* rti.connextdds.AnyDataReader) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.AnyDataReaderSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.AnyDataReaderSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.AnyDataReaderSeq, *arg0:* rti.connextdds.AnyDataReaderSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.AnyDataReaderSeq, s: slice) -> rti.connextdds.AnyDataReaderSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.AnyDataReaderSeq, arg0: int) -> rti.connextdds.AnyDataReader

__hash__ = None

__iadd__(*self:* rti.connextdds.AnyDataReaderSeq, *arg0:* rti.connextdds.AnyDataReaderSeq) → *rti.connextdds.AnyDataReaderSeq*

__imul__(*self:* rti.connextdds.AnyDataReaderSeq, *arg0: int*) → *rti.connextdds.AnyDataReaderSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.AnyDataReaderSeq) -> None

2. __init__(self: rti.connextdds.AnyDataReaderSeq, arg0: rti.connextdds.AnyDataReaderSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.AnyDataReaderSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.AnyDataReaderSeq) → Iterator

__len__ (*self:* rti.connextdds.AnyDataReaderSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.AnyDataReaderSeq, *arg0: int*) → *rti.connextdds.AnyDataReaderSeq*

__ne__ (*self:* rti.connextdds.AnyDataReaderSeq, *arg0:* rti.connextdds.AnyDataReaderSeq) → bool

__rmul__ (*self:* rti.connextdds.AnyDataReaderSeq, *arg0: int*) → *rti.connextdds.AnyDataReaderSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.AnyDataReaderSeq, arg0: int, arg1: rti.connextdds.AnyDataReader) -> None

2. __setitem__(self: rti.connextdds.AnyDataReaderSeq, arg0: slice, arg1: rti.connextdds.AnyDataReaderSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.AnyDataReaderSeq, *x:* rti.connextdds.AnyDataReader) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.AnyDataReaderSeq) → None

Clear the contents

count (*self:* rti.connextdds.AnyDataReaderSeq, *x:* rti.connextdds.AnyDataReader) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.AnyDataReaderSeq, L: rti.connextdds.AnyDataReaderSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.AnyDataReaderSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.AnyDataReaderSeq, *i: int*, *x:* rti.connextdds.AnyDataReader) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.AnyDataReaderSeq) -> rti.connextdds.AnyDataReader

Remove and return the last item

2. pop(self: rti.connextdds.AnyDataReaderSeq, i: int) -> rti.connextdds.AnyDataReader

Remove and return the item at index `i`

remove (*self:* rti.connextdds.AnyDataReaderSeq, *x:* rti.connextdds.AnyDataReader) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**AnyDataWriter**

Bases: *IAnyDataWriter*

__init__ (*self:* rti.connextdds.AnyDataWriter, *writer:* rti.connextdds.IAnyDataWriter) → None

Create an AnyDataWriter instance from a object that conforms to the IAnyDataWriter interface.

__module__ = 'rti.connextdds'

class rti.connextdds.**AnyDataWriterListener**

Bases: pybind11_object

__init__ (*self:* rti.connextdds.AnyDataWriterListener) → None

__module__ = 'rti.connextdds'

on_application_acknowledgment (*self:* rti.connextdds.AnyDataWriterListener, *arg0:* rti.connextdds.AnyDataWriter, *arg1:* rti.connextdds.AcknowledgmentInfo) → None

On application acknowledgment callback

on_instance_replaced (*self:* rti.connextdds.AnyDataWriterListener, *arg0:* rti.connextdds.AnyDataWriter, *arg1:* rti.connextdds.InstanceHandle) → None

On instance replaced callback.

on_liveliness_lost (*self:* rti.connextdds.AnyDataWriterListener, *arg0:* rti.connextdds.AnyDataWriter, *arg1:* rti.connextdds.LivelinessLostStatus) → None

Liveliness lost callback.

on_offered_deadline_missed (*self:* rti.connextdds.AnyDataWriterListener, *arg0:* rti.connextdds.AnyDataWriter, *arg1:* rti.connextdds.OfferedDeadlineMissedStatus) → None

Offered deadline missed callback.

on_offered_incompatible_qos (*self:* rti.connextdds.AnyDataWriterListener, *arg0:* rti.connextdds.AnyDataWriter, *arg1:* rti.connextdds.OfferedIncompatibleQosStatus) → None

Offered incompatible QoS callback.

on_publication_matched (*self:* rti.connextdds.AnyDataWriterListener, *arg0:* rti.connextdds.AnyDataWriter, *arg1:* rti.connextdds.PublicationMatchedStatus) → None

Publication matched callback.

on_reliable_reader_activity_changed (*self:* rti.connextdds.AnyDataWriterListener, *arg0:* rti.connextdds.AnyDataWriter, *arg1:* rti.connextdds.ReliableReaderActivityChangedStatus) → None

Reliable reader activity changed callback.

on_reliable_writer_cache_changed (*self:* rti.connextdds.AnyDataWriterListener, *arg0:* rti.connextdds.AnyDataWriter, *arg1:* rti.connextdds.ReliableWriterCacheChangedStatus) → None

Reliable writer cache changed callback.

on_service_request_accepted (*self:* rti.connextdds.AnyDataWriterListener, *arg0:* rti.connextdds.AnyDataWriter, *arg1:* rti.connextdds.ServiceRequestAcceptedStatus) → None

On service request accepted callback.

class rti.connextdds.**AnyDataWriterSeq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.AnyDataWriterSeq, *arg0:* rti.connextdds.AnyDataWriterSeq) → *rti.connextdds.AnyDataWriterSeq*

__bool__ (*self:* rti.connextdds.AnyDataWriterSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.AnyDataWriterSeq, *x:* rti.connextdds.AnyDataWriter) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.AnyDataWriterSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.AnyDataWriterSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.AnyDataWriterSeq, *arg0:* rti.connextdds.AnyDataWriterSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.AnyDataWriterSeq, s: slice) -> rti.connextdds.AnyDataWriterSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.AnyDataWriterSeq, arg0: int) -> rti.connextdds.AnyDataWriter

__hash__ = None

__iadd__ (*self:* rti.connextdds.AnyDataWriterSeq, *arg0:* rti.connextdds.AnyDataWriterSeq) → *rti.connextdds.AnyDataWriterSeq*

__imul__ (*self:* rti.connextdds.AnyDataWriterSeq, *arg0: int*) → *rti.connextdds.AnyDataWriterSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.AnyDataWriterSeq) -> None

2. __init__(self: rti.connextdds.AnyDataWriterSeq, arg0: rti.connextdds.AnyDataWriterSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.AnyDataWriterSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.AnyDataWriterSeq) → Iterator

__len__ (*self:* rti.connextdds.AnyDataWriterSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.AnyDataWriterSeq, *arg0: int*) → *rti.connextdds.AnyDataWriterSeq*

__ne__ (*self:* rti.connextdds.AnyDataWriterSeq, *arg0:* rti.connextdds.AnyDataWriterSeq) → bool

__rmul__ (*self:* rti.connextdds.AnyDataWriterSeq, *arg0: int*) → *rti.connextdds.AnyDataWriterSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.AnyDataWriterSeq, arg0: int, arg1: rti.connextdds.AnyDataWriter) -> None

2. __setitem__(self: rti.connextdds.AnyDataWriterSeq, arg0: slice, arg1: rti.connextdds.AnyDataWriterSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.AnyDataWriterSeq, *x:* rti.connextdds.AnyDataWriter) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.AnyDataWriterSeq) → None

Clear the contents

count (*self:* rti.connextdds.AnyDataWriterSeq, *x:* rti.connextdds.AnyDataWriter) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.AnyDataWriterSeq, L: rti.connextdds.AnyDataWriterSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.AnyDataWriterSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.AnyDataWriterSeq, *i: int*, *x:* rti.connextdds.AnyDataWriter) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.AnyDataWriterSeq) -> rti.connextdds.AnyDataWriter

Remove and return the last item

2. pop(self: rti.connextdds.AnyDataWriterSeq, i: int) -> rti.connextdds.AnyDataWriter

Remove and return the item at index `i`

remove (*self:* rti.connextdds.AnyDataWriterSeq, *x:* rti.connextdds.AnyDataWriter) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**AnyTopic**

Bases: *IAnyTopic*

__init__ (*self:* rti.connextdds.AnyTopic, *topic:* rti.connextdds.IAnyTopic) → None

Create an AnyTopic instance from a object that conforms to the IAnyTopic interface.

__module__ = 'rti.connextdds'

class rti.connextdds.**AnyTopicListener**

Bases: pybind11_object

__init__ (*self:* rti.connextdds.AnyTopicListener) → None

__module__ = 'rti.connextdds'

on_inconsistent_topic(*self:* rti.connextdds.AnyTopicListener, *arg0:* rti.connextdds.AnyTopic, *arg1:* rti.connextdds.InconsistentTopicStatus) → None

Inconsistent topic callback.

class rti.connextdds.**AnyTopicSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.AnyTopicSeq, *arg0:* rti.connextdds.AnyTopicSeq) → *rti.connextdds.AnyTopicSeq*

__bool__(*self:* rti.connextdds.AnyTopicSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.AnyTopicSeq, *x:* rti.connextdds.AnyTopic) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.AnyTopicSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.AnyTopicSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.AnyTopicSeq, *arg0:* rti.connextdds.AnyTopicSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.AnyTopicSeq, s: slice) -> rti.connextdds.AnyTopicSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.AnyTopicSeq, arg0: int) -> rti.connextdds.AnyTopic

__hash__ = None

__iadd__(*self:* rti.connextdds.AnyTopicSeq, *arg0:* rti.connextdds.AnyTopicSeq) → *rti.connextdds.AnyTopicSeq*

__imul__(*self:* rti.connextdds.AnyTopicSeq, *arg0: int*) → *rti.connextdds.AnyTopicSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.AnyTopicSeq) -> None

2. __init__(self: rti.connextdds.AnyTopicSeq, arg0: rti.connextdds.AnyTopicSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.AnyTopicSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.AnyTopicSeq) → Iterator

__len__ (*self:* rti.connextdds.AnyTopicSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.AnyTopicSeq, *arg0: int*) → *rti.connextdds.AnyTopicSeq*

__ne__ (*self:* rti.connextdds.AnyTopicSeq, *arg0:* rti.connextdds.AnyTopicSeq) → bool

__rmul__ (*self:* rti.connextdds.AnyTopicSeq, *arg0: int*) → *rti.connextdds.AnyTopicSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.AnyTopicSeq, arg0: int, arg1: rti.connextdds.AnyTopic) -> None
2. __setitem__(self: rti.connextdds.AnyTopicSeq, arg0: slice, arg1: rti.connextdds.AnyTopicSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.AnyTopicSeq, *x:* rti.connextdds.AnyTopic) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.AnyTopicSeq) → None

Clear the contents

count (*self:* rti.connextdds.AnyTopicSeq, *x:* rti.connextdds.AnyTopic) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.AnyTopicSeq, L: rti.connextdds.AnyTopicSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.AnyTopicSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.AnyTopicSeq, *i: int*, *x:* rti.connextdds.AnyTopic) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.AnyTopicSeq) -> rti.connextdds.AnyTopic

Remove and return the last item

2. pop(self: rti.connextdds.AnyTopicSeq, i: int) -> rti.connextdds.AnyTopic

Remove and return the item at index `i`

remove (*self:* rti.connextdds.AnyTopicSeq, *x:* rti.connextdds.AnyTopic) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**ArrayType**

Bases: *CollectionType*

__eq__ (*self:* rti.connextdds.ArrayType, *arg0:* rti.connextdds.ArrayType) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ArrayType, data_type: rti.connextdds.DynamicType, dimension: int) -> None

Creates a unidimensional array.

2. __init__(self: rti.connextdds.ArrayType, data_type: rti.connextdds.DynamicType, dimensions: rti.connextdds.Int32Seq) -> None

Create a multidimensional array.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ArrayType, *arg0:* rti.connextdds.ArrayType) → bool

Test for inequality.

dimension (*self:* rti.connextdds.ArrayType, *index: int*) → int

Returns the size of the ith dimension

property dimension_count

Number of dimensions.

property total_element_count

Total element count across all dimensions.

class rti.connextdds.**AsynchronousPublisher**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.AsynchronousPublisher, *arg0:* rti.connextdds.AsynchronousPublisher) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.AsynchronousPublisher) → None

Create a default AsynchronousPublisher QoS policy.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.AsynchronousPublisher, *arg0:* rti.connextdds.AsynchronousPublisher) → bool

Test for inequality.

property asynchronous_batch_thread

Get/set the settings associated with the asynchronous batch flush thread.

property disable_asynchronous_batch

Get/set the asynchronous batch flushing capability for this policy.

property disable_asynchronous_write

Get/set the asynchronous write capability for this policy.

property disable_topic_query_publication

Get/set the topic query publication capability for this policy.

disabled = <rti.connextdds.AsynchronousPublisher object>

static enabled(*enable_async_batch: bool = False*) → *rti.connextdds.AsynchronousPublisher*

Create an AsynchronousPublisher QoS policy with asynchronous write (and optionally asynchronous batch flushing) enabled.

property thread

Get/set the settings associated with the asynchronous publisher thread.

property topic_query_publication_thread

Get/set the settings associated with the topic query publication thread.

class rti.connextdds.**Availability**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.Availability, *arg0:* rti.connextdds.Availability) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Availability) -> None

Create a default Availability QoS policy.

2. __init__(self: rti.connextdds.Availability, enable_required_subscriptions: bool, data_waiting_time: rti.connextdds.Duration, endpoint_waiting_time: rti.connextdds.Duration, required_endpount_groups: rti.connextdds.EndpointGroupVector) -> None

Create an Availability QoS policy with the provided parameters.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Availability, *arg0:* rti.connextdds.Availability) → bool

Test for inequality.

property enable_required_subscriptions

Get/set the status of required subscriptions for this policy.

property max_data_availability_waiting_time

Get/set the amount of time to wait before delivering a sample to the application without having received some of the previous samples.

property max_endpoint_availability_waiting_time

Get/set the amount of time to wait to discover DataWriters providing samples for the same data source (virtual GUID).

property required_matched_endpoint_groups

Get/set a copy of the required endpoint groups.

class rti.connextdds.**Batch**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.Batch, *arg0:* rti.connextdds.Batch) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.Batch) → None

Create a Batch QoS policy with the default settings (disabled).

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.Batch, *arg0:* rti.connextdds.Batch) → bool

Test for inequality.

disabled = <rti.connextdds.Batch object>

property enable

Get/set boolean that specifies whether batching is enabled.

enabled = <rti.connextdds.Batch object>

static enabled_with_max_data_bytes(*max_data_bytes: int*) → *rti.connextdds.Batch*

Create a Batch policy with batching enabled that will flush based on the cummulative size of serialized samples.

static enabled_with_max_samples(*max_samples: int*) → *rti.connextdds.Batch*

Create a Batch policy with batching enabled that will flush based on the number of samples written to the batch.

property max_data_bytes

Get/set the maximum cummulative size of serialized samples in a batch. When this limit is reached the batch will be flushed.

property max_flush_delay

Get/set the maximum delay to flush a batch, measured from the time the first sample in the batch is written.

property max_samples

Get/set the maximum number of serialized samples in a batch. When this limit is reached the batch will be flushed.

property source_timestamp_resolution

Get/set the timestamp resolution of the samples in the batch.

property thread_safe_write

Get/set boolean that specifies whether batch writes must be thread safe.

class rti.connextdds.**BoolSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.BoolSeq, *arg0:* rti.connextdds.BoolSeq) → *rti.connextdds.BoolSeq*

__bool__(*self:* rti.connextdds.BoolSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.BoolSeq, *x: bool*) → bool

Return true the container contains x

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.BoolSeq, arg0: int) -> None

Delete the list elements at index i

2. __delitem__(self: rti.connextdds.BoolSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.BoolSeq, *arg0:* rti.connextdds.BoolSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.BoolSeq, s: slice) -> rti.connextdds.BoolSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.BoolSeq, arg0: int) -> bool

__hash__ = None

__iadd__(*self:* rti.connextdds.BoolSeq, *arg0:* rti.connextdds.BoolSeq) → *rti.connextdds.BoolSeq*

__imul__(*self:* rti.connextdds.BoolSeq, *arg0: int*) → *rti.connextdds.BoolSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.BoolSeq) -> None

2. __init__(self: rti.connextdds.BoolSeq, arg0: rti.connextdds.BoolSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.BoolSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.BoolSeq) → Iterator

__len__ (*self:* rti.connextdds.BoolSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.BoolSeq, *arg0: int*) → *rti.connextdds.BoolSeq*

__ne__ (*self:* rti.connextdds.BoolSeq, *arg0:* rti.connextdds.BoolSeq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__ (*self:* rti.connextdds.BoolSeq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.BoolSeq, *arg0: int*) → *rti.connextdds.BoolSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.BoolSeq, arg0: int, arg1: bool) -> None

2. __setitem__(self: rti.connextdds.BoolSeq, arg0: slice, arg1: rti.connextdds.BoolSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.BoolSeq, *x: bool*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.BoolSeq) → None

Clear the contents

count (*self:* rti.connextdds.BoolSeq, *x: bool*) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.BoolSeq, L: rti.connextdds.BoolSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.BoolSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.BoolSeq, *i: int*, *x: bool*) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.BoolSeq) -> bool

Remove and return the last item

2. pop(self: rti.connextdds.BoolSeq, i: int) -> bool

Remove and return the item at index `i`

remove (*self:* rti.connextdds.BoolSeq, *x: bool*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**BoolType**

Bases: *DynamicType*

__eq__ (*self:* rti.connextdds.BoolType, *arg0:* rti.connextdds.BoolType) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.BoolType) → None

Get the singleton for BoolType

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.BoolType, *arg0:* rti.connextdds.BoolType) → bool

Test for inequality.

class rti.connextdds.**BuiltinProfiles**

Bases: pybind11_object

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

baseline = 'BuiltinQosLib::Baseline'

baseline_5_0_0 = 'BuiltinQosLib::Baseline.5.0.0'

baseline_5_1_0 = 'BuiltinQosLib::Baseline.5.1.0'

baseline_5_2_0 = 'BuiltinQosLib::Baseline.5.2.0'

baseline_5_3_0 = 'BuiltinQosLib::Baseline.5.3.0'

baseline_6_0_0 = 'BuiltinQosLib::Baseline.6.0.0'

generic_auto_tuning = 'BuiltinQosLib::Generic.AutoTuning'

generic_best_effort = 'BuiltinQosLib::Generic.BestEffort'

generic_common = 'BuiltinQosLib::Generic.Common'

```
generic_connext_micro_compatibility =
'BuiltinQosLib::Generic.ConnextMicroCompatibility'

generic_connext_micro_compatibility_2_4_3 =
'BuiltinQosLib::Generic.ConnextMicroCompatibility.2.4.3'

generic_connext_micro_compatibility_2_4_9 =
'BuiltinQosLib::Generic.ConnextMicroCompatibility.2.4.9'

generic_keep_last_reliable =
'BuiltinQosLib::Generic.KeepLastReliable'

generic_keep_last_reliable_large_data =
'BuiltinQosLib::Generic.KeepLastReliable.LargeData'

generic_keep_last_reliable_large_data_fast_flow =
'BuiltinQosLib::Generic.KeepLastReliable.LargeData.FastFlow'

generic_keep_last_reliable_large_data_medium_flow =
'BuiltinQosLib::Generic.KeepLastReliable.LargeData.MediumFlow'

generic_keep_last_reliable_large_data_slow_flow =
'BuiltinQosLib::Generic.KeepLastReliable.LargeData.SlowFlow'

generic_keep_last_reliable_persistent =
'BuiltinQosLib::Generic.KeepLastReliable.Persistent'

generic_keep_last_reliable_transient =
'BuiltinQosLib::Generic.KeepLastReliable.Transient'

generic_keep_last_reliable_transient_local =
'BuiltinQosLib::Generic.KeepLastReliable.TransientLocal'

generic_minimal_memory_footprint =
'BuiltinQosLib::Generic.MinimalMemoryFootprint'

generic_monitoring_common =
'BuiltinQosLib::Generic.Monitoring.Common'

generic_other_dds_vendor_compatibility =
'BuiltinQosLib::Generic.510TransportCompatibility'

generic_participant_large_data =
'BuiltinQosLib::Generic.Participant.LargeData'

generic_participant_large_data_monitoring =
'BuiltinQosLib::Generic.Participant.LargeData.Monitoring'

generic_security = 'BuiltinQosLib::Generic.Security'

generic_strict_reliable = 'BuiltinQosLib::Generic.StrictReliable'
```

generic_strict_reliable_high_throughput = 'BuiltinQosLib::Generic.StrictReliable.HighThroughput'

generic_strict_reliable_large_data = 'BuiltinQosLib::Generic.StrictReliable.LargeData'

generic_strict_reliable_large_data_fast_flow = 'BuiltinQosLib::Generic.StrictReliable.LargeData.FastFlow'

generic_strict_reliable_large_data_medium_flow = 'BuiltinQosLib::Generic.StrictReliable.LargeData.MediumFlow'

generic_strict_reliable_large_data_slow_flow = 'BuiltinQosLib::Generic.StrictReliable.LargeData.SlowFlow'

generic_strict_reliable_low_latency = 'BuiltinQosLib::Generic.StrictReliable.LowLatency'

library_name = 'BuiltinQosLib'

pattern_alarm_event = 'BuiltinQosLib::Pattern.AlarmEvent'

pattern_alarm_status = 'BuiltinQosLib::Pattern.AlarmStatus'

pattern_event = 'BuiltinQosLib::Pattern.Event'

pattern_last_value_cache = 'BuiltinQosLib::Pattern.LastValueCache'

pattern_periodic_data = 'BuiltinQosLib::Pattern.PeriodicData'

pattern_reliable_streaming = 'BuiltinQosLib::Pattern.ReliableStreaming'

pattern_status = 'BuiltinQosLib::Pattern.Status'

pattern_streaming = 'BuiltinQosLib::Pattern.Streaming'

class rti.connextdds.**BuiltinTopicKey**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.BuiltinTopicKey, *arg0:* rti.connextdds.BuiltinTopicKey) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.BuiltinTopicKey) → None

Creates a key whose value is all zeros.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.BuiltinTopicKey, *arg0:* rti.connextdds.BuiltinTopicKey) → bool

Test for inequality.

__repr__ (*self:* rti.connextdds.BuiltinTopicKey) → str

Returns a string representation of the key.

property value

Returns a copy of the four integers that represent the key.

class rti.connextdds.**BuiltinTopicReaderResourceLimits**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.BuiltinTopicReaderResourceLimits, *arg0:* rti.connextdds.BuiltinTopicReaderResourceLimits) → bool

__hash__ = None

__init__ (*self:* rti.connextdds.BuiltinTopicReaderResourceLimits) → None

Create a policy with default settings.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.BuiltinTopicReaderResourceLimits, *arg0:* rti.connextdds.BuiltinTopicReaderResourceLimits) → bool

property disable_fragmentation_support

Determines whether the DataReader can receive fragmented samples.

property dynamically_allocate_fragmented_samples

Determines whether the DataReader pre-allocates storage for storing fragmented samples.

property initial_fragmented_samples

The initial number of samples for which a DataReader may store fragments.

property initial_infos

Initial number of sample infos.

property initial_outstanding_reads

The initial number of outstanding calls to read/take (or one of their variants) on the same DataReader for which memory has not been returned by calling LoanedSamples.return_loan().

property initial_samples

Initial number of samples

property max_fragmented_samples

The maximum number of samples for which the DataReader may store fragments at a given point in time.

property max_fragmented_samples_per_remote_writer

The maximum number of samples per remote writer for which a built-in topic reader may store fragments.

property max_fragments_per_sample

Maximum number of fragments for a single sample.

property max_infos

Maximum number of sample infos.

property max_outstanding_reads

The max number of outstanding calls to read/take (or one of their variants) on the same DataReader for which memory has not been returned by calling LoanedSamples.return_loan().

property max_samples

Maximum number of samples

property max_samples_per_read

The maximum number of data samples that the application can receive from the middleware in a single call to DataReader.read() or DataReader.take(). If more data exists in the middleware, the application will need to issue multiple read/take calls.

class rti.connextdds.**ByteVector**

Bases: pybind11_object

A DDS standard container with functionality similar to a C++ vector.

__eq__(*self:* rti.connextdds.ByteVector, *arg0:* rti.connextdds.ByteVector) → bool

Compare ByteVectors for equality.

__getitem__(*self:* rti.connextdds.ByteVector, *arg0: int*) → int

Get the value at the specified index.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ByteVector, buffer: buffer) -> None

Create a ByteVector from another Python buffer.

2. __init__(self: rti.connextdds.ByteVector) -> None

Create an empty ByteVector

3. __init__(self: rti.connextdds.ByteVector, size: int) -> None

Create a ByteVector with a preallocated size.

4. __init__(self: rti.connextdds.ByteVector, vector: rti.connextdds.ByteVector) -> None

Create a copy from another ByteVector.

5. __init__(self: rti.connextdds.ByteVector, arg0: Iterable) -> None

6. __init__(self: rti.connextdds.ByteVector, list: rti.connextdds.Uint8Seq) -> None

Create a ByteVector from a list of values.

__iter__(*self:* rti.connextdds.ByteVector) → Iterator

Iterate over the contents of the vector.

__len__ (*self:* rti.connextdds.ByteVector) → int

Get the length of the ByteVector.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ByteVector, *arg0:* rti.connextdds.ByteVector) → bool

Compare ByteVectors for inequality.

__setitem__ (*self:* rti.connextdds.ByteVector, *arg0: int*, *arg1: int*) → None

Set the value at the specified index.

clear (*self:* rti.connextdds.ByteVector) → None

Resize ByteVector to 0.

resize (*self:* rti.connextdds.ByteVector, *size: int*) → None

Resize ByteVector.

class rti.connextdds.**CdrPaddingKind**

Bases: pybind11_object

AUTO = <CdrPaddingKind.AUTO: 2>

class CdrPaddingKind

Bases: pybind11_object

Members:

ZERO : Set padding bytes to zero during CDR serialization.

NOT_SET : Don't set padding bytes to any value.

AUTO : Set the value automatically, depending on the Entity.

When set on a DomainParticipant the default behavior is NOT_SET

When set on a DataWriter or DataReader the behavior is to inherit the value from the DomainParticipant.

AUTO = <CdrPaddingKind.AUTO: 2>

NOT_SET = <CdrPaddingKind.NOT_SET: 1>

ZERO = <CdrPaddingKind.ZERO: 0>

__eq__ (*self: object*, *other: object*) → bool

__getstate__ (*self: object*) → int

__hash__ (*self: object*) → int

__index__ (*self:* rti.connextdds.CdrPaddingKind.CdrPaddingKind) → int

__init__ (*self:* rti.connextdds.CdrPaddingKind.CdrPaddingKind, *value: int*) → None

__int__ (*self:* rti.connextdds.CdrPaddingKind.CdrPaddingKind) → int

__members__ = {'AUTO': <CdrPaddingKind.AUTO: 2>, 'NOT_SET': <CdrPaddingKind.NOT_SET: 1>, 'ZERO': <CdrPaddingKind.ZERO: 0>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.CdrPaddingKind.CdrPaddingKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

NOT_SET = <CdrPaddingKind.NOT_SET: 1>

ZERO = <CdrPaddingKind.ZERO: 0>

__eq__(*self:* rti.connextdds.CdrPaddingKind, *arg0:* rti.connextdds.CdrPaddingKind) → bool

Apply operator to underlying enumerated values.

__ge__(*self:* rti.connextdds.CdrPaddingKind, *arg0:* rti.connextdds.CdrPaddingKind) → bool

Apply operator to underlying enumerated values.

__gt__(*self:* rti.connextdds.CdrPaddingKind, *arg0:* rti.connextdds.CdrPaddingKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.CdrPaddingKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.CdrPaddingKind, arg0: rti.connextdds.CdrPaddingKind.CdrPaddingKind) -> None

Copy constructor.

__int__(*self:* rti.connextdds.CdrPaddingKind) → *rti.connextdds.CdrPaddingKind.CdrPaddingKind*

Int conversion.

__le__(*self:* rti.connextdds.CdrPaddingKind, *arg0:* rti.connextdds.CdrPaddingKind) → bool

Apply operator to underlying enumerated values.

__lt__(*self:* rti.connextdds.CdrPaddingKind, *arg0:* rti.connextdds.CdrPaddingKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.CdrPaddingKind, *arg0:* rti.connextdds.CdrPaddingKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.CdrPaddingKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**ChannelSettings**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.ChannelSettings, *arg0:* rti.connextdds.ChannelSettings) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.ChannelSettings, *multicast_settings:* rti.connextdds.TransportMulticastSettingsSeq, *filter_expression: str*, *priority: int*) → None

Creates an instance with the specified multicast settings, filter expression and priority.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ChannelSettings, *arg0:* rti.connextdds.ChannelSettings) → bool

Test for inequality.

property filter_expression

A logical expression used to determine the data that will be published in the channel.

property multicast_settings

A sequence of TransportMulticastSettings used to configure the multicast addresses associated with a channel.

property priority

Publication priority.

class rti.connextdds.**ChannelSettingsSeq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.ChannelSettingsSeq, *arg0:* rti.connextdds.ChannelSettingsSeq) → *rti.connextdds.ChannelSettingsSeq*

__bool__ (*self:* rti.connextdds.ChannelSettingsSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.ChannelSettingsSeq, *x:* rti.connextdds.ChannelSettings) → bool

Return true the container contains x

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.ChannelSettingsSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.ChannelSettingsSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.ChannelSettingsSeq, *arg0:* rti.connextdds.ChannelSettingsSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.ChannelSettingsSeq, s: slice) -> rti.connextdds.ChannelSettingsSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.ChannelSettingsSeq, arg0: int) -> rti.connextdds.ChannelSettings

__hash__ = None

__iadd__ (*self:* rti.connextdds.ChannelSettingsSeq, *arg0:* rti.connextdds.ChannelSettingsSeq) → *rti.connextdds.ChannelSettingsSeq*

__imul__ (*self:* rti.connextdds.ChannelSettingsSeq, *arg0: int*) → *rti.connextdds.ChannelSettingsSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ChannelSettingsSeq) -> None

2. __init__(self: rti.connextdds.ChannelSettingsSeq, arg0: rti.connextdds.ChannelSettingsSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.ChannelSettingsSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.ChannelSettingsSeq) → Iterator

__len__ (*self:* rti.connextdds.ChannelSettingsSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.ChannelSettingsSeq, *arg0: int*) → *rti.connextdds.ChannelSettingsSeq*

__ne__ (*self:* rti.connextdds.ChannelSettingsSeq, *arg0:* rti.connextdds.ChannelSettingsSeq) → bool

__rmul__ (*self:* rti.connextdds.ChannelSettingsSeq, *arg0: int*) → *rti.connextdds.ChannelSettingsSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.ChannelSettingsSeq, arg0: int, arg1: rti.connextdds.ChannelSettings) -> None

2. __setitem__(self: rti.connextdds.ChannelSettingsSeq, arg0: slice, arg1: rti.connextdds.ChannelSettingsSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.ChannelSettingsSeq, *x:* rti.connextdds.ChannelSettings) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.ChannelSettingsSeq) → None

Clear the contents

count (*self:* rti.connextdds.ChannelSettingsSeq, *x:* rti.connextdds.ChannelSettings) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.ChannelSettingsSeq, L: rti.connextdds.ChannelSettingsSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.ChannelSettingsSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.ChannelSettingsSeq, *i: int*, *x:* rti.connextdds.ChannelSettings) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.ChannelSettingsSeq) -> rti.connextdds.ChannelSettings

Remove and return the last item

2. pop(self: rti.connextdds.ChannelSettingsSeq, i: int) -> rti.connextdds.ChannelSettings

Remove and return the item at index `i`

remove (*self:* rti.connextdds.ChannelSettingsSeq, *x:* rti.connextdds.ChannelSettings) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**CharSeq**

Bases: `pybind11_object`

__add__ (*self:* rti.connextdds.CharSeq, *arg0:* rti.connextdds.CharSeq) → *rti.connextdds.CharSeq*

__bool__ (*self:* rti.connextdds.CharSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.CharSeq, *x: str*) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.CharSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.CharSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.CharSeq, *arg0:* rti.connextdds.CharSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.CharSeq, s: slice) -> rti.connextdds.CharSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.CharSeq, arg0: int) -> str

__getstate__ (*self:* rti.connextdds.CharSeq) → bytes

__hash__ = None

__iadd__ (*self:* rti.connextdds.CharSeq, *arg0:* rti.connextdds.CharSeq) → *rti.connextdds.CharSeq*

__imul__ (*self:* rti.connextdds.CharSeq, *arg0: int*) → *rti.connextdds.CharSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.CharSeq, arg0: buffer) -> None

2. __init__(self: rti.connextdds.CharSeq) -> None

3. __init__(self: rti.connextdds.CharSeq, arg0: rti.connextdds.CharSeq) -> None

Copy constructor

4. __init__(self: rti.connextdds.CharSeq, arg0: Iterable) -> None

5. __init__(self: rti.connextdds.CharSeq, arg0: int) -> None

__iter__ (*self:* rti.connextdds.CharSeq) → Iterator

__len__ (*self:* rti.connextdds.CharSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.CharSeq, *arg0: int*) → *rti.connextdds.CharSeq*

__ne__ (*self:* rti.connextdds.CharSeq, *arg0:* rti.connextdds.CharSeq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__ (*self:* rti.connextdds.CharSeq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.CharSeq, *arg0: int*) → *rti.connextdds.CharSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.CharSeq, arg0: int, arg1: str) -> None
2. __setitem__(self: rti.connextdds.CharSeq, arg0: slice, arg1: rti.connextdds.CharSeq) -> None

Assign list elements using a slice object

__setstate__ (*self:* rti.connextdds.CharSeq, *arg0: bytes*) → None

append (*self:* rti.connextdds.CharSeq, *x: str*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.CharSeq) → None

Clear the contents

count (*self:* rti.connextdds.CharSeq, *x: str*) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.CharSeq, L: rti.connextdds.CharSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.CharSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.CharSeq, *i: int*, *x: str*) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.CharSeq) -> str

Remove and return the last item

2. pop(self: rti.connextdds.CharSeq, i: int) -> str

Remove and return the item at index i

remove (*self:* rti.connextdds.CharSeq, *x: str*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

resize (*self:* rti.connextdds.CharSeq, *arg0: int*) → None

resizes the vector to the given size

rti.connextdds.**CharType**

alias of *Int8Type*

class rti.connextdds.**CoherentAccess**

Bases: pybind11_object

__enter__ (*self:* rti.connextdds.CoherentAccess) → *rti.connextdds.CoherentAccess*

Start a context managed coherent access block.

__exit__ (*self:* rti.connextdds.CoherentAccess, *arg0: object*, *arg1: object*, *arg2: object*) → None

End a context managed coherent access block.

__init__ (*self:* rti.connextdds.CoherentAccess, *subscriber:* rti.connextdds.Subscriber) → None

Creating a CoherentAccess object indicates that the application is about to access the data samples in any of the DataReader objects attached to the provided Subscriber.

__module__ = 'rti.connextdds'

end (*self:* rti.connextdds.CoherentAccess) → None

Explicitly indicate that the application has finished accessing the data samples in DataReader objects managed by the Subscriber.

class rti.connextdds.**CoherentSet**

Bases: pybind11_object

__enter__ (*self:* rti.connextdds.CoherentSet) → *rti.connextdds.CoherentSet*

Context manage the CoherentSet.

__exit__ (*self:* rti.connextdds.CoherentSet, *arg0: object*, *arg1: object*, *arg2: object*) → None

__init__ (*self:* rti.connextdds.CoherentSet, *publisher:* rti.connextdds.Publisher) → None

Creating a CoherentSet object indicates that the application will begin a coherent set of modifications using DataWriter objects attached to the Publisher.

__module__ = 'rti.connextdds'

end (*self:* rti.connextdds.CoherentSet) → None

Explicitly terminate a coherent set initiated by the CoherentSet constructor.

class rti.connextdds.**CoherentSetInfo**

Bases: pybind11_object

UNKNOWN = <rti.connextdds.CoherentSetInfo object>

__eq__ (*self:* rti.connextdds.CoherentSetInfo, *arg0:* rti.connextdds.CoherentSetInfo) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.CoherentSetInfo) -> None

Create a default CoherentSetInfo.

2. __init__(self: rti.connextdds.CoherentSetInfo, group_guid: rti.connextdds.Guid, coherent_set_sequence_number: rti.connextdds.SequenceNumber, group_coherent_set_sequence_number: rti.connextdds.SequenceNumber, incomplete_coherent_set: bool) -> None

Create a CoherentSetInfo with the specified parameters.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.CoherentSetInfo, *arg0:* rti.connextdds.CoherentSetInfo) → bool

Test for inequality.

__str__ (*self:* rti.connextdds.CoherentSetInfo) → str

Convert CoherentSetInfo to str.

property coherent_set_sequence_number

The coherent set sequence number that identifies a sample as part of a DataWriter coherent set.

property group_coherent_set_sequence_number

The group coherent set sequence number that identifies a sample as part of a group coherent set.

property group_guid

Coherent set group.

property incomplete_coherent_set

The incomplete coherent set status.

class rti.connextdds.**CollectionType**

Bases: *DynamicType*

__eq__ (*self:* rti.connextdds.CollectionType, *arg0:* rti.connextdds.CollectionType) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.CollectionType, *arg0:* rti.connextdds.CollectionType) → bool

Test for inequality.

property content_type

Gets the type of elements of this collection.

class rti.connextdds.**CompressionIdMask**

Bases: pybind11_object

ALL = 0000000000000111

BZIP2 = 0000000000000010

DEFAULT_PUBLICATION = 0000000000000000

DEFAULT_SUBSCRIPTION = 0000000000000111

LZ4 = 0000000000000100

NONE = 0000000000000000

ZLIB = 0000000000000001

__and__(*self:* rti.connextdds.CompressionIdMask, *arg0:* rti.connextdds.CompressionIdMask) → *rti.connextdds.CompressionIdMask*

Bitwise logical AND of masks.

__bool__(*self:* rti.connextdds.CompressionIdMask) → bool

Test if any bits are set.

__contains__(*self:* rti.connextdds.CompressionIdMask, *arg0:* rti.connextdds.CompressionIdMask) → bool

__eq__(*self:* rti.connextdds.CompressionIdMask, *arg0:* rti.connextdds.CompressionIdMask) → bool

Compare masks for equality.

__getitem__(*self:* rti.connextdds.CompressionIdMask, *arg0: int*) → bool

Get individual mask bit.

__hash__ = None

__iand__(*self:* rti.connextdds.CompressionIdMask, *arg0:* rti.connextdds.CompressionIdMask) → *rti.connextdds.CompressionIdMask*

Set mask to logical AND with another mask.

__ilshift__(*self:* rti.connextdds.CompressionIdMask, *arg0: int*) → *rti.connextdds.CompressionIdMask*

Left shift bits in mask.

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.CompressionIdMask) -> None

Create a CompressionIdMask with no bits set.

2. __init__(self: rti.connextdds.CompressionIdMask, value: int) -> None

Creates a mask from the bits in an integer.

__int__ (*self:* rti.connextdds.CompressionIdMask) → int

Convert mask to int.

__ior__ (*self:* rti.connextdds.CompressionIdMask, *arg0:* rti.connextdds.CompressionIdMask) → *rti.connextdds.CompressionIdMask*

Set mask to logical OR with another mask.

__irshift__ (*self:* rti.connextdds.CompressionIdMask, *arg0: int*) → *rti.connextdds.CompressionIdMask*

Right shift bits in mask.

__ixor__ (*self:* rti.connextdds.CompressionIdMask, *arg0:* rti.connextdds.CompressionIdMask) → *rti.connextdds.CompressionIdMask*

Set mask to logical XOR with another mask.

__lshift__ (*self:* rti.connextdds.CompressionIdMask, *arg0: int*) → *rti.connextdds.CompressionIdMask*

Left shift bits in mask.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.CompressionIdMask, *arg0:* rti.connextdds.CompressionIdMask) → bool

Compare masks for inequality.

__or__ (*self:* rti.connextdds.CompressionIdMask, *arg0:* rti.connextdds.CompressionIdMask) → *rti.connextdds.CompressionIdMask*

Bitwise logical OR of masks.

__repr__ (*self:* rti.connextdds.CompressionIdMask) → str

Convert mask to string.

__rshift__ (*self:* rti.connextdds.CompressionIdMask, *arg0: int*) → *rti.connextdds.CompressionIdMask*

Right shift bits in mask.

__setitem__ (*self:* rti.connextdds.CompressionIdMask, *arg0: int*, *arg1: bool*) → None

Set individual mask bit

__str__ (*self:* rti.connextdds.CompressionIdMask) → str

Convert mask to string.

__xor__ (*self:* rti.connextdds.CompressionIdMask, *arg0:* rti.connextdds.CompressionIdMask) → *rti.connextdds.CompressionIdMask*

Bitwise logical XOR of masks.

property count

Returns the number of bits set in the mask.

flip(**args*, ***kwargs*)

Overloaded function.

1. flip(self: rti.connextdds.CompressionIdMask) -> rti.connextdds.CompressionIdMask

Flip all bits in the mask.

2. flip(self: rti.connextdds.CompressionIdMask, pos: int) -> rti.connextdds.CompressionIdMask

Flip the mask bit at the specified position.

reset(**args*, ***kwargs*)

Overloaded function.

1. reset(self: rti.connextdds.CompressionIdMask) -> rti.connextdds.CompressionIdMask

Clear all bits in the mask.

2. reset(self: rti.connextdds.CompressionIdMask, pos: int) -> rti.connextdds.CompressionIdMask

Clear the mask bit at the specified position.

set(**args*, ***kwargs*)

Overloaded function.

1. set(self: rti.connextdds.CompressionIdMask) -> rti.connextdds.CompressionIdMask

Set all bits in the mask.

2. set(self: rti.connextdds.CompressionIdMask, pos: int, value: bool = True) -> rti.connextdds.CompressionIdMask

Set the mask bit at the specified position to the provided value (default: true).

property size

Returns the number of bits in the mask type.

test(*self:* rti.connextdds.CompressionIdMask, *pos: int*) → bool

Test whether the mask bit at position "pos" is set.

test_all(*self:* rti.connextdds.CompressionIdMask) → bool

Test if all bits are set.

test_any(*self:* rti.connextdds.CompressionIdMask) → bool

Test if any bits are set.

test_none(*self:* rti.connextdds.CompressionIdMask) → bool

Test if none of the bits are set.

class rti.connextdds.**CompressionSettings**

Bases: pybind11_object

COMPRESSION_LEVEL_BEST_COMPRESSION = 10

COMPRESSION_LEVEL_BEST_SPEED = 1

COMPRESSION_LEVEL_DEFAULT = 10

__eq__(*self:* rti.connextdds.CompressionSettings, *arg0:* rti.connextdds.CompressionSettings) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.CompressionSettings) -> None

Create a CompressionSettings object with default settings.

2. __init__(self: rti.connextdds.CompressionSettings, compression_ids: rti.connextdds.CompressionIdMask) -> None

Creates an instance with the given compression_ids.

3. __init__(self: rti.connextdds.CompressionSettings, compression_ids: rti.connextdds.CompressionIdMask, writer_compression_level: int, writer_compression_threshold: int) -> None

Creates an instance with the given compression_ids, writer_compression_level and writer_compression_threshold.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.CompressionSettings, *arg0:* rti.connextdds.CompressionSettings) → bool

Test for inequality.

property compression_ids

Compression ID settings.

property writer_compression_level

Writer compression level.

property writer_compression_threshold

Writer compression threshold

class rti.connextdds.**Condition**

Bases: *ICondition*

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

class rti.connextdds.**ConditionSeq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.ConditionSeq, *arg0:* rti.connextdds.ConditionSeq) → *rti.connextdds.ConditionSeq*

__bool__ (*self:* rti.connextdds.ConditionSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.ConditionSeq, *x:* rti.connextdds.Condition) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.ConditionSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.ConditionSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.ConditionSeq, *arg0:* rti.connextdds.ConditionSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.ConditionSeq, s: slice) -> rti.connextdds.ConditionSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.ConditionSeq, arg0: int) -> rti.connextdds.Condition

__hash__ = None

__iadd__ (*self:* rti.connextdds.ConditionSeq, *arg0:* rti.connextdds.ConditionSeq) → *rti.connextdds.ConditionSeq*

__imul__ (*self:* rti.connextdds.ConditionSeq, *arg0: int*) → *rti.connextdds.ConditionSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ConditionSeq) -> None

2. __init__(self: rti.connextdds.ConditionSeq, arg0: rti.connextdds.ConditionSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.ConditionSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.ConditionSeq) → Iterator

__len__ (*self:* rti.connextdds.ConditionSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.ConditionSeq, *arg0: int*) → *rti.connextdds.ConditionSeq*

__ne__ (*self:* rti.connextdds.ConditionSeq, *arg0:* rti.connextdds.ConditionSeq) → bool

__rmul__ (*self:* rti.connextdds.ConditionSeq, *arg0: int*) → *rti.connextdds.ConditionSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.ConditionSeq, arg0: int, arg1: rti.connextdds.Condition) -> None
2. __setitem__(self: rti.connextdds.ConditionSeq, arg0: slice, arg1: rti.connextdds.ConditionSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.ConditionSeq, *x:* rti.connextdds.Condition) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.ConditionSeq) → None

Clear the contents

count (*self:* rti.connextdds.ConditionSeq, *x:* rti.connextdds.Condition) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.ConditionSeq, L: rti.connextdds.ConditionSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.ConditionSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.ConditionSeq, *i: int*, *x:* rti.connextdds.Condition) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.ConditionSeq) -> rti.connextdds.Condition

Remove and return the last item

2. pop(self: rti.connextdds.ConditionSeq, i: int) -> rti.connextdds.Condition

Remove and return the item at index `i`

remove (*self:* rti.connextdds.ConditionSeq, *x:* rti.connextdds.Condition) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**ContentFilterBase**

Bases: `pybind11_object`

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

class rti.connextdds.**ContentFilterProperty**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.ContentFilterProperty, *arg0:* rti.connextdds.ContentFilterProperty) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.ContentFilterProperty, *arg0:* rti.connextdds.ContentFilterProperty) → bool

Test for inequality.

property content_filter_topic_name

The ContentFilteredTopic filter parameters.

property filter_class_name

Identifies the filter class this filter belongs to.

property filter_expression

The filter expression.

property related_topic_name

The name of the ContentFilteredTopic's related Topic.

class rti.connextdds.**ContentFilteredTopic**

Bases: *ITopicDescription*, *IAnyTopic*

__eq__(*self:* rti.connextdds.ContentFilteredTopic, *arg0:* rti.connextdds.ContentFilteredTopic) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ContentFilteredTopic, topic: rti.connextdds.Topic, name: str, contentfilter: rti.connextdds.Filter) -> None

Create a ContentFilteredTopic with a name and Filter.

2. __init__(self: rti.connextdds.ContentFilteredTopic, topic_description: rti.connextdds.ITopicDescription) -> None

Cast a TopicDescription to a ContentFilteredTopic.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ContentFilteredTopic, *arg0:* rti.connextdds.ContentFilteredTopic) → bool

Test for inequality.

append_to_expression_parameter (*self:* rti.connextdds.ContentFilteredTopic, *index: int*, *extension: str*) → None

Append the extension to the end of parameter at the provided index, separated by a comma.

property filter_expression

Get the filter expression

property filter_parameters

Get/set the filter parameters.

static find (*participant:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.ContentFilteredTopic*]

Look up a ContentFilteredTopic by its name in the DomainParticipant.

remove_from_expression_parameter (*self:* rti.connextdds.ContentFilteredTopic, *index: int*, *remove_term: str*) → None

Removes the specified term from the parameter at the provided index.

set_filter (*self:* rti.connextdds.ContentFilteredTopic, *arg0:* rti.connextdds.Filter) → None

Set the filter.

property topic

Get the underlying Topic.

class rti.connextdds.**ContentFilteredTopicSeq**

Bases: pybind11_object

__add__ (*self: List[*rti.connextdds.ContentFilteredTopic*]*, *arg0: List[*rti.connextdds.ContentFilteredTopic*]*) → List[*rti.connextdds.ContentFilteredTopic*]

__bool__ (*self: List[*rti.connextdds.ContentFilteredTopic*]*) → bool

Check whether the list is nonempty

__contains__ (*self: List[*rti.connextdds.ContentFilteredTopic*]*, *x:* rti.connextdds.ContentFilteredTopic) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: List[rti.connextdds.ContentFilteredTopic], arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: List[rti.connextdds.ContentFilteredTopic], arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self: List[*rti.connextdds.ContentFilteredTopic*]*, *arg0: List[*rti.connextdds.ContentFilteredTopic*]*) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: List[rti.connextdds.ContentFilteredTopic], s: slice) -> List[rti.connextdds.ContentFilteredTopic]

Retrieve list elements using a slice object

2. __getitem__(self: List[rti.connextdds.ContentFilteredTopic], arg0: int) -> rti.connextdds.ContentFilteredTopic

__hash__ = None

__iadd__ (*self: List[*rti.connextdds.ContentFilteredTopic*]*, *arg0: List[*rti.connextdds.ContentFilteredTopic*]*) → List[*rti.connextdds.ContentFilteredTopic*]

__imul__ (*self: List[*rti.connextdds.ContentFilteredTopic*]*, *arg0: int*) → List[*rti.connextdds.ContentFilteredTopic*]

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ContentFilteredTopicSeq) -> None

2. __init__(self: rti.connextdds.ContentFilteredTopicSeq, arg0: List[rti.connextdds.ContentFilteredTopic]) -> None

Copy constructor

3. __init__(self: rti.connextdds.ContentFilteredTopicSeq, arg0: Iterable) -> None

__iter__ (*self: List[*rti.connextdds.ContentFilteredTopic*]*) → Iterator

__len__ (*self: List[*rti.connextdds.ContentFilteredTopic*]*) → int

__module__ = 'rti.connextdds'

__mul__ (*self: List[*rti.connextdds.ContentFilteredTopic*]*, *arg0: int*) → List[*rti.connextdds.ContentFilteredTopic*]

__ne__ (*self: List[*rti.connextdds.ContentFilteredTopic*]*, *arg0: List[*rti.connextdds.ContentFilteredTopic*]*) → bool

__rmul__ (*self: List[*rti.connextdds.ContentFilteredTopic*]*, *arg0: int*) → List[*rti.connextdds.ContentFilteredTopic*]

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: List[rti.connextdds.ContentFilteredTopic], arg0: int, arg1: rti.connextdds.ContentFilteredTopic) -> None

2. __setitem__(self: List[rti.connextdds.ContentFilteredTopic], arg0: slice, arg1: List[rti.connextdds.ContentFilteredTopic]) -> None

Assign list elements using a slice object

append (*self: List[*rti.connextdds.ContentFilteredTopic*]*, *x:* rti.connextdds.ContentFilteredTopic) → None

Add an item to the end of the list

clear (*self: List[*rti.connextdds.ContentFilteredTopic*]*) → None

Clear the contents

count (*self: List[*rti.connextdds.ContentFilteredTopic*]*, *x:* rti.connextdds.ContentFilteredTopic) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: List[rti.connextdds.ContentFilteredTopic], L: List[rti.connextdds.ContentFilteredTopic]) -> None

Extend the list by appending all the items in the given list

2. extend(self: List[rti.connextdds.ContentFilteredTopic], L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self: List[*rti.connextdds.ContentFilteredTopic*]*, *i: int*, *x:* rti.connextdds.ContentFilteredTopic) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: List[rti.connextdds.ContentFilteredTopic]) -> rti.connextdds.ContentFilteredTopic

Remove and return the last item

2. pop(self: List[rti.connextdds.ContentFilteredTopic], i: int) -> rti.connextdds.ContentFilteredTopic

Remove and return the item at index `i`

remove (*self: List[*rti.connextdds.ContentFilteredTopic*]*, *x:* rti.connextdds.ContentFilteredTopic) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**Cookie**

Bases: `pybind11_object`

Unique identifier for a written data sample in the form of a sequence of bytes.

__eq__ (*self:* rti.connextdds.Cookie, *arg0:* rti.connextdds.Cookie) → bool

Compare Cookies for equality

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Cookie) -> None

Creates an empty Cookie.

2. __init__(self: rti.connextdds.Cookie, bytes: rti.connextdds.ByteVector) -> None

Cookie from bytes.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.Cookie, *arg0:* rti.connextdds.Cookie) → bool

Compare Cookies for inequality

property value

Retrieve the bytes of the Cookie's contents.

class rti.connextdds.**CookieSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.CookieSeq, *arg0:* rti.connextdds.CookieSeq) → *rti.connextdds.CookieSeq*

__bool__(*self:* rti.connextdds.CookieSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.CookieSeq, *x:* rti.connextdds.Cookie) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.CookieSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.CookieSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.CookieSeq, *arg0:* rti.connextdds.CookieSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.CookieSeq, s: slice) -> rti.connextdds.CookieSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.CookieSeq, arg0: int) -> rti.connextdds.Cookie

__hash__ = None

__iadd__ (*self:* rti.connextdds.CookieSeq, *arg0:* rti.connextdds.CookieSeq) → *rti.connextdds.CookieSeq*

__imul__ (*self:* rti.connextdds.CookieSeq, *arg0: int*) → *rti.connextdds.CookieSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.CookieSeq) -> None
2. __init__(self: rti.connextdds.CookieSeq, arg0: rti.connextdds.CookieSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.CookieSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.CookieSeq) → Iterator

__len__ (*self:* rti.connextdds.CookieSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.CookieSeq, *arg0: int*) → *rti.connextdds.CookieSeq*

__ne__ (*self:* rti.connextdds.CookieSeq, *arg0:* rti.connextdds.CookieSeq) → bool

__repr__ (*self:* rti.connextdds.CookieSeq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.CookieSeq, *arg0: int*) → *rti.connextdds.CookieSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.CookieSeq, arg0: int, arg1: rti.connextdds.Cookie) -> None
2. __setitem__(self: rti.connextdds.CookieSeq, arg0: slice, arg1: rti.connextdds.CookieSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.CookieSeq, *x:* rti.connextdds.Cookie) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.CookieSeq) → None

Clear the contents

count (*self:* rti.connextdds.CookieSeq, *x:* rti.connextdds.Cookie) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.CookieSeq, L: rti.connextdds.CookieSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.CookieSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.CookieSeq, *i: int*, *x:* rti.connextdds.Cookie) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.CookieSeq) -> rti.connextdds.Cookie

Remove and return the last item

2. pop(self: rti.connextdds.CookieSeq, i: int) -> rti.connextdds.Cookie

Remove and return the item at index `i`

remove (*self:* rti.connextdds.CookieSeq, *x:* rti.connextdds.Cookie) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**CookieVector**

Bases: pybind11_object

A DDS standard container with functionality similar to a C++ vector.

__eq__ (*self:* rti.connextdds.CookieVector, *arg0:* rti.connextdds.CookieVector) → bool

Compare CookieVectors for equality.

__getitem__ (*self:* rti.connextdds.CookieVector, *arg0: int*) → *rti.connextdds.Cookie*

Get the value at the specified index.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.CookieVector) -> None

Create an empty CookieVector

2. __init__(self: rti.connextdds.CookieVector, size: int) -> None

Create a CookieVector with a preallocated size.

3. __init__(self: rti.connextdds.CookieVector, vector: rti.connextdds.CookieVector) -> None

Create a copy from another CookieVector.

4. __init__(self: rti.connextdds.CookieVector, arg0: Iterable) -> None

5. __init__(self: rti.connextdds.CookieVector, list: rti.connextdds.CookieSeq) -> None

Create a CookieVector from a list of values.

__iter__ (*self:* rti.connextdds.CookieVector) → Iterator

Iterate over the contents of the vector.

__len__ (*self:* rti.connextdds.CookieVector) → int

Get the length of the CookieVector.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.CookieVector, *arg0:* rti.connextdds.CookieVector) → bool

Compare CookieVectors for inequality.

__setitem__ (*self:* rti.connextdds.CookieVector, *arg0: int*, *arg1:* rti.connextdds.Cookie) → None

Set the value at the specified index.

clear (*self:* rti.connextdds.CookieVector) → None

Resize CookieVector to 0.

resize (*self:* rti.connextdds.CookieVector, *size: int*) → None

Resize CookieVector.

class rti.connextdds.**DataReader**

Bases: *IDataReader*

class LoanedSample

Bases: pybind11_object

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DataReader.LoanedSample) -> None

Basic constructor

2. __init__(self: rti.connextdds.DataReader.LoanedSample, data: rti.connextdds.UserDataSample, info: rti.connextdds.SampleInfo) -> None

Construct LoanedSample with data and info.

__iter__ (*self:* rti.connextdds.DataReader.LoanedSample) → object

__module__ = 'rti.connextdds'

property data

Get the data associated with the sample.

property info

Get the info associated with the sample.

class LoanedSamples

Bases: pybind11_object

__enter__ (*self:* rti.connextdds.DataReader.LoanedSamples) → *rti.connextdds.DataReader.LoanedSamples*

Enter a context for the loaned samples, loan returned on context exit.

__exit__ (*self:* rti.connextdds.DataReader.LoanedSamples, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for the loaned samples, returning the resources.

__getitem__(*self:* rti.connextdds.DataReader.LoanedSamples, *arg0: int*) → *rti.connextdds.DataReader.LoanedSample*

Access a LoanedSample object in an array-like syntax

__init__(*self:* rti.connextdds.DataReader.LoanedSamples) → None

Create an empty LoanedSamples object.

__iter__(*self:* rti.connextdds.DataReader.LoanedSamples) → Iterator

__len__(*self:* rti.connextdds.DataReader.LoanedSamples) → int

Get the number of samples in the loan.

__module__ = 'rti.connextdds'

property length

Get the number of samples in the loan.

return_loan(*self:* rti.connextdds.DataReader.LoanedSamples) → None

Returns the loan to the DataReader.

class Selector

Bases: pybind11_object

__init__(*self:* rti.connextdds.DataReader.Selector, *datareader:* rti.connextdds.DataReader) → None

Create a Selector for a DataReader to read/take based on selected conditions

__module__ = 'rti.connextdds'

condition(*self:* rti.connextdds.DataReader.Selector, *condition:* rti.connextdds.IReadCondition) → *rti.connextdds.DataReader.Selector*

Select samples based on a ReadCondition.

content(*self:* rti.connextdds.DataReader.Selector, *query:* rti.connextdds.Query) → *rti.connextdds.DataReader.Selector*

Select samples based on a Query.

instance(*self:* rti.connextdds.DataReader.Selector, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.DataReader.Selector*

Select a specific instance to read/take.

max_samples(*self:* rti.connextdds.DataReader.Selector, *max: int*) → *rti.connextdds.DataReader.Selector*

Limit the number of samples read/taken by the Select.

next_instance(*self:* rti.connextdds.DataReader.Selector, *previous:* rti.connextdds.InstanceHandle) → *rti.connextdds.DataReader.Selector*

Select the instance after the specified instance to read/take.

read(*self:* rti.connextdds.DataReader.Selector) → list

Read copies of all available data and info based on Selector settings.

read_data (*self:* rti.connextdds.DataReader.Selector) → list

Read copies of all available valid data based on Selector settings.

read_loaned (*self:* rti.connextdds.DataReader.Selector) → *rti.connextdds.DataReader.LoanedSamples*

(Advanced) Read data as a collection of loaned data in C format and info objects based on Selector settings

state (*self:* rti.connextdds.DataReader.Selector, *state:* rti.connextdds.DataState) → *rti.connextdds.DataReader.Selector*

Select samples with a specified data state.

take (*self:* rti.connextdds.DataReader.Selector) → list

Take copies of all available data and info based on Selector settings.

take_data (*self:* rti.connextdds.DataReader.Selector) → list

Take copies of all available valid data based on Selector settings.

take_loaned (*self:* rti.connextdds.DataReader.Selector) → *rti.connextdds.DataReader.LoanedSamples*

(Advanced) Take data as a collection of loaned data in C format and info objects based on Selector settings

__enter__ (*self:* rti.connextdds.DataReader) → *rti.connextdds.DataReader*

Enter a context for this DataReader, to be cleaned up on exiting context

__eq__ (*self:* rti.connextdds.DataReader, *arg0:* rti.connextdds.DataReader) → bool

Test for equality.

__exit__ (*self:* rti.connextdds.DataReader, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for this DataReader, cleaning up resources.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DataReader, topic: rti.connextdds.Topic) -> None

Create a DataReader in the implicit subscriber with default QoS.

2. __init__(self: rti.connextdds.DataReader, sub: rti.connextdds.Subscriber, topic: rti.connextdds.Topic) -> None

Create a DataReader in a subscriber with default QoS.

3. __init__(self: rti.connextdds.DataReader, topic: rti.connextdds.Topic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader in the implicit subscriber with specific QoS and a listener.

4. __init__(self: rti.connextdds.DataReader, sub: rti.connextdds.Subscriber, topic: rti.connextdds.Topic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader in a subscriber with specific QoS and a listener.

5. __init__(self: rti.connextdds.DataReader, cft: rti.connextdds.ContentFilteredTopic) -> None

Create a DataReader with a ContentFilteredTopic in the participant's implicit subscriber with default QoS.

6. __init__(self: rti.connextdds.DataReader, sub: rti.connextdds.Subscriber, cft: rti.connextdds.ContentFilteredTopic) -> None

Create a DataReader with a ContentFilteredTopic in a subscriber with default QoS.

7. __init__(self: rti.connextdds.DataReader, cft: rti.connextdds.ContentFilteredTopic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader with a ContentFilteredTopic in the implicit subscriber with specific QoS.

8. __init__(self: rti.connextdds.DataReader, sub: rti.connextdds.Subscriber, cft: rti.connextdds.ContentFilteredTopic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader with a ContentFilteredTopic in a subscribrer with specific QoS.

9. __init__(self: rti.connextdds.DataReader, reader: rti.connextdds.IAnyDataReader) -> None

Get a typed DataReader from an AnyDataReader.

10. __init__(self: rti.connextdds.DataReader, entity: rti.connextdds.IEntity) -> None

Get a typed DataReader from an Entity.

__lshift__(*self:* rti.connextdds.DataReader, *arg0:* rti.connextdds.DataReaderQos) → *rti.connextdds.DataReader*

Set the DataReaderQos for this DataReader.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DataReader, *arg0:* rti.connextdds.DataReader) → bool

Test for inequality.

__rshift__(*self:* rti.connextdds.DataReader, *arg0:* rti.connextdds.DataReaderQos) → *rti.connextdds.DataReader*

Get the DataReaderQos from this DataReader

acknowledge_all(**args*, ***kwargs*)

Overloaded function.

1. acknowledge_all(self: rti.connextdds.DataReader) -> None

Acknowledge all previously accessed samples.

2. acknowledge_all(self: rti.connextdds.DataReader, arg0: rti.connextdds.AckResponseData) -> None

Acknowledge all previously accessed samples.

acknowledge_sample(**args*, ***kwargs*)

Overloaded function.

1. acknowledge_sample(self: rti.connextdds.DataReader, sample_info: rti.connextdds.SampleInfo) -> None

Acknowledge a single sample.

2. acknowledge_sample(self: rti.connextdds.DataReader, sample_info: rti.connextdds.SampleInfo, ack_response_data: rti.connextdds.AckResponseData) -> None

Acknowledge a single sample with ack response data.

close(*self:* rti.connextdds.DataReader) → None

Close this DataReader.

property datareader_cache_status

Get the DataReaderCacheStatus for the DataReader.

property datareader_protocol_status

Get the DataReaderProtocolStatus for the DataReader.

property default_filter_state

Returns the filter state for the read/take operations.

static find_all_by_topic(*subscriber:* rti.connextdds.Subscriber, *topic_name: str*) → List[*rti.connextdds.DataReader*]

Retrieve all DataReaders for the given topic name in the subscriber.

static find_by_name(**args*, ***kwargs*)

Overloaded function.

1. find_by_name(participant: rti.connextdds.DomainParticipant, name: str) -> Optional[rti.connextdds.DataReader]

Find DataReader in DomainParticipant with the DataReader's name, returning the first found.

2. find_by_name(subscriber: rti.connextdds.Subscriber, name: str) -> Optional[rti.connextdds.DataReader]

Find DataReader in Subscriber with the DataReader's name, returning the first found.

static find_by_topic(*subscriber:* rti.connextdds.Subscriber, *name: str*) → Optional[*rti.connextdds.DataReader*]

Find DataReader in Subscriber with a topic name, returning the first found.

is_matched_publication_alive(*self:* rti.connextdds.DataReader, *arg0:* rti.connextdds.InstanceHandle) → bool

Check if a matched publication is alive.

key_value (*self:* rti.connextdds.DataReader, *handle:* rti.connextdds.InstanceHandle) → object

Retrieve the instance key that corresponds to an instance handle.

property listener

Gets or sets the listener with StatusMask.ALL

property liveliness_changed_status

Get the LivelinessChangedStatus for this DataReader.

lookup_instance (**args*, ***kwargs*)

Overloaded function.

1. lookup_instance(self: rti.connextdds.DataReader, key_holder: rti.connextdds.UserDataSample) -> rti.connextdds.InstanceHandle

Retrieve the instance handle that corresponds to an instance key_holder

2. lookup_instance(self: rti.connextdds.DataReader, key_holder: object) -> rti.connextdds.InstanceHandle

Retrieve the instance handle that corresponds to an instance key_holder

matched_publication_data (*self:* rti.connextdds.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.PublicationBuiltinTopicData*

Get the PublicationBuiltinTopicData for a publication matched to this DataReader.

matched_publication_datareader_protocol_status (*self:* rti.connextdds.DataReader, *publication_handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.DataReaderProtocolStatus*

Get the DataReaderProtocolStatus for the DataReader based on the matched publication handle.

matched_publication_participant_data (*self:* rti.connextdds.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ParticipantBuiltinTopicData*

Get the ParticipantBuiltinTopicData for a publication matched to this DataReader.

property matched_publications

Get a copy of the list of the currently matched publication handles.

property qos

The DataReaderQos for this DataReader.

This property's getter returns a deep copy.

read (*self:* rti.connextdds.DataReader) → list

Read copies of all available data and info

read_data (*self:* rti.connextdds.DataReader) → list

Read copies of all available valid data

read_loaned (*self:* rti.connextdds.DataReader) → *rti.connextdds.DataReader.LoanedSamples*

(Advanced) Read all data as a collection of loaned data in C format and info objects

property requested_deadline_missed_status

Get the RequestedDeadlineMissed status for the DataReader

property requested_incompatible_qos_status

Get the RequestedIncompatibleQosStatus for the DataReader.

property sample_lost_status

Get the SampleLostStatus for the DataReader.

property sample_rejected_status

Get the SampleRejectedStatus for the DataReader.

select (*self:* rti.connextdds.DataReader) → *rti.connextdds.DataReader.Selector*

Get a Selector to perform complex data selections, such as per-instance selection, content, and status filtering.

set_listener (*self:* rti.connextdds.DataReader, *listener:* rti.connextdds.DataReaderListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and associated event mask.

property subscriber

Returns the parent Subscriber of the DataReader.

property subscription_matched_status

Get the SubscriptionMatchedStatus for the DataReader.

take (*self:* rti.connextdds.DataReader) → list

Take copies of all available data and info

take_async (*condition: Optional[*ReadCondition*] = None*)

take_data (*self:* rti.connextdds.DataReader) → list

Take copies of all available valid data

take_data_async (*condition: Optional[*ReadCondition*] = None*)

take_loaned (*self:* rti.connextdds.DataReader) → *rti.connextdds.DataReader.LoanedSamples*

(Advanced) Take all data as a collection of loaned data in C format and info objects

property topic_description

Returns the TopicDescription associated with the DataReader.

property topic_name

Get the topic name associated with this DataReader.

property type_name

Get the type name associated with this DataReader.

wait_for_historical_data(*self:* rti.connextdds.DataReader, *max_wait:* rti.connextdds.Duration) → None

Waits until all "historical" data is received for DataReaders that have a non-VOLATILE Durability Qos kind.

wait_for_historical_data_async(*self:* rti.connextdds.DataReader, *max_wait:* rti.connextdds.Duration) → object

Waits until all "historical" data is received for DataReaders that have a non-VOLATILE Durability Qos kind. This call is awaitable and only for use with asyncio.

class rti.connextdds.**DataReaderCacheStatus**

Bases: pybind11_object

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property alive_instance_count

The number of instances in the DataReader's queue with an instance state equal to InstanceState.ALIVE.

property alive_instance_count_peak

The highest value of DataReaderCacheStatus.alive_instance_count over the lifetime of the DataReader.

property compressed_sample_count

The number of received compressed samples.

property content_filter_dropped_sample_count

The number of user samples filtered by the DataReader due to Content-Filtered Topics.

property detached_instance_count

The number of instances in the DataReader's queue with an instance state equal to InstanceState.NOT_ALIVE_DISPOSED.

property detached_instance_count_peak

The highest value of DataReaderCacheStatus.detached_instance_count over the lifetime of the DataReader.

property disposed_instance_count

The number of instances in the DataReader's queue with an instance state equal to InstanceState.NOT_ALIVE_DISPOSED.

property disposed_instance_count_peak

The number of minimal instance states currently being maintained in the DataReader's queue.

property expired_dropped_sample_count

The number of samples expired by the DataReader due to Lifespan QoS or the autopurge sample delays.

property no_writers_instance_count

The number of instances in the DataReader's queue with an instance state equal to InstanceState.NOT_ALIVE_NO_WRITERS.

property no_writers_instance_count_peak

The highest value of DataReaderCacheStatus.no_writers_instance_count over the lifetime of the DataReader.

property old_source_timestamp_dropped_sample_count

The number of samples dropped as a result of receiving a sample older than the last one, using DestinationOrderKind.BY_SOURCE_TIMESTAMP.

property ownership_dropped_sample_count

The number of samples dropped as a result of receiving a sample from a DataWriter with a lower strength, using Exclusive Ownership.

property replaced_dropped_sample_count

The number of samples replaced by the DataReader due to HistoryKind.KEEP_LAST replacement.

property sample_count

The number of samples in the DataReader's queue.

property sample_count_peak

The highest number of samples in the DataReader's queue over the lifetime of the reader.

property time_based_filter_dropped_sample_count

The number of user samples filtered by the DataReader due to TimeBasedFilter QoS.

property tolerance_source_timestamp_dropped_sample_count

The number of samples dropped as a result of receiving a sample in the future, using DestinationOrderKind.BY_SOURCE_TIMESTAMP.

property total_samples_dropped_by_instance_replacement

The number of samples with sample state SampleState.NOT_READ that were dropped when removing an instance due to instance replacement. See DataReaderResourceLimits.instance_replacement for more details about when instances are replaced.

property virtual_duplicate_dropped_sample_count

The number of virtual duplicate samples dropped by the DataReader. A sample is a virtual duplicate if it has the same identity (Virtual Writer GUID and Virtual Sequence Number) as a previously received sample.

property writer_removed_batch_sample_dropped_sample_count

The number of batch samples received by the DataReader that were marked as removed by the DataWriter.

class rti.connextdds.**DataReaderInstanceRemovalKind**

Bases: pybind11_object

ANY_INSTANCE = <DataReaderInstanceRemovalKind.ANY_INSTANCE: 3>

class DataReaderInstanceRemovalKind

Bases: pybind11_object

Members:

NO_INSTANCE : No instance can be removed.

EMPTY_INSTANCES : Only empty instances can be removed

FULLY_PROCESSED_INSTANCES : Only fully-processed instances can be removed.

ANY_INSTANCE : Any instance can be removed.

ANY_INSTANCE = <DataReaderInstanceRemovalKind.ANY_INSTANCE: 3>

EMPTY_INSTANCES =
<DataReaderInstanceRemovalKind.EMPTY_INSTANCES: 1>

FULLY_PROCESSED_INSTANCES =
<DataReaderInstanceRemovalKind.FULLY_PROCESSED_INSTANCES: 2>

NO_INSTANCE = <DataReaderInstanceRemovalKind.NO_INSTANCE: 0>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.DataReaderInstanceRemovalKind.DataReaderInstanceRemovalKind) → int

__init__(*self:* rti.connextdds.DataReaderInstanceRemovalKind.DataReaderInstanceRemovalKind, *value: int*) → None

__int__(*self:* rti.connextdds.DataReaderInstanceRemovalKind.DataReaderInstanceRemovalKind) → int

__members__ = {'ANY_INSTANCE':
<DataReaderInstanceRemovalKind.ANY_INSTANCE: 3>,
'EMPTY_INSTANCES':
<DataReaderInstanceRemovalKind.EMPTY_INSTANCES: 1>,
'FULLY_PROCESSED_INSTANCES':
<DataReaderInstanceRemovalKind.FULLY_PROCESSED_INSTANCES: 2>,
'NO_INSTANCE': <DataReaderInstanceRemovalKind.NO_INSTANCE: 0>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.DataReaderInstanceRemovalKind.DataReaderInstanceRemovalKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

EMPTY_INSTANCES = <DataReaderInstanceRemovalKind.EMPTY_INSTANCES: 1>

FULLY_PROCESSED_INSTANCES = <DataReaderInstanceRemovalKind.FULLY_PROCESSED_INSTANCES: 2>

NO_INSTANCE = <DataReaderInstanceRemovalKind.NO_INSTANCE: 0>

__eq__(*self:* rti.connextdds.DataReaderInstanceRemovalKind, *arg0:* rti.connextdds.DataReaderInstanceRemovalKind) → bool

Apply operator to underlying enumerated values.

__ge__(*self:* rti.connextdds.DataReaderInstanceRemovalKind, *arg0:* rti.connextdds.DataReaderInstanceRemovalKind) → bool

Apply operator to underlying enumerated values.

__gt__(*self:* rti.connextdds.DataReaderInstanceRemovalKind, *arg0:* rti.connextdds.DataReaderInstanceRemovalKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DataReaderInstanceRemovalKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.DataReaderInstanceRemovalKind, arg0: rti.connextdds.DataReaderInstanceRemovalKind.DataReaderInstanceRemovalKind) -> None

Copy constructor.

__int__(*self:* rti.connextdds.DataReaderInstanceRemovalKind) → *rti.connextdds.DataReaderInstanceRemovalKind.DataReaderInstanceRemovalKind*

Int conversion.

__le__ (*self:* rti.connextdds.DataReaderInstanceRemovalKind, *arg0:* rti.connextdds.DataReaderInstanceRemovalKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.DataReaderInstanceRemovalKind, *arg0:* rti.connextdds.DataReaderInstanceRemovalKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DataReaderInstanceRemovalKind, *arg0:* rti.connextdds.DataReaderInstanceRemovalKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.DataReaderInstanceRemovalKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**DataReaderListener**

Bases: pybind11_object

__init__ (*self:* rti.connextdds.DataReaderListener) → None

__module__ = 'rti.connextdds'

on_data_available (*self:* rti.connextdds.DataReaderListener, *arg0:* rti.connextdds.DataReader) → None

Data available callback.

on_liveliness_changed (*self:* rti.connextdds.DataReaderListener, *arg0:* rti.connextdds.DataReader, *arg1:* rti.connextdds.LivelinessChangedStatus) → None

Liveliness changed callback.

on_requested_deadline_missed (*self:* rti.connextdds.DataReaderListener, *arg0:* rti.connextdds.DataReader, *arg1:* rti.connextdds.RequestedDeadlineMissedStatus) → None

Requested deadline missed callback.

on_requested_incompatible_qos (*self:* rti.connextdds.DataReaderListener, *arg0:* rti.connextdds.DataReader, *arg1:* rti.connextdds.RequestedIncompatibleQosStatus) → None

Requested incompatible QoS callback.

on_sample_lost (*self:* rti.connextdds.DataReaderListener, *arg0:* rti.connextdds.DataReader, *arg1:* rti.connextdds.SampleLostStatus) → None

Sample lost callback.

on_sample_rejected(*self:* rti.connextdds.DataReaderListener, *arg0:* rti.connextdds.DataReader, *arg1:* rti.connextdds.SampleRejectedStatus) → None

Sample rejected callback.

on_subscription_matched(*self:* rti.connextdds.DataReaderListener, *arg0:* rti.connextdds.DataReader, *arg1:* rti.connextdds.SubscriptionMatchedStatus) → None

Subscription matched callback.

class rti.connextdds.**DataReaderProtocol**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.DataReaderProtocol, *arg0:* rti.connextdds.DataReaderProtocol) → bool

Compare for equality.

__hash__ = None

__init__(*self:* rti.connextdds.DataReaderProtocol) → None

Create a default DataReaderProtocol policy.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DataReaderProtocol, *arg0:* rti.connextdds.DataReaderProtocol) → bool

Compare for inequality.

property disable_positive_acks

Get/set the boolean for whether the reader will use send positive acknowledgments.

property expects_inline_qos

Get/set the boolean for whether the reader will expect inline QoS with each sample.

property propagate_dispose_of_unregistered_instances

Get/set the boolean for whether an instance can move to the not_alive_disposed state without being in the alive state.

property propagate_unregister_of_disposed_instances

Get/set the boolean for whether an instance can move to the not_alive_no_writers state without being in the alive state.

property rtps_object_id

Get/set the RTPS object ID.

property rtps_reliable_reader

Get/set the reliable reader protocol settings.

property virtual_guid

Get/set a copy of the virtual GUID.

class rti.connextdds.**DataReaderProtocolStatus**

Bases: pybind11_object

Information about the DataReader's protocol status.

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property dropped_fragment_count

The number of DATA_FRAG messages that have been dropped by a DataReader.

property duplicate_sample_bytes

The number of bytes of sample data from a remote DataWriter received, not for the first time, by a local DataReader.

property duplicate_sample_count

The number of samples from a remote DataWriter received, not for the first time, by a local DataReader.

property first_available_sample_sequence_number

Sequence number of the first available sample in a matched DataWriters reliability queue.

property last_available_sample_sequence_number

Sequence number of the last available sample in a matched Datawriter's reliability queue.

property last_committed_sample_sequence_number

Sequence number of the newest sample received from the matched DataWriter committed to the DataReader's queue.

property out_of_range_rejected_sample_count

The number of samples dropped by the DataReader due to received window is full and the sample is out-of-order.

property reassembled_sample_count

The number of fragmented samples that have been reassembled by a DataReader.

property received_fragment_count

The number of DATA_FRAG messages that have been received by this DataReader.

property received_gap_bytes

The number of bytes of GAP data received from remote DataWriter to this DataReader.

property received_gap_count

The number of GAPs received from remote DataWriter to this DataReader.

property received_heartbeat_bytes

The number of bytes of Heartbeat data from a remote DataWriter received by a local DataReader.

property received_heartbeat_count

The number of Heartbeats from a remote DataWriter received by a local DataReader.

property received_sample_bytes

The number of bytes from samples received by a DataReader.

property received_sample_count

The number of samples received by a DataReader.

property rejected_sample_count

The number of times samples were rejected due to exceptions in the receive path.

property sent_ack_bytes

The number of bytes of ACK data sent from a local DataReader to a matching remote DataWriter.

property sent_ack_count

The number of ACKs sent from a local DataReader to a matching remote DataWriter.

property sent_nack_bytes

The number of bytes of NACK data sent from a local DataReader to a matching remote DataWriter.

property sent_nack_count

The number of NACKs sent from a local DataReader to a matching remote DataWriter.

property sent_nack_fragment_bytes

The number of NACK fragment bytes that have been sent from a DataReader to a DataWriter.

property sent_nack_fragment_count

The number of NACK fragments that have been sent from a DataReader to a DataWriter.

property uncommitted_sample_count

Number of received samples that are not yet available to be read or taken, due to being received out of order.

class rti.connextdds.**DataReaderQos**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.DataReaderQos, *arg0:* rti.connextdds.DataReaderQos) → bool

Test for equality

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DataReaderQos) -> None

Create a DataReaderQos with the default value for each policy.

2. __init__(self: rti.connextdds.DataReaderQos, reader: rti.connextdds.IAnyDataReader) -> None

Create a DataReaderQos with settings equivalent to those of the provided DataReader.

3. __init__(self: rti.connextdds.DataReaderQos, other: rti.connextdds.DataReaderQos) -> None

Create a copy of a DataReaderQos object.

__lshift__(**args*, ***kwargs*)

Overloaded function.

1. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.Durability) -> rti.connextdds.DataReaderQos

Set the DurabilityQoS.

2. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.Deadline) -> rti.connextdds.DataReaderQos

Set the DeadlineQoS.

3. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.LatencyBudget) -> rti.connextdds.DataReaderQos

Set the LatencyBudgetQoS.

4. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.Liveliness) -> rti.connextdds.DataReaderQos

Set the LivelinessQoS.

5. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.Reliability) -> rti.connextdds.DataReaderQos

Set the ReliabilityQoS.

6. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.DestinationOrder) -> rti.connextdds.DataReaderQos

Set the DestinationOrderQoS.

7. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.History) -> rti.connextdds.DataReaderQos

Set the HistoryQoS.

8. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.ResourceLimits) -> rti.connextdds.DataReaderQos

Set the ResourceLimitsQoS.

9. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.UserData) -> rti.connextdds.DataReaderQos

Set the UserDataQoS.

10. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.Ownership) -> rti.connextdds.DataReaderQos

Set the OwnershipQoS.

11. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.TimeBasedFilter) -> rti.connextdds.DataReaderQos

Set the TimeBasedFilterQoS.

12. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.ReaderDataLifecycle) -> rti.connextdds.DataReaderQos

Set the ReaderDataLifecycleQoS.

13. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.TransportPriority) -> rti.connextdds.DataReaderQos

Set the TransportPriorityQoS.

14. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.TypeConsistencyEnforcement) -> rti.connextdds.DataReaderQos

Set the TypeConsistencyEnforcementQoS.

15. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.DataReaderResourceLimits) -> rti.connextdds.DataReaderQos

Set the DataReaderResourceLimitsQoS.

16. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.DataReaderProtocol) -> rti.connextdds.DataReaderQos

Set the DataReaderProtocolQoS.

17. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.TransportSelection) -> rti.connextdds.DataReaderQos

Set the TransportSelectionQoS.

18. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.TransportUnicast) -> rti.connextdds.DataReaderQos

Set the TransportUnicastQoS.

19. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.TransportMulticast) -> rti.connextdds.DataReaderQos

Set the TransportMulticastQoS.

20. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.Property) -> rti.connextdds.DataReaderQos

Set the PropertyQoS.

21. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.Service) -> rti.connextdds.DataReaderQos

Set the ServiceQoS.

22. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.Availability) -> rti.connextdds.DataReaderQos

Set the AvailabilityQoS.

23. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.EntityName) -> rti.connextdds.DataReaderQos

Set the EntityNameQoS.

24. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.TypeSupport) -> rti.connextdds.DataReaderQos

Set the TypeSupportQoS.

25. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.DataRepresentation) -> rti.connextdds.DataReaderQos

Set the DataRepresentationQoS.

26. __lshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.DataTag) -> rti.connextdds.DataReaderQos

Set the DataTagQoS.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DataReaderQos, *arg0:* rti.connextdds.DataReaderQos) → bool

Test for inequality.

__rshift__(**args*, ***kwargs*)

Overloaded function.

1. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.Durability) -> rti.connextdds.DataReaderQos

Get the DurabilityQoS.

2. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.Deadline) -> rti.connextdds.DataReaderQos

Get the DeadlineQoS.

3. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.LatencyBudget) -> rti.connextdds.DataReaderQos

Get the LatencyBudgetQoS.

4. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.Liveliness) -> rti.connextdds.DataReaderQos

Get the LivelinessQoS.

5. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.Reliability) -> rti.connextdds.DataReaderQos

Get the ReliabilityQoS.

6. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.DestinationOrder) -> rti.connextdds.DataReaderQos

Get the DestinationOrderQoS.

7. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.History) -> rti.connextdds.DataReaderQos

Get the HistoryQoS.

8. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.ResourceLimits) -> rti.connextdds.DataReaderQos

Get the ResourceLimitsQoS.

9. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.UserData) -> rti.connextdds.DataReaderQos

Get the UserDataQoS.

10. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.Ownership) -> rti.connextdds.DataReaderQos

Get the OwnershipQoS.

11. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.TimeBasedFilter) -> rti.connextdds.DataReaderQos

Get the TimeBasedFilterQoS.

12. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.ReaderDataLifecycle) -> rti.connextdds.DataReaderQos

Get the ReaderDataLifecycleQoS.

13. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.TransportPriority) -> rti.connextdds.DataReaderQos

Get the TransportPriorityQoS.

14. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.TypeConsistencyEnforcement) -> rti.connextdds.DataReaderQos

Get the TypeConsistencyEnforcementQoS.

15. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.DataReaderResourceLimits) -> rti.connextdds.DataReaderQos

Get the DataReaderResourceLimitsQoS.

16. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.DataReaderProtocol) -> rti.connextdds.DataReaderQos

Get the DataReaderProtocolQoS.

17. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.TransportSelection) -> rti.connextdds.DataReaderQos

Get the TransportSelectionQoS.

18. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.TransportUnicast) -> rti.connextdds.DataReaderQos

Get the TransportUnicastQoS.

19. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.TransportMulticast) -> rti.connextdds.DataReaderQos

Get the TransportMulticastQoS.

20. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.Property) -> rti.connextdds.DataReaderQos

Get the PropertyQoS.

21. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.Service) -> rti.connextdds.DataReaderQos

Get the ServiceQoS.

22. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.Availability) -> rti.connextdds.DataReaderQos

Get the AvailabilityQoS.

23. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.EntityName) -> rti.connextdds.DataReaderQos

Get the EntityNameQoS.

24. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.TypeSupport) -> rti.connextdds.DataReaderQos

Get the TypeSupportQoS.

25. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.DataRepresentation) -> rti.connextdds.DataReaderQos

Get the DataRepresentationQoS.

26. __rshift__(self: rti.connextdds.DataReaderQos, arg0: rti.connextdds.DataTag) -> rti.connextdds.DataReaderQos

Get the DataTagQoS.

__str__ (*self:* rti.connextdds.DataReaderQos) → str

property availability

Get/set Availability QoS.

property data_reader_protocol

(Deprecated) use protocol instead

property data_reader_resource_limits

(Deprecated) use reader_resource_limits instead

property data_representation

Get/set DataRepresentation QoS.

property data_tag

Get/set DataTag QoS.

property deadline

Get/set Deadline QoS.

property destination_order

Get/set DestinationOrder QoS.

property durability

Get/set Durability QoS.

property entity_name

Get/set EntityName QoS.

property history

Get/set History QoS.

property latency_budget

Get/set LatencyBudget QoS.

property liveliness

Get/set Liveliness QoS.

property ownership

Get/set Ownership QoS.

property property

Get/set Property QoS.

property protocol

Get/set DataReaderProtocol QoS.

property reader_data_lifecycle

Get/set ReaderDataLifecycle QoS.

property reader_resource_limits

Get/set DataReaderResourceLimits QoS.

property reliability

Get/set Reliability QoS.

property resource_limits

Get/set ResourceLimits QoS.

property service

Get/set Service QoS.

property time_based_filter

Get/set TimeBasedFilter QoS.

to_string(*self:* rti.connextdds.DataReaderQos, *format:* rti.connextdds.QosPrintFormat = *QosPrintFormat()*, *base: Optional[*rti.connextdds.DataReaderQos*]* = *None*, *print_all: bool* = *False*) → str

Convert QoS to string based on params.

property transport_multicast

Get/set TransportMulticast QoS.

property transport_priority

Get/set TransportPriority QoS.

property transport_selection

Get/set TransportSelection QoS.

property transport_unicast

Get/set TransportUnicast QoS.

property type_consistency

Get/set TypeConsistencyEnforcement QoS.

property type_consistency_enforcement

(Deprecated) use type_consistency instead

property type_support

Get/set TypeSupport QoS.

property user_data

Get/set UserData QoS.

class rti.connextdds.**DataReaderResourceLimits**

Bases: pybind11_object

AUTO_MAX_TOTAL_INSTANCES = 0

__eq__(*self:* rti.connextdds.DataReaderResourceLimits, *arg0:* rti.connextdds.DataReaderResourceLimits) → bool

__hash__ = None

__init__(*self:* rti.connextdds.DataReaderResourceLimits) → None

Create a default DataReaderResourceLimits policy with default settings.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DataReaderResourceLimits, *arg0:* rti.connextdds.DataReaderResourceLimits) → bool

property autopurge_remote_not_alive_writer_delay

Maximum duration for which the DataReader will maintain information regarding a DataWriter once the DataWriter has become not alive.

This property's getter returns a deep copy.

property autopurge_remote_virtual_writer_delay

Maximum duration for which the DataReader will maintain information regarding a virtual DataWriter once it has become not alive.

This property's getter returns a deep copy.

property disable_fragmentation_support

Determines whether the DataReader can receive fragmented samples.

property dynamically_allocate_fragmented_samples

Determines whether the DataReader pre-allocates storage for storing fragmented samples.

property initial_fragmented_samples

The initial number of samples for which a DataReader may store fragments.

property initial_infos

The initial number of info units that a DataReader can use to store SampleInfo.

property initial_outstanding_reads

The initial number of outstanding calls to read/take (or one of their variants) on the same DataReader for which memory has not been returned by calling LoanedSamples.return_loan().

property initial_remote_virtual_writers

The initial number of remote virtual writers from which a DataReader may read, including all instances.

property initial_remote_virtual_writers_per_instance

The initial number of virtual remote writers that can be associated with an instance.

property initial_remote_writers

The initial number of remote writers from which a DataReader may read, including all instances.

property initial_remote_writers_per_instance

The initial number of remote writers from which a DataReader may read a single instance.

property initial_topic_queries

The initial number of TopicQueries allocated by a DataReader.

property instance_replacement

The instance replacement policy.

property keep_minimum_state_for_instances

Whether or not keep a minimum instance state for up to DataReaderResourceLimits.max_total_instances.

property max_app_ack_response_length

Maximum length of application-level acknowledgment response data.

property max_fragmented_samples

The maximum number of samples for which the DataReader may store fragments at a given point in time.

property max_fragmented_samples_per_remote_writer

The maximum number of samples per remote writer for which a DataReader may store fragments.

property max_fragments_per_sample

Maximum number of fragments for a single sample.

property max_infos

The maximum number of info units that a DataReader can use to store SampleInfo.

property max_outstanding_reads

The max number of outstanding calls to read/take (or one of their variants) on the same DataReader for which memory has not been returned by calling LoanedSamples.return_loan().

property max_query_condition_filters

The maximum number of query condition filters a reader is allowed.

property max_remote_virtual_writers

The maximum number of remote virtual writers from which a DataReader may read, including all instances.

property max_remote_virtual_writers_per_instance

The maximum number of virtual remote writers that can be associated with an instance.

property max_remote_writers

The maximum number of remote writers from which a DataReader may read, including all instances.

property max_remote_writers_per_instance

The maximum number of remote writers from which a DataReadermay read a single instance.

property max_remote_writers_per_sample

The maximum number of remote writers allowed to write the same sample.

property max_samples_per_read

The maximum number of data samples that the application can receive from the middleware in a single call to DataReader.read() or DataReader.take(). If more data exists in the middleware, the application will need to issue multiple read/take calls.

property max_samples_per_remote_writer

The maximum number of out-of-order samples from a given remote DataWriter that a DataReader may store when maintaining a reliable connection to the DataWriter.

property max_topic_queries

The maximum number of TopicQueries allocated by a DataReader.

property max_total_instances

Maximum number of instances for which a DataReader will keep state.

property shmem_ref_transfer_mode_attached_segment_allocation

The initial number of TopicQueries allocated by a DataReader.

class
rti.connextdds.**DataReaderResourceLimitsInstanceReplacementSettings**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.DataReaderResourceLimitsInstanceReplacementSettings, *arg0:* rti.connextdds.DataReaderResourceLimitsInstanceReplacementSettings) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DataReaderResourceLimitsInstanceReplacementSettings) -> None

Creates an instance with the default removal kind for each instance state.

2. __init__(self: rti.connextdds.DataReaderResourceLimitsInstanceReplacementSettings, alive_instance_removal: rti.connextdds.DataReaderInstanceRemovalKind, disposed_instance_removal: rti.connextdds.DataReaderInstanceRemovalKind, no_writers_instance_removal: rti.connextdds.DataReaderInstanceRemovalKind) -> None

Creates an instance with the given DataReaderInstanceRemovalKind for alive, disposed and no_writers instances, respectively.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DataReaderResourceLimitsInstanceReplacementSettings, *arg0:* rti.connextdds.DataReaderResourceLimitsInstanceReplacementSettings) → bool

Test for inequality.

property alive_instance_removal

The instance replacment policy for alive instances.

property disposed_instance_removal

The instance replacment policy for disposed instances.

property no_writers_instance_removal

The instance replacment policy for not-alive-no-writer instances.

class rti.connextdds.**DataReaderSeq**

Bases: pybind11_object

__add__(*self: List[*rti.connextdds.DataReader*]*, *arg0: List[*rti.connextdds.DataReader*]*) → List[*rti.connextdds.DataReader*]

__bool__(*self: List[*rti.connextdds.DataReader*]*) → bool

Check whether the list is nonempty

__contains__(*self: List[*rti.connextdds.DataReader*]*, *x:* rti.connextdds.DataReader) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: List[rti.connextdds.DataReader], arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: List[rti.connextdds.DataReader], arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self: List[*rti.connextdds.DataReader*]*, *arg0: List[*rti.connextdds.DataReader*]*) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: List[rti.connextdds.DataReader], s: slice) -> List[rti.connextdds.DataReader]

Retrieve list elements using a slice object

2. __getitem__(self: List[rti.connextdds.DataReader], arg0: int) -> rti.connextdds.DataReader

__hash__ = None

__iadd__ (*self: List[*rti.connextdds.DataReader*]*, *arg0: List[*rti.connextdds.DataReader*]*) → List[*rti.connextdds.DataReader*]

__imul__ (*self: List[*rti.connextdds.DataReader*]*, *arg0: int*) → List[*rti.connextdds.DataReader*]

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DataReaderSeq) -> None

2. __init__(self: rti.connextdds.DataReaderSeq, arg0: List[rti.connextdds.DataReader]) -> None

Copy constructor

3. __init__(self: rti.connextdds.DataReaderSeq, arg0: Iterable) -> None

__iter__ (*self: List[*rti.connextdds.DataReader*]*) → Iterator

__len__ (*self: List[*rti.connextdds.DataReader*]*) → int

__module__ = 'rti.connextdds'

__mul__ (*self: List[*rti.connextdds.DataReader*]*, *arg0: int*) → List[*rti.connextdds.DataReader*]

__ne__ (*self: List[*rti.connextdds.DataReader*]*, *arg0: List[*rti.connextdds.DataReader*]*) → bool

__rmul__ (*self: List[*rti.connextdds.DataReader*]*, *arg0: int*) → List[*rti.connextdds.DataReader*]

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: List[rti.connextdds.DataReader], arg0: int, arg1: rti.connextdds.DataReader) -> None

2. __setitem__(self: List[rti.connextdds.DataReader], arg0: slice, arg1: List[rti.connextdds.DataReader]) -> None

Assign list elements using a slice object

append (*self: List[*rti.connextdds.DataReader*]*, *x:* rti.connextdds.DataReader) → None

Add an item to the end of the list

clear (*self: List[*rti.connextdds.DataReader*]*) → None

Clear the contents

count (*self: List[*rti.connextdds.DataReader*]*, *x:* rti.connextdds.DataReader) → int

Return the number of times x appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: List[rti.connextdds.DataReader], L: List[rti.connextdds.DataReader]) -> None

Extend the list by appending all the items in the given list

2. extend(self: List[rti.connextdds.DataReader], L: Iterable) -> None

Extend the list by appending all the items in the given list

insert(*self: List[*rti.connextdds.DataReader*]*, *i: int*, *x:* rti.connextdds.DataReader) → None

Insert an item at a given position.

pop(**args*, ***kwargs*)

Overloaded function.

1. pop(self: List[rti.connextdds.DataReader]) -> rti.connextdds.DataReader

Remove and return the last item

2. pop(self: List[rti.connextdds.DataReader], i: int) -> rti.connextdds.DataReader

Remove and return the item at index `i`

remove(*self: List[*rti.connextdds.DataReader*]*, *x:* rti.connextdds.DataReader) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**DataRepresentation**

Bases: `pybind11_object`

AUTO_ID = -1

XCDR = 0

XCDR2 = 2

XML = 1

__eq__(*self:* rti.connextdds.DataRepresentation, *arg0:* rti.connextdds.DataRepresentation) → bool

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DataRepresentation) -> None

Creates an instance with one element, AUTO_ID.

2. __init__(self: rti.connextdds.DataRepresentation, representations: rti.connextdds.Int16Seq) -> None

Creates an instance with a sequence of Data Representation IDs.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DataRepresentation, *arg0:* rti.connextdds.DataRepresentation) → bool

property compression_settings

The compression settings.

property value

A sequence of IDs representing the allowed data representations.

This property uses copy semantics. Changes to the sequence are not reflected in the policy unless the sequence is commited back to the policy object via the property's setter.

class rti.connextdds.**DataState**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.DataState, *arg0:* rti.connextdds.DataState) → bool

Compare DataState objects for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DataState) -> None

Create a DataState with InstanceState.any(), ViewState.any(), and SampleState.ANY

2. __init__(self: rti.connextdds.DataState, sample_state: rti.connextdds.SampleState) -> None

Create a DataState with InstanceState.any(), ViewState.any(), and the provided SampleState.

3. __init__(self: rti.connextdds.DataState, view_state: rti.connextdds.ViewState) -> None

Create a DataState with InstanceState.any(), SampleState.any(), and the provided ViewState.

4. __init__(self: rti.connextdds.DataState, instance_state: rti.connextdds.InstanceState) -> None

Create a DataState with ViewState.any(), SampleState.any(), and the provided InstanceState.

5. __init__(self: rti.connextdds.DataState, sample_state: rti.connextdds.SampleState, view_state: rti.connextdds.ViewState, instance_state: rti.connextdds.InstanceState) -> None

Create a DataState with the provided SampleState, ViewState, and InstanceState.

__lshift__ (**args*, ***kwargs*)

Overloaded function.

1. __lshift__(self: rti.connextdds.DataState, arg0: rti.connextdds.SampleState) -> rti.connextdds.DataState

Set the SampleState for the DataState.

2. __lshift__(self: rti.connextdds.DataState, arg0: rti.connextdds.ViewState) -> rti.connextdds.DataState

Set the ViewState for the DataState.

3. __lshift__(self: rti.connextdds.DataState, arg0: rti.connextdds.InstanceState) -> rti.connextdds.DataState

Set the InstanceState for the DataState.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DataState, *arg0:* rti.connextdds.DataState) → bool

Compare DataState objects for inequality.

__repr__(*self:* rti.connextdds.DataState) → str

any = [sample_state = any, view_state = any, instance_state = any]

any_data = [sample_state = any, view_state = any, instance_state = alive]

property instance_state

The InstanceState of the DataState.

new_data = [sample_state = not_read, view_state = any, instance_state = alive]

new_instance = [sample_state = any, view_state = new_view, instance_state = alive]

property sample_state

The SampleState of the DataState.

property view_state

The ViewState of the DataState.

class rti.connextdds.**DataStateEx**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.DataStateEx, *arg0:* rti.connextdds.DataStateEx) → bool

Compare DataStateEx objects for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DataStateEx) -> None

Create a DataStateEx with InstanceState.ANY, ViewState.ANY, SampleState.ANY, and StreamKind.ANY.

2. __init__(self: rti.connextdds.DataStateEx, stream_kind: rti.connextdds.StreamKind) -> None

Create a DataStateEx with InstanceState.ANY, ViewState.ANY, SampleState.ANY and the provided StreamKind.

3. __init__(self: rti.connextdds.DataStateEx, data_state: rti.connextdds.DataState) -> None

Create a DataStateEx with the provided SampleState, ViewState, InstanceState, and StreamKind.ANY

4. __init__(self: rti.connextdds.DataStateEx, data_state: rti.connextdds.DataState, stream_kind: rti.connextdds.StreamKind) -> None

Create a DataStateEx with the provided SampleState, ViewState, InstanceState, and StreamKind.

__lshift__ (**args*, ***kwargs*)

Overloaded function.

1. __lshift__(self: rti.connextdds.DataStateEx, arg0: rti.connextdds.StreamKind) -> rti.connextdds.DataStateEx

Set the StreamKind for the DataStateEx.

2. __lshift__(self: rti.connextdds.DataStateEx, arg0: rti.connextdds.SampleState) -> rti.connextdds.DataState

Set the SampleState for the DataState.

3. __lshift__(self: rti.connextdds.DataStateEx, arg0: rti.connextdds.ViewState) -> rti.connextdds.DataState

Set the ViewState for the DataState.

4. __lshift__(self: rti.connextdds.DataStateEx, arg0: rti.connextdds.InstanceState) -> rti.connextdds.DataState

Set the InstanceState for the DataState.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DataStateEx, *arg0:* rti.connextdds.DataStateEx) → bool

Compare DataStateEx objects for inequality.

any = [sample_state = any, view_state = any, instance_state = any]

any_data = [sample_state = any, view_state = any, instance_state = alive]

property data_state

Access the view, sample and instance states.

property instance_state

The InstanceState of the DataStateEx.

new_data = [sample_state = not_read, view_state = any, instance_state = alive]

new_instance = [sample_state = any, view_state = new_view, instance_state = alive]

property sample_state

The SampleState of the DataStateEx.

property stream_kind

The StreamKind.

property view_state

The ViewState of the DataStateEx.

class rti.connextdds.**DataTag**

Bases: pybind11_object

__contains__ (*self:* rti.connextdds.DataTag, *arg0: str*) → bool

__eq__ (*self:* rti.connextdds.DataTag, *arg0:* rti.connextdds.DataTag) → bool

Test for equality.

__getitem__ (*self:* rti.connextdds.DataTag, *arg0: str*) → Optional[str]

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DataTag) -> None

Creates a policy with an empty sequence of tags.

2. __init__(self: rti.connextdds.DataTag, entries: rti.connextdds.StringPairSeq) -> None

Adds tags from a list.

3. __init__(self: rti.connextdds.DataTag, entries: dict) -> None

Adds tags from a dictionary.

__len__ (*self:* rti.connextdds.DataTag) → int

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DataTag, *arg0:* rti.connextdds.DataTag) → bool

Test for inequality.

__setitem__ (*self:* rti.connextdds.DataTag, *arg0: str*, *arg1: str*) → None

exists (*self:* rti.connextdds.DataTag, *key: str*) → bool

Returns true if a tag exists.

get (*self:* rti.connextdds.DataTag, *key: str*) → str

Returns the value of a tag identified by a key if it exists.

get_all (*self:* rti.connextdds.DataTag) → *rti.connextdds.StringMap*

Retrieves a copy of all the entries.

remove (*self:* rti.connextdds.DataTag, *key: str*) → bool

Removes the tag identified by a key.

set (**args*, ***kwargs*)

Overloaded function.

1. set(self: rti.connextdds.DataTag, entry_list: rti.connextdds.StringPairSeq) -> None

Adds or assigns tags from a list of string pairs.

2. set(self: rti.connextdds.DataTag, entry_map: dict) -> None

Adds or assigns tags from a dictionary.

3. set(self: rti.connextdds.DataTag, entry: Tuple[str, str]) -> None

Adds or assigns a tag from a pair of strings.

4. set(self: rti.connextdds.DataTag, key: str, value: str) -> None

Adds or assigns a tag from a key string and a value string.

size (*self:* rti.connextdds.DataTag) → int

Returns the number of tags.

try_get (*self:* rti.connextdds.DataTag, *key: str*) → Optional[str]

Returns the value of a tag identified by a key if it exists.

class rti.connextdds.**DataWriter**

Bases: *IEntity*, *IAnyDataWriter*

__enter__ (*self:* rti.connextdds.DataWriter) → *rti.connextdds.DataWriter*

Enter a context for this DataWriter, to be cleaned up on exiting context

__eq__ (*self:* rti.connextdds.DataWriter, *arg0:* rti.connextdds.DataWriter) → bool

Test for equality.

__exit__ (*self:* rti.connextdds.DataWriter, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for this DataWriter, cleaning up resources.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DataWriter, topic: rti.connextdds.Topic) -> None

Creates a DataWriter in the implicit publisher with default QoS.

2. __init__(self: rti.connextdds.DataWriter, topic: rti.connextdds.Topic, qos: rti.connextdds.DataWriterQos, listener: rti.connextdds.DataWriterListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Creates a DataWriter in the implicit publisher with specific QoS and optionally a listener.

3. __init__(self: rti.connextdds.DataWriter, pub: rti.connextdds.Publisher, topic: rti.connextdds.Topic) -> None

Creates a DataWriter in a publisher with default QoS.

4. __init__(self: rti.connextdds.DataWriter, pub: rti.connextdds.Publisher, topic: rti.connextdds.Topic, qos: rti.connextdds.DataWriterQos, listener: rti.connextdds.DataWriterListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Creates a DataWriter in a publisher with specific QoS and optionally a listener.

5. __init__(self: rti.connextdds.DataWriter, writer: rti.connextdds.IAnyDataWriter) -> None

Create a typed DataWriter from an AnyDataWriter.

6. __init__(self: rti.connextdds.DataWriter, entity: rti.connextdds.IEntity) -> None

Create a typed DataWriter from an Entity.

__lshift__ (**args*, ***kwargs*)

Overloaded function.

1. __lshift__(self: rti.connextdds.DataWriter, arg0: rti.connextdds.DataWriterQos) -> rti.connextdds.DataWriter

Sets the DataWriterQos.

2. __lshift__(self: rti.connextdds.DataWriter, arg0: Tuple[object, rti.connextdds.Time]) -> rti.connextdds.DataWriter

Writes a paired sample with a timestamp.

3. __lshift__(self: rti.connextdds.DataWriter, arg0: Tuple[object, rti.connextdds.InstanceHandle]) -> rti.connextdds.DataWriter

Writes a paired sample with an instance handle.

4. __lshift__(self: rti.connextdds.DataWriter, arg0: List[Tuple[object, rti.connextdds.Time]]) -> rti.connextdds.DataWriter

Writes a sequence of pairs of samples with timestamps.

5. __lshift__(self: rti.connextdds.DataWriter, arg0: List[object]) -> rti.connextdds.DataWriter

Writes a sequence of samples.

6. __lshift__(self: rti.connextdds.DataWriter, arg0: object) -> rti.connextdds.DataWriter

Writes a sample.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DataWriter, *arg0:* rti.connextdds.DataWriter) → bool

Test for inequality.

__rshift__ (*self:* rti.connextdds.DataWriter, *arg0:* rti.connextdds.DataWriterQos) → *rti.connextdds.DataWriter*

Get the DataWriterQos.

assert_liveliness (*self:* rti.connextdds.DataWriter) → None

Manually asserts the liveliness of the DataWriter.

close (*self:* rti.connextdds.DataWriter) → None

Close this DataWriter.

property datawriter_cache_status

Get a copy of the cache status for this writer.

property datawriter_protocol_status

Get a copy of the protocol status for this writer.

dispose_instance (**args*, ***kwargs*)

Overloaded function.

1. dispose_instance(self: rti.connextdds.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.DataWriter

Dispose an instance.

2. dispose_instance(self: rti.connextdds.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> rti.connextdds.DataWriter

Dispose an instance with a timestamp.

3. dispose_instance(self: rti.connextdds.DataWriter, params: rti.connextdds.WriteParams) -> None

Dispose an instance with params.

static find_all_by_topic (*publisher:* rti.connextdds.Publisher, *topic_name: str*) → List[*rti.connextdds.DataWriter*]

Retrieve all DataWriters for the given topic name in the publisher.

static find_by_name (**args*, ***kwargs*)

Overloaded function.

1. find_by_name(participant: rti.connextdds.DomainParticipant, name: str) -> Optional[rti.connextdds.DataWriter]

Find DataWriter in DomainParticipant with the provided name, returning the first found.

2. find_by_name(publisher: rti.connextdds.Publisher, name: str) -> Optional[rti.connextdds.DataWriter]

Find DataWriter in Publisher with the DataReader's name, returning the first found.

static find_by_topic (*publisher:* rti.connextdds.Publisher, *name: str*) → Optional[*rti.connextdds.DataWriter*]

Find DataWriter in publisher with a topic name, returning the first found.

flush (*self:* rti.connextdds.DataWriter) → None

Flushes the batch in progress in the context of thecalling thread.

is_matched_subscription_active (*self:* rti.connextdds.DataWriter, *arg0:* rti.connextdds.InstanceHandle) → bool

A boolean indicating whether or not the matched subscription is active.

is_sample_app_acknowledged(*self:* rti.connextdds.DataWriter, *sample_id:* rti.connextdds.SampleIdentity) → bool

Indicates if a sample is considered application-acknowledged.

key_value(*self:* rti.connextdds.DataWriter, *handle:* rti.connextdds.InstanceHandle) → object

Retrieve the instance key that corresponds to an instance handle.

property listener

Get the listener associated with the DataWriter or set the listener.

property liveliness_lost_status

Get a copy of the LivelinessLostStatus.

lookup_instance(*self:* rti.connextdds.DataWriter, *key_holder: object*) → *rti.connextdds.InstanceHandle*

Retrieve the instance handle that corresponds to an instance key_holder

matched_subscription_data(*self:* rti.connextdds.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.SubscriptionBuiltinTopicData*

Get the SubscriptionBuiltinTopicData for a subscription matched to this DataWriter.

matched_subscription_datawriter_protocol_status(**args*, ***kwargs*)

Overloaded function.

1. matched_subscription_datawriter_protocol_status(self: rti.connextdds.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.DataWriterProtocolStatus

Get a copy of the protocol status for this writer per a matched subscription handle.

2. matched_subscription_datawriter_protocol_status(self: rti.connextdds.DataWriter, locator: rti.connextdds.Locator) -> rti.connextdds.DataWriterProtocolStatus

Get a copy of the protocol status for this writer per a matched subscription locator.

matched_subscription_participant_data(*self:* rti.connextdds.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ParticipantBuiltinTopicData*

Get the ParticipantBuiltinTopicData for a subscription matched to this DataWriter.

property matched_subscriptions

Get a copy of the list of the currently matched subscription handles.

property matched_subscriptions_locators

The locators used to communicate with matched DataReaders.

property offered_deadline_missed_status

Get a copy of the OfferedDeadlineMissedStatus.

property offered_incompatible_qos_status

Get a copy of the OfferedIncompatibleQosStatus

property publication_matched_status

Get a copy of the PublicationMatchedStatus

property publisher

Get the Publisher that owns this DataWriter.

property qos

The DataWriterQos for this DataWriter.This property's getter returns a deep copy.

register_instance(**args*, ***kwargs*)

Overloaded function.

1. register_instance(self: rti.connextdds.DataWriter, key_holder: object) -> rti.connextdds.InstanceHandle

Informs RTI Connext that the application will be modifying a particular instance.

2. register_instance(self: rti.connextdds.DataWriter, key_holder: object, timestamp: rti.connextdds.Time) -> rti.connextdds.InstanceHandle

Informs RTI Connext that the application will be modifying a particular instance and specified the timestamp.

3. register_instance(self: rti.connextdds.DataWriter, key_holder: object, params: rti.connextdds.WriteParams) -> rti.connextdds.InstanceHandle

Registers instance with parameters.

property reliable_reader_activity_changed_status

Get a copy of the reliable reader activity changed status for this writer.

property reliable_writer_cache_changed_status

Get a copy of the reliable cache status for this writer.

property service_request_accepted_status

Get a copy of the service request accepted status for this writer.

set_listener(*self:* rti.connextdds.DataWriter, *listener:* rti.connextdds.DataWriterListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and event mask for the DataWriter.

property topic

Get the Topic object associated with this DataWriter.

property topic_name

Get the topic name associated with this DataWriter.

property type_name

Get the type name for the topic object associated with this DataWriter.

unregister_instance(**args*, ***kwargs*)

Overloaded function.

1. unregister_instance(self: rti.connextdds.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.DataWriter

Unregister an instance.

2. unregister_instance(self: rti.connextdds.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> rti.connextdds.DataWriter

Unregister an instance with timestamp.

3. unregister_instance(self: rti.connextdds.DataWriter, params: rti.connextdds.WriteParams) -> None

Unregister an instance with parameters.

wait_for_acknowledgments(*self:* rti.connextdds.DataWriter, *max_wait:* rti.connextdds.Duration) → None

Blocks the calling thread until all data written by a reliable DataWriter is acknowledged or until the timeout expires.

wait_for_asynchronous_publishing(*self:* rti.connextdds.DataWriter, *max_wait:* rti.connextdds.Duration) → None

This operation blocks the calling thread (up to max_wait) until all data written by the asynchronous DataWriter is sent and acknowledged (if reliable) by all matched DataReader entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; a time out indicates that max_wait elapsed before all the data was sent and/or acknowledged.

In other words, this guarantees that sending to best effort DataReader is complete in addition to what DataWriter.wait_for_acknowledgments() provides.

If the DataWriter does not have PublishMode kind set to PublishModeKind.ASYNCHRONOUS the operation will complete immediately

wait_for_asynchronous_publishing_async(*self:* rti.connextdds.DataWriter, *max_wait:* rti.connextdds.Duration) → object

This function is awaitable until either a timeout of max_wait or all data written by the asynchronous DataWriter is sent and acknowledged (if reliable) by all matched DataReader entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; a time out indicates that max_wait elapsed before all the data was sent and/or acknowledged.This function works with asyncio.

In other words, this guarantees that sending to best effort DataReader is complete in addition to what DataWriter.wait_for_acknowledgments() provides.

If the DataWriter does not have PublishMode kind set to PublishModeKind.ASYNCHRONOUS the operation will complete immediately

write(**args*, ***kwargs*)

Overloaded function.

1. write(self: rti.connextdds.DataWriter, samples: List[object]) -> None

Write a sequence of samples.

2. write(self: rti.connextdds.DataWriter, samples: List[object], timestamp: rti.connextdds.Time) -> None

Write a sequence of samples with a timestamp.

3. write(self: rti.connextdds.DataWriter, sample: object) -> None

Write a sample.

4. write(self: rti.connextdds.DataWriter, sample: object, timestamp: rti.connextdds.Time) -> None

Write a sample with a specified timestamp.

5. write(self: rti.connextdds.DataWriter, sample: object, handle: rti.connextdds.InstanceHandle) -> None

Write a sample with an instance handle.

6. write(self: rti.connextdds.DataWriter, sample: object, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> None

Write a sample with an instance handle and specified timestamp.

7. write(self: rti.connextdds.DataWriter, sample: object, params: rti.connextdds.WriteParams) -> None

Write with advanced parameters.

class rti.connextdds.**DataWriterCacheStatus**

Bases: pybind11_object

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property alive_instance_count

Number of alive instances in the DataWriter's queue.

property alive_instance_count_peak

Highest number of alive instances in the writer's queue over the lifetime of the writer.

property disposed_instance_count

Number of disposed instances in the DataWriter's queue.

property disposed_instance_count_peak

Highest number of disposed instances in the writer's queue over the lifetime of the writer.

property sample_count

Number of samples in the DataWriter's queue, including unregister and dispose samples.

property sample_count_peak

Highest number of samples in the writer's queue over the lifetime of the writer.

property unregistered_instance_count

Number of unregistered instances in the DataWriter's queue.

property unregistered_instance_count_peak

Highest number of unregistered instances in the writer's queue over the lifetime of the writer.

class rti.connextdds.**DataWriterListener**

Bases: pybind11_object

__init__(*self:* rti.connextdds.DataWriterListener) → None

__module__ = 'rti.connextdds'

on_application_acknowledgment(*self:* rti.connextdds.DataWriterListener, *arg0:* rti.connextdds.DataWriter, *arg1:* rti.connextdds.AcknowledgmentInfo) → None

On application acknowledgment callback

on_instance_replaced(*self:* rti.connextdds.DataWriterListener, *arg0:* rti.connextdds.DataWriter, *arg1:* rti.connextdds.InstanceHandle) → None

On instance replaced callback.

on_liveliness_lost(*self:* rti.connextdds.DataWriterListener, *arg0:* rti.connextdds.DataWriter, *arg1:* rti.connextdds.LivelinessLostStatus) → None

Liveliness lost callback.

on_offered_deadline_missed(*self:* rti.connextdds.DataWriterListener, *arg0:* rti.connextdds.DataWriter, *arg1:* rti.connextdds.OfferedDeadlineMissedStatus) → None

Offered deadline missed callback.

on_offered_incompatible_qos(*self:* rti.connextdds.DataWriterListener, *arg0:* rti.connextdds.DataWriter, *arg1:* rti.connextdds.OfferedIncompatibleQosStatus) → None

Offered incompatible QoS callback.

on_publication_matched(*self:* rti.connextdds.DataWriterListener, *arg0:* rti.connextdds.DataWriter, *arg1:* rti.connextdds.PublicationMatchedStatus) → None

Publication matched callback.

on_reliable_reader_activity_changed(*self:* rti.connextdds.DataWriterListener, *arg0:* rti.connextdds.DataWriter, *arg1:* rti.connextdds.ReliableReaderActivityChanged-Status) → None

Reliable reader activity changed callback.

on_reliable_writer_cache_changed(*self:* rti.connextdds.DataWriterListener, *arg0:* rti.connextdds.DataWriter, *arg1:* rti.connextdds.ReliableWriterCacheChangedStatus) → None

Reliable writer cache changed callback.

on_service_request_accepted(*self:* rti.connextdds.DataWriterListener, *arg0:* rti.connextdds.DataWriter, *arg1:* rti.connextdds.ServiceRequestAcceptedStatus) → None

On service request accepted callback.

class rti.connextdds.**DataWriterProtocol**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.DataWriterProtocol, *arg0:* rti.connextdds.DataWriterProtocol) → bool

__hash__ = None

__init__(*self:* rti.connextdds.DataWriterProtocol) → None

Create a default DataWriterProtocol policy.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DataWriterProtocol, *arg0:* rti.connextdds.DataWriterProtocol) → bool

property disable_inline_keyhash

Controls whether a keyhash is propagated on the wire with each sample.

property disable_positive_acks

Controls whether the DataWriter expects positive acknowledgments from matched DataReaders.

property initial_virtual_sequence_number

The initial virtual sequence number of the DataWriter.

property propagate_app_ack_with_no_response

Controls whether or not a DataWriter receives on_application_acknowledgment notifications with an empty or invalid response.

property push_on_write

Determines whether a sample is pushed on a call to write.

property rtps_object_id

The RTPS object ID of the DataWriter.

property rtps_reliable_writer

RTPS protocol-related configuration settings for the RTPS reliable writer associated to a DataWriter. This parameter only has effect if both the writer and the matching reader are configured with ReliabilityKind.RELIABLE.

property serialize_key_with_dispose

Controls whether or not the serialized key is propagated on the wire with dispose samples.

property virtual_guid

The virtual GUID of the DataWriter. This property uses value semantics.

class rti.connextdds.**DataWriterProtocolStatus**

Bases: pybind11_object

Information about the DataWriter protocol status

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property first_available_sample_sequence_number

The sequence number of the first available sample currently queued in the local DataWriter.

property first_available_sample_virtual_sequence_number

The virtual sequence number of the first available sample currently queued in the local DataWriter.

property first_unacknowledged_sample_sequence_number

The sequence number of the first unacknowledged sample currently queued in the local DataWriter.

property first_unacknowledged_sample_subscription_handle

The handle of a remote DataReader that has not acknowledged the first unacknowledged sample of the local DataWriter.

property first_unacknowledged_sample_virtual_sequence_number

The virtual sequence number of the first unacknowledged sample currently queued in the local DataWriter.

property first_unelapsed_keep_duration_sample_sequence_number

The sequence number of the first sample whose keep duration has not yet elapsed.

property last_available_sample_sequence_number

The sequence number of the last available sample currently queued in the local DataWriter.

property last_available_sample_virtual_sequence_number

The virtual sequence number of the last available sample currently queued in the local DataWriter.

property pulled_fragment_bytes

The number of bytes of DATA_FRAG messages that have been pulled from this DataWriter.

property pulled_fragment_count

The number of DATA_FRAG messages that have been pulled from this DataWriter.

property pulled_sample_bytes

The number of bytes of user samples pulled from local DataWriter by matching DataReaders.

property pulled_sample_count

The number of user samples pulled from local DataWriter by matching DataReaders.

property pushed_fragment_bytes

The number of bytes of DATA_FRAG messages that have been pushed by this DataWriter.

property pushed_fragment_count

The number of DATA_FRAG messages that have been pushed by this DataWriter.

property pushed_sample_bytes

The number of bytes of user samples pushed on write from a local DataWriter to a matching remote DataReader.

property pushed_sample_count

The number of user samples pushed on write from a local DataWriter to a matching remote DataReader.

property received_ack_bytes

The number of bytes of ACKs from a remote DataReader received by a local DataWriter.

property received_ack_count

The number of ACKs from a remote DataReader received by a local DataWriter.

property received_nack_bytes

The number of bytes of NACKs from a remote DataReader received by a local DataWriter.

property received_nack_count

The number of NACKs from a remote DataReader received by a local DataWriter.

property received_nack_fragment_bytes

The number of bytes of NACK_FRAG messages that have been received by this DataWriter.

property received_nack_fragment_count

The number of NACK_FRAG messages that have been received by this DataWriter.

property rejected_sample_count

The number of times a sample is rejected due to exceptions in the send path.

property send_window_size

Current maximum number of outstanding samples allowed in the DataWriter's queue.

property sent_gap_bytes

The number of bytes of GAPs sent from local DataWriter to matching remote DataReaders.

property sent_gap_count

The number of GAPs sent from local DataWriter to matching remote DataReaders.

property sent_heartbeat_bytes

The number of bytes of Heartbeats sent between a local DataWriter and matching remote DataReader.

property sent_heartbeat_count

The number of Heartbeats sent between a local DataWriter and matching remote DataReader.

class rti.connextdds.**DataWriterQos**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.DataWriterQos, *arg0:* rti.connextdds.DataWriterQos) → bool

Test for equality

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DataWriterQos) -> None

Create a DataWriterQos with the default value for each policy.

2. __init__(self: rti.connextdds.DataWriterQos, writer: rti.connextdds.IAnyDataWriter) -> None

Create a DataWriterQos with settings equivalent to those of the provided DataWriter.

3. __init__(self: rti.connextdds.DataWriterQos, other: rti.connextdds.DataWriterQos) -> None

Create a copy of a DataWriterQos object.

__lshift__(**args*, ***kwargs*)

Overloaded function.

1. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Durability) -> rti.connextdds.DataWriterQos

Set the DurabilityQoS.

2. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.DurabilityService) -> rti.connextdds.DataWriterQos

Set the DurabilityServiceQoS.

3. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Deadline) -> rti.connextdds.DataWriterQos

Set the DeadlineQoS.

4. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.LatencyBudget) -> rti.connextdds.DataWriterQos

Set the LatencyBudgetQoS.

5. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Liveliness) -> rti.connextdds.DataWriterQos

Set the LivelinessQoS.

6. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Reliability) -> rti.connextdds.DataWriterQos

Set the ReliabilityQoS.

7. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.DestinationOrder) -> rti.connextdds.DataWriterQos

Set the DestinationOrderQoS.

8. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.History) -> rti.connextdds.DataWriterQos

Set the HistoryQoS.

9. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.ResourceLimits) -> rti.connextdds.DataWriterQos

Set the ResourceLimitsQoS.

10. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.TransportPriority) -> rti.connextdds.DataWriterQos

Set the TransportPriorityQoS.

11. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Lifespan) -> rti.connextdds.DataWriterQos

Set the LifespanQoS.

12. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.UserData) -> rti.connextdds.DataWriterQos

Set the UserDataQoS.

13. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Ownership) -> rti.connextdds.DataWriterQos

Set the OwnershipQoS.

14. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.OwnershipStrength) -> rti.connextdds.DataWriterQos

Set the OwnershipStrengthQoS.

15. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.WriterDataLifecycle) -> rti.connextdds.DataWriterQos

Set the WriterDataLifecycleQoS.

16. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.DataWriterResourceLimits) -> rti.connextdds.DataWriterQos

Set the DataWriterResourceLimitsQoS.

17. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.DataWriterProtocol) -> rti.connextdds.DataWriterQos

Set the DataWriterProtocolQoS.

18. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.TransportSelection) -> rti.connextdds.DataWriterQos

Set the TransportSelectionQoS.

19. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.TransportUnicast) -> rti.connextdds.DataWriterQos

Set the TransportUnicastQoS.

20. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.PublishMode) -> rti.connextdds.DataWriterQos

Set the PublishModeQoS.

21. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Property) -> rti.connextdds.DataWriterQos

Set the PropertyQoS.

22. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Service) -> rti.connextdds.DataWriterQos

Set the ServiceQoS.

23. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Batch) -> rti.connextdds.DataWriterQos

Set the BatchQoS.

24. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.MultiChannel) -> rti.connextdds.DataWriterQos

Set the MultiChannelQoS.

25. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Availability) -> rti.connextdds.DataWriterQos

Set the AvailabilityQoS.

26. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.EntityName) -> rti.connextdds.DataWriterQos

Set the EntityNameQoS.

27. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.TopicQueryDispatch) -> rti.connextdds.DataWriterQos

Set the TopicQueryDispatchQoS.

28. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.TypeSupport) -> rti.connextdds.DataWriterQos

Set the TypeSupportQoS.

29. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.DataRepresentation) -> rti.connextdds.DataWriterQos

Set the DataRepresentationQoS.

30. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.DataTag) -> rti.connextdds.DataWriterQos

Set the DataTagQoS.

31. __lshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.DataWriterTransferMode) -> rti.connextdds.DataWriterQos

Set the DataWriterTransferModeQoS.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DataWriterQos, *arg0:* rti.connextdds.DataWriterQos) → bool

Test for inequality.

__rshift__ (**args*, ***kwargs*)

Overloaded function.

1. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Durability) -> rti.connextdds.DataWriterQos

Get the DurabilityQoS.

2. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.DurabilityService) -> rti.connextdds.DataWriterQos

Get the DurabilityServiceQoS.

3. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Deadline) -> rti.connextdds.DataWriterQos

Get the DeadlineQoS.

4. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.LatencyBudget) -> rti.connextdds.DataWriterQos

Get the LatencyBudgetQoS.

5. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Liveliness) -> rti.connextdds.DataWriterQos

Get the LivelinessQoS.

6. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Reliability) -> rti.connextdds.DataWriterQos

Get the ReliabilityQoS.

7. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.DestinationOrder) -> rti.connextdds.DataWriterQos

Get the DestinationOrderQoS.

8. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.History) -> rti.connextdds.DataWriterQos

Get the HistoryQoS.

9. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.ResourceLimits) -> rti.connextdds.DataWriterQos

Get the ResourceLimitsQoS.

10. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.TransportPriority) -> rti.connextdds.DataWriterQos

Get the TransportPriorityQoS.

11. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Lifespan) -> rti.connextdds.DataWriterQos

Get the LifespanQoS.

12. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.UserData) -> rti.connextdds.DataWriterQos

Get the UserDataQoS.

13. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Ownership) -> rti.connextdds.DataWriterQos

Get the OwnershipQoS.

14. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.OwnershipStrength) -> rti.connextdds.DataWriterQos

Get the OwnershipStrengthQoS.

15. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.WriterDataLifecycle) -> rti.connextdds.DataWriterQos

Get the WriterDataLifecycleQoS.

16. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.DataWriterResourceLimits) -> rti.connextdds.DataWriterQos

Get the DataWriterResourceLimitsQoS.

17. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.DataWriterProtocol) -> rti.connextdds.DataWriterQos

Get the DataWriterProtocolQoS.

18. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.TransportSelection) -> rti.connextdds.DataWriterQos

Get the TransportSelectionQoS.

19. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.TransportUnicast) -> rti.connextdds.DataWriterQos

Get the TransportUnicastQoS.

20. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.PublishMode) -> rti.connextdds.DataWriterQos

Get the PublishModeQoS.

21. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Property) -> rti.connextdds.DataWriterQos

Get the PropertyQoS.

22. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Service) -> rti.connextdds.DataWriterQos

Get the ServiceQoS.

23. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Batch) -> rti.connextdds.DataWriterQos

Get the BatchQoS.

24. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.MultiChannel) -> rti.connextdds.DataWriterQos

Get the MultiChannelQoS.

25. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.Availability) -> rti.connextdds.DataWriterQos

Get the AvailabilityQoS.

26. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.EntityName) -> rti.connextdds.DataWriterQos

Get the EntityNameQoS.

27. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.TopicQueryDispatch) -> rti.connextdds.DataWriterQos

Get the TopicQueryDispatchQoS.

28. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.TypeSupport) -> rti.connextdds.DataWriterQos

Get the TypeSupportQoS.

29. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.DataRepresentation) -> rti.connextdds.DataWriterQos

Get the DataRepresentationQoS.

30. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.DataTag) -> rti.connextdds.DataWriterQos

Get the DataTagQoS.

31. __rshift__(self: rti.connextdds.DataWriterQos, arg0: rti.connextdds.DataWriterTransferMode) -> rti.connextdds.DataWriterQos

Get the DataWriterTransferModeQoS.

__str__(*self:* rti.connextdds.DataWriterQos) → str

property availability

Get/set Availability QoS.

property batch

Get/set Batch QoS.

property data_representation

Get/set DataRepresentation QoS.

property data_tag

Get/set DataTag QoS.

property data_writer_protocol

(Deprecated) use protocol instead

property data_writer_resource_limits

(Deprecated) use writer_resource_limits instead

property data_writer_transfer_mode

Get/set DataWriterTransferMode QoS.

property deadline

Get/set Deadline QoS.

property destination_order

Get/set DestinationOrder QoS.

property durability

Get/set Durability QoS.

property durability_service

Get/set DurabilityService QoS.

property entity_name

Get/set EntityName QoS.

property history

Get/set History QoS.

property latency_budget

Get/set LatencyBudget QoS.

property lifespan

Get/set Lifespan QoS.

property liveliness

Get/set Liveliness QoS.

property multi_channel

Get/set MultiChannel QoS.

property ownership

Get/set Ownership QoS.

property ownership_strength

Get/set OwnershipStrength QoS.

property property

Get/set Property QoS.

property protocol

Get/set DataWriterProtocol QoS.

property publish_mode

Get/set PublishMode QoS.

property reliability

Get/set Reliability QoS.

property resource_limits

Get/set ResourceLimits QoS.

property service

Get/set Service QoS.

to_string(*self:* rti.connextdds.DataWriterQos, *format:* rti.connextdds.QosPrintFormat = *QosPrintFormat()*, *base: Optional[*rti.connextdds.DataWriterQos*] = None*, *print_all: bool = False*) → str

Convert QoS to string based on params.

property topic_query_dispatch

Get/set TopicQueryDispatch QoS.

property transport_priority

Get/set TransportPriority QoS.

property transport_selection

Get/set TransportSelection QoS.

property transport_unicast

Get/set TransportUnicast QoS.

property type_support

Get/set TypeSupport QoS.

property user_data

Get/set UserData QoS.

property writer_data_lifecycle

Get/set WriterDataLifecycle QoS.

property writer_resource_limits

Get/set DataWriterResourceLimits QoS.

class rti.connextdds.**DataWriterResourceLimits**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.DataWriterResourceLimits, *arg0:* rti.connextdds.DataWriterResourceLimits) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.DataWriterResourceLimits) → None

Create a default DataWriterResourceLimits policy.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DataWriterResourceLimits, *arg0:* rti.connextdds.DataWriterResourceLimits) → bool

Test for inequality.

property autoregister_instances

Whether or not to automatically register new instances.

property cookie_max_length

Represents the maximum length in bytes of a Cookie.

property initial_active_topic_queries

Represents the initial number of active topic queries a DataWriter will manage.

property initial_batches

Represents the initial number of batches a DataWriter will manage.

property initial_concurrent_blocking_threads

The initial number of threads that are allowed to concurrently block on write call on the same DataWriter.

property initial_virtual_writers

The initial number of virtual writers supported by a DataWriter.

property initialize_writer_loaned_sample

Whether or not to initialize loaned samples returned by a DataWriter.

property instance_replacement

Sets the kinds of instances allowed to be replaced when instance resource limits are reached.

property max_active_topic_queries

Represents the maximum number of active topic queries a DataWriter will manage.

property max_app_ack_remote_readers

The maximum number of application-level acknowledging remote readers supported by a DataWriter.

property max_batches

Represents the maximum number of batches a DataWriter will manage.

property max_concurrent_blocking_threads

The maximum number of threads that are allowed to concurrently block on write call on the same DataWriter.

property max_remote_reader_filters

The maximum number of remote DataReaders for which the DataWriter will perform content-based filtering.

property max_remote_readers

The maximum number of remote readers supported by a DataWriter.

property max_virtual_writers

The maximum number of virtual writers supported by a DataWriter.

property replace_empty_instances

Whether or not to replace empty instances during instance replacement.

property writer_loaned_sample_allocation

Represents the allocation settings of loaned samples managed by a DataWriter.

class rti.connextdds.**DataWriterResourceLimitsInstanceReplacementKind**

Bases: pybind11_object

ALIVE = <DataWriterResourceLimitsInstanceReplacementKind.ALIVE: 1>

ALIVE_OR_DISPOSED = <DataWriterResourceLimitsInstanceReplacementKind.ALIVE_OR_DISPOSED: 5>

ALIVE_THEN_DISPOSED = <DataWriterResourceLimitsInstanceReplacementKind.ALIVE_THEN_DISPOSED: 3>

DISPOSED = <DataWriterResourceLimitsInstanceReplacementKind.DISPOSED: 2>

DISPOSED_THEN_ALIVE = <DataWriterResourceLimitsInstanceReplacementKind.DISPOSED_THEN_ALIVE: 4>

class DataWriterResourceLimitsInstanceReplacementKind

Bases: pybind11_object

Members:

UNREGISTERED : Allows a DataWriter to reclaim unregistered acknowledged instances.

By default, all instance replacement kinds first attempt to reclaim an unregistered, acknowledged instance. Used in DataWriterResourceLimits.instance_replacement [default]

ALIVE : Allows a DataWriter to reclaim alive, acknowledged instances.

When an unregistered, acknowledged instance is not available to reclaim, this kind allows a DataWriter to reclaim an alive, acknowledged instance, where an alive instance is a registered, non-disposed instance. The least recently registered or written alive instance will be reclaimed.

DISPOSED : Allows a DataWriter to reclaim disposed acknowledged instances.

When an unregistered, acknowledged instance is not available to reclaim, this kind allows a DataWriter to reclaim a disposed, acknowledged instance. The least recently disposed instance will be reclaimed.

ALIVE_THEN_DISPOSED : Allows a DataWriter first to reclaim an alive, acknowledged instance, and then, if necessary, a disposed, acknowledged instance.

When an unregistered, acknowledged instance is not available to reclaim, this kind allows a DataWriter to first try reclaiming an alive, acknowledged instance. If no instance is reclaimable, then it tries reclaiming a disposed, acknowledged instance. The least recently used (i.e., registered, written, or disposed) instance will be reclaimed.

DISPOSED_THEN_ALIVE : Allows a DataWriter first to reclaim a disposed, acknowledged instance, and then, if necessary, an alive, acknowledged instance.

When an unregistered, acknowledged instance is not available to reclaim, this kind allows a DataWriter to first try reclaiming a disposed, acknowledged instance. If no instance is reclaimable, then it tries reclaiming an alive, acknowledged instance. The least recently used (i.e., disposed, registered, or written) instance will be reclaimed.

ALIVE_OR_DISPOSED : Allows a DataWriter to reclaim a either an alive acknowledged instance or a disposed acknowledged instance.

When an unregistered acknowledged instance is not available to reclaim, this kind allows a DataWriter to reclaim either an alive, acknowledged instance or a disposed, acknowledged instance. If both instance kinds are available to reclaim, the DataWriter will reclaim the least recently used (i.e. disposed, registered, or written) instance.

ALIVE = <DataWriterResourceLimitsInstanceReplacementKind.ALIVE: 1>

ALIVE_OR_DISPOSED = <DataWriterResourceLimitsInstanceReplacementKind. ALIVE_OR_DISPOSED: 5>

ALIVE_THEN_DISPOSED = <DataWriterResourceLimitsInstanceReplacementKind. ALIVE_THEN_DISPOSED: 3>

DISPOSED = <DataWriterResourceLimitsInstanceReplacementKind.DISPOSED: 2>

DISPOSED_THEN_ALIVE = <DataWriterResourceLimitsInstanceReplacementKind. DISPOSED_THEN_ALIVE: 4>

UNREGISTERED = <DataWriterResourceLimitsInstanceReplacementKind.UNREGISTERED: 0>

__eq__(*self: object*, *other: object*) → bool

__getstate__ (*self: object*) → int

__hash__ (*self: object*) → int

__index__ (*self:* rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind.DataWriterResourceLimitsInstanceReplacementKind) → int

__init__ (*self:* rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind.DataWriterResourceLimitsInstanceReplacementKind, *value: int*) → None

__int__ (*self:* rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind.DataWriterResourceLimitsInstanceReplacementKind) → int

__members__ = {'ALIVE': <DataWriterResourceLimitsInstanceReplacementKind.ALIVE: 1>, 'ALIVE_OR_DISPOSED': <DataWriterResourceLimitsInstanceReplacementKind.ALIVE_OR_DISPOSED: 5>, 'ALIVE_THEN_DISPOSED': <DataWriterResourceLimitsInstanceReplacementKind.ALIVE_THEN_DISPOSED: 3>, 'DISPOSED': <DataWriterResourceLimitsInstanceReplacementKind.DISPOSED: 2>, 'DISPOSED_THEN_ALIVE': <DataWriterResourceLimitsInstanceReplacementKind.DISPOSED_THEN_ALIVE: 4>, 'UNREGISTERED': <DataWriterResourceLimitsInstanceReplacementKind.UNREGISTERED: 0>}

__module__ = 'rti.connextdds'

__ne__ (*self: object*, *other: object*) → bool

__repr__ (*self: object*) → str

__setstate__ (*self:* rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind.DataWriterResourceLimitsInstanceReplacementKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

UNREGISTERED = <DataWriterResourceLimitsInstanceReplacementKind.UNREGISTERED: 0>

__eq__ (*self:* rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind, *arg0:* rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind) → bool

Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind, *arg0:* rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind, *arg0:* rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind, arg0: rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind.DataWriterResourceLimitsInstanceReplacementKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind) → *rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind.DataWriterResourceLimitsInstanceReplacementKind*

Int conversion.

__le__ (*self:* rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind, *arg0:* rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind, *arg0:* rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind, *arg0:* rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.DataWriterResourceLimitsInstanceReplacementKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**DataWriterSeq**

Bases: pybind11_object

__add__(*self: List[*rti.connextdds.DataWriter*]*, *arg0: List[*rti.connextdds.DataWriter*]*) → List[*rti.connextdds.DataWriter*]

__bool__(*self: List[*rti.connextdds.DataWriter*]*) → bool

Check whether the list is nonempty

__contains__(*self: List[*rti.connextdds.DataWriter*]*, *x:* rti.connextdds.DataWriter) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: List[rti.connextdds.DataWriter], arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: List[rti.connextdds.DataWriter], arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self: List[*rti.connextdds.DataWriter*]*, *arg0: List[*rti.connextdds.DataWriter*]*) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: List[rti.connextdds.DataWriter], s: slice) -> List[rti.connextdds.DataWriter]

Retrieve list elements using a slice object

2. __getitem__(self: List[rti.connextdds.DataWriter], arg0: int) -> rti.connextdds.DataWriter

__hash__ = None

__iadd__(*self: List[*rti.connextdds.DataWriter*]*, *arg0: List[*rti.connextdds.DataWriter*]*) → List[*rti.connextdds.DataWriter*]

__imul__(*self: List[*rti.connextdds.DataWriter*]*, *arg0: int*) → List[*rti.connextdds.DataWriter*]

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DataWriterSeq) -> None

2. __init__(self: rti.connextdds.DataWriterSeq, arg0: List[rti.connextdds.DataWriter]) -> None

Copy constructor

3. __init__(self: rti.connextdds.DataWriterSeq, arg0: Iterable) -> None

__iter__(*self: List[*rti.connextdds.DataWriter*]*) → Iterator

__len__(*self: List[*rti.connextdds.DataWriter*]*) → int

__module__ = 'rti.connextdds'

__mul__(*self: List[*rti.connextdds.DataWriter*]*, *arg0: int*) → List[*rti.connextdds.DataWriter*]

__ne__(*self: List[*rti.connextdds.DataWriter*]*, *arg0: List[*rti.connextdds.DataWriter*]*) → bool

__rmul__(*self: List[*rti.connextdds.DataWriter*]*, *arg0: int*) → List[*rti.connextdds.DataWriter*]

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: List[rti.connextdds.DataWriter], arg0: int, arg1: rti.connextdds.DataWriter) -> None
2. __setitem__(self: List[rti.connextdds.DataWriter], arg0: slice, arg1: List[rti.connextdds.DataWriter]) -> None

Assign list elements using a slice object

append(*self: List[*rti.connextdds.DataWriter*]*, *x:* rti.connextdds.DataWriter) → None

Add an item to the end of the list

clear(*self: List[*rti.connextdds.DataWriter*]*) → None

Clear the contents

count(*self: List[*rti.connextdds.DataWriter*]*, *x:* rti.connextdds.DataWriter) → int

Return the number of times `x` appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: List[rti.connextdds.DataWriter], L: List[rti.connextdds.DataWriter]) -> None

Extend the list by appending all the items in the given list

2. extend(self: List[rti.connextdds.DataWriter], L: Iterable) -> None

Extend the list by appending all the items in the given list

insert(*self: List[*rti.connextdds.DataWriter*]*, *i: int*, *x:* rti.connextdds.DataWriter) → None

Insert an item at a given position.

pop(**args*, ***kwargs*)

Overloaded function.

1. pop(self: List[rti.connextdds.DataWriter]) -> rti.connextdds.DataWriter

Remove and return the last item

2. pop(self: List[rti.connextdds.DataWriter], i: int) -> rti.connextdds.DataWriter

Remove and return the item at index `i`

remove(*self: List[*rti.connextdds.DataWriter*]*, *x:* rti.connextdds.DataWriter) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**DataWriterShmemRefTransferModeSettings**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.DataWriterShmemRefTransferModeSettings, *arg0:* rti.connextdds.DataWriterShmemRefTransferModeSettings) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.DataWriterShmemRefTransferModeSettings) → None

Creates an instance with the default settings.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DataWriterShmemRefTransferModeSettings, *arg0:* rti.connextdds.DataWriterShmemRefTransferModeSettings) → bool

Test for inequality.

property enable_data_consistency_check

Controls if samples can be checked for consistency.

class rti.connextdds.**DataWriterTransferMode**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.DataWriterTransferMode, *arg0:* rti.connextdds.DataWriterTransferMode) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.DataWriterTransferMode) → None

Creates a DataWriterTransferMode qos policy with default values.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DataWriterTransferMode, *arg0:* rti.connextdds.DataWriterTransferMode) → bool

Test for inequality.

property shmem_ref_settings

Settings related to transferring data using shared memory references.

class rti.connextdds.**Database**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.Database, *arg0:* rti.connextdds.Database) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.Database) → None

Create a Database QoS policy with default settings.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Database, *arg0:* rti.connextdds.Database) → bool

Test for inequality.

property cleanup_period

Get/set the cleanup period.

property initial_records

Get/set the number of database records to be allocated initially.

property initial_weak_references

Get/set the initial number of weak references.

property max_skiplist_level

Get/set the skiplist level.

property max_weak_references

Get/set the maximum number of weak references.

property shutdown_cleanup_period

Get/set the shutdown cleanup period.

property shutdown_timeout

Get/set the shutdown timeout.

property thread

Get/set the thread settings for the Database thread.

class rti.connextdds.**Deadline**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.Deadline, *arg0:* rti.connextdds.Deadline) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Deadline) -> None

Creates the default deadline, with an infinite period.

2. __init__(self: rti.connextdds.Deadline, period: rti.connextdds.Duration) -> None

Creates a deadline policy with the specified period.

3. __init__(self: rti.connextdds.Deadline, sec: int, nanosec: int) -> None

Creates a deadline policy with the specified period.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Deadline, *arg0:* rti.connextdds.Deadline) → bool

Test for inequality.

property period

The duration of the deadline period.

class rti.connextdds.**DestinationOrder**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.DestinationOrder, *arg0:* rti.connextdds.DestinationOrder) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DestinationOrder) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.DestinationOrder, kind: rti.connextdds.DestinationOrderKind) -> None

Creates a policy with the specified destination order kind.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DestinationOrder, *arg0:* rti.connextdds.DestinationOrder) → bool

Test for inequality.

property kind

The destination order kind.

property scope

The destination order scope.

property source_timestamp_tolerance

The allowed tolerance between source timestamps of consecutive samples.

class rti.connextdds.**DestinationOrderKind**

Bases: pybind11_object

BY_RECEPTION_TIMESTAMP =
<DestinationOrderKind.BY_RECEPTION_TIMESTAMP: 0>

BY_SOURCE_TIMESTAMP = <DestinationOrderKind.BY_SOURCE_TIMESTAMP: 1>

class DestinationOrderKind

Bases: pybind11_object

Members:

BY_RECEPTION_TIMESTAMP : [default] Indicates that data is ordered based on the reception time at each Subscriber.

Since each subscriber may receive the data at different times there is no guarantee that the changes will be seen in the same order. Consequently, it is possible for each subscriber to end up with a different final value for the data.

> BY_SOURCE_TIMESTAMP : Indicates that data is ordered based on a time-stamp placed at the source (by RTI Connext or by the application).

In any case this guarantees a consistent final value for the data in all subscribers.

Note: If Batching is needed along with DestinationOrderKind.BY_SOURCE_TIMESTAMP and DestinationOrderScopeKind.INSTANCE, then the Batch.source_timestamp_resolution and Batch.thread_safe_write setting of Batch should be set to Duration.zero() and true respectively.

BY_RECEPTION_TIMESTAMP =
<DestinationOrderKind.BY_RECEPTION_TIMESTAMP: 0>

BY_SOURCE_TIMESTAMP =
<DestinationOrderKind.BY_SOURCE_TIMESTAMP: 1>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.DestinationOrderKind.DestinationOrderKind) → int

__init__(*self:* rti.connextdds.DestinationOrderKind.DestinationOrderKind, *value: int*) → None

__int__(*self:* rti.connextdds.DestinationOrderKind.DestinationOrderKind) → int

__members__ = {'BY_RECEPTION_TIMESTAMP':
<DestinationOrderKind.BY_RECEPTION_TIMESTAMP: 0>,
'BY_SOURCE_TIMESTAMP':
<DestinationOrderKind.BY_SOURCE_TIMESTAMP: 1>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.DestinationOrderKind.DestinationOrderKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

__eq__ (*self:* rti.connextdds.DestinationOrderKind, *arg0:* rti.connextdds.DestinationOrderKind) → bool

Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.DestinationOrderKind, *arg0:* rti.connextdds.DestinationOrderKind) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.DestinationOrderKind, *arg0:* rti.connextdds.DestinationOrderKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DestinationOrderKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.DestinationOrderKind, arg0: rti.connextdds.DestinationOrderKind.DestinationOrderKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.DestinationOrderKind) → *rti.connextdds.DestinationOrderKind.DestinationOrderKind*

Int conversion.

__le__ (*self:* rti.connextdds.DestinationOrderKind, *arg0:* rti.connextdds.DestinationOrderKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.DestinationOrderKind, *arg0:* rti.connextdds.DestinationOrderKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DestinationOrderKind, *arg0:* rti.connextdds.DestinationOrderKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.DestinationOrderKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**DestinationOrderScopeKind**

Bases: pybind11_object

class DestinationOrderScopeKind

Bases: pybind11_object

Members:

INSTANCE : [default] Indicates that data is ordered on a per instance basis if used along with DestinationOrderKind.BY_SOURCE_TIMESTAMP.

The source timestamp of the current sample is compared to the source timestamp of the previously received sample for the same instance. The tolerance check is also applied per instance.

TOPIC : Indicates that data is ordered on a per topic basis if used along with DestinationOrderKind.BY_SOURCE_TIMESTAMP.

The source timestamp of the current sample is compared to the source timestamp of the previously received sample for the same topic. The tolerance check is also applied per topic.

INSTANCE = <DestinationOrderScopeKind.INSTANCE: 0>

TOPIC = <DestinationOrderScopeKind.TOPIC: 1>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.DestinationOrderScopeKind.DestinationOrderScopeKind) → int

__init__(*self:* rti.connextdds.DestinationOrderScopeKind.DestinationOrderScopeKind, *value: int*) → None

__int__(*self:* rti.connextdds.DestinationOrderScopeKind.DestinationOrderScopeKind) → int

__members__ = {'INSTANCE': <DestinationOrderScopeKind.INSTANCE: 0>, 'TOPIC': <DestinationOrderScopeKind.TOPIC: 1>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.DestinationOrderScopeKind.DestinationOrderScopeKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

INSTANCE = <DestinationOrderScopeKind.INSTANCE: 0>

TOPIC = <DestinationOrderScopeKind.TOPIC: 1>

__eq__ (*self:* rti.connextdds.DestinationOrderScopeKind, *arg0:* rti.connextdds.DestinationOrderScopeKind) → bool

Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.DestinationOrderScopeKind, *arg0:* rti.connextdds.DestinationOrderScopeKind) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.DestinationOrderScopeKind, *arg0:* rti.connextdds.DestinationOrderScopeKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DestinationOrderScopeKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.DestinationOrderScopeKind, arg0: rti.connextdds.DestinationOrderScopeKind.DestinationOrderScopeKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.DestinationOrderScopeKind) → *rti.connextdds.DestinationOrderScopeKind.DestinationOrderScopeKind*

Int conversion.

__le__ (*self:* rti.connextdds.DestinationOrderScopeKind, *arg0:* rti.connextdds.DestinationOrderScopeKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.DestinationOrderScopeKind, *arg0:* rti.connextdds.DestinationOrderScopeKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DestinationOrderScopeKind, *arg0:* rti.connextdds.DestinationOrderScopeKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.DestinationOrderScopeKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**Discovery**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.Discovery, *arg0:* rti.connextdds.Discovery) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.Discovery) → None

Creates the default policy.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.Discovery, *arg0:* rti.connextdds.Discovery) → bool

Test for inequality.

property accept_unknown_peers

Whether to accept a new participant that is not in the initial peers list.

property enable_endpoint_discovery

Whether to automatically enable endpoint discovery for all the remote participants.

property enabled_transports

The transports (by their aliases) available for the discovery mechanism.

This property's getter returns a deep copy.

property initial_peers

The initial list of peers that the discovery mechanism will contact to announce this DomainParticipant.

This property's getter returns a deep copy.

property metatraffic_transport_priority

The transport priority to use for the Discovery meta-traffic.

property multicast_receive_addresses

The multicast group addresses on which discovery-related meta-traffic can be received by the DomainParticipant.

This property's getter returns a deep copy.

class rti.connextdds.**DiscoveryConfig**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.DiscoveryConfig, *arg0:* rti.connextdds.DiscoveryConfig) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.DiscoveryConfig) → None

Creates the default policy.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DiscoveryConfig, *arg0:* rti.connextdds.DiscoveryConfig) → bool

Test for inequality.

property asynchronous_publisher

Asynchronous publishing settings for the discovery Publisher and all entities that are created by it.

property builtin_discovery_plugins

The kind mask for built-in discovery plugins.

property default_domain_announcement_period

The period to announce a participant to the default domain 0.

This property's getter returns a deep copy.

property dns_tracker_polling_period

Duration that specifies the period used by the DNS tracker to poll the DNS service and check for changes in the hostnames.

property enabled_builtin_channels

The mask specifying which built-in channels should be enabled.

property endpoint_type_object_lb_serialization_threshold

Option to reduce the size required to propagate a TypeObject in Simple Endpoint Discovery.

property ignore_default_domain_announcements

Used to ignore the announcements received by a participant on the default domain 0 corresponding to participants running on domains IDs other than 0.

property initial_participant_announcements

The number of initial announcements sent when a participant is first enabled or when a remote participant is newly discovered.

property locator_reachability_assert_period

Period at which this DomainParticipant will assert the locators discovered from other DomainParticipants.

This property's getter returns a deep copy.

property locator_reachability_change_detection_period

Period at which this DomainParticipant will check if its locators are reachable from other DomainParticipants.

This property's getter returns a deep copy.

property locator_reachability_lease_duration

The time period after which other DomainParticipants can consider one of their locators as "unreachable" if they do not receive a REACHABILITY PING from this DomainParticipant.

This property's getter returns a deep copy.

property max_initial_participant_announcement_period

The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.

property max_liveliness_loss_detection_period

The period to assert liveliness for the participant.

This property's getter returns a deep copy.

property min_initial_participant_announcement_period

The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.

This property's getter returns a deep copy.

property new_remote_participant_announcements

The number of participant announcements sent when a remote participant is newly discovered.

property participant_announcement_period

The period at which a participant announces itself to potential peers when using the Simple Participant Discovery Protocol 2.0 (SPDP2)

property participant_liveliness_assert_period

The period to assert liveliness for the participant.

This property's getter returns a deep copy.

property participant_liveliness_lease_duration

The liveliness lease duration for the participant.

property participant_message_reader

Reliability policy for a built-in participant message reader.

property participant_message_reader_reliability_kind

Reliability policy for a built-in participant message reader.

property participant_message_writer

Reliability policy for a built-in participant message writer.

property participant_reader_resource_limits

Resource limits.

property publication_reader

RTPS protocol-related configuration settings for the RTPS reliable reader associated to a built-in publication reader.

property publication_reader_resource_limits

Publication reader resource limits.

property publication_writer

RTPS protocol-related configuration settings for the RTPS reliable writer associated to a built-in publication reader.

property publication_writer_data_lifecycle

riter data lifecycle settings for a built-in publication writer.

property publication_writer_publish_mode

Publish mode policy for the built-in publication writer.

property remote_participant_purge_kind

The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.

property secure_volatile_reader

RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.

property secure_volatile_writer

RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.

property secure_volatile_writer_publish_mode

Publish mode policy for the built-in secure volatile writer.

property service_request_reader

RTPS protocol-related configuration settings for the RTPS reliable reader associated to a built-in publication reader.

property service_request_writer

RTPS protocol-related configuration settings for the RTPS reliable writer associated to the built-in ServiceRequest writer.

property service_request_writer_data_lifecycle

Writer data lifecycle settings for a built-in ServiceRequest writer.

property service_request_writer_publish_mode

Publish mode policy for the built-in service request writer.

property subscription_reader

RTPS protocol-related configuration settings for the RTPS reliable reader associated to a built-in subscription reader.

property subscription_reader_resource_limits

Subscription reader resource limits.

property subscription_writer

RTPS protocol-related configuration settings for the RTPS reliable writer associated to a built-in subscription reader.

property subscription_writer_data_lifecycle

riter data lifecycle settings for a built-in subscription writer.

property subscription_writer_publish_mode

Publish mode policy for the built-in subscription writer.

class rti.connextdds.**DiscoveryConfigBuiltinChannelKindMask**

Bases: pybind11_object

ALL = 1111111

NONE = 0000000

SERVICE_REQUEST = 0000011

__and__(*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, *arg0:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) → *rti.connextdds.DiscoveryConfigBuiltinChannelKindMask*

Bitwise logical AND of masks.

__bool__(*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) → bool

Test if any bits are set.

__contains__(*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, *arg0:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) → bool

__eq__(*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, *arg0:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) → bool

Compare masks for equality.

__getitem__(*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, *arg0: int*) → bool

Get individual mask bit.

__hash__ = None

__iand__(*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, *arg0:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) → *rti.connextdds.DiscoveryConfigBuiltinChannelKindMask*

Set mask to logical AND with another mask.

__ilshift__(*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, *arg0: int*) → *rti.connextdds.DiscoveryConfigBuiltinChannelKindMask*

Left shift bits in mask.

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) -> None

Create a DiscoveryConfigBuiltinChannelKindMask equivalent to DiscoveryConfigBuiltinChannelKindMask.NONE

2. __init__(self: rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, value: int) -> None

Creates a mask from the bits in an integer.

__int__(*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) → int

Convert mask to int.

__ior__ (*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, *arg0:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) → *rti.connextdds.DiscoveryConfigBuiltinChannelKindMask*

Set mask to logical OR with another mask.

__irshift__ (*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, *arg0: int*) → *rti.connextdds.DiscoveryConfigBuiltinChannelKindMask*

Right shift bits in mask.

__ixor__ (*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, *arg0:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) → *rti.connextdds.DiscoveryConfigBuiltinChannelKindMask*

Set mask to logical XOR with another mask.

__lshift__ (*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, *arg0: int*) → *rti.connextdds.DiscoveryConfigBuiltinChannelKindMask*

Left shift bits in mask.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, *arg0:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) → bool

Compare masks for inequality.

__or__ (*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, *arg0:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) → *rti.connextdds.DiscoveryConfigBuiltinChannelKindMask*

Bitwise logical OR of masks.

__repr__ (*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) → str

Convert mask to string.

__rshift__ (*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, *arg0: int*) → *rti.connextdds.DiscoveryConfigBuiltinChannelKindMask*

Right shift bits in mask.

__setitem__ (*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, *arg0: int*, *arg1: bool*) → None

Set individual mask bit

__str__ (*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) → str

Convert mask to string.

__xor__ (*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, *arg0:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) → *rti.connextdds.DiscoveryConfigBuiltinChannelKindMask*

Bitwise logical XOR of masks.

property count

Returns the number of bits set in the mask.

flip (**args*, ***kwargs*)

Overloaded function.

1. flip(self: rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) -> rti.connextdds.DiscoveryConfigBuiltinChannelKindMask

Flip all bits in the mask.

2. flip(self: rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, pos: int) -> rti.connextdds.DiscoveryConfigBuiltinChannelKindMask

Flip the mask bit at the specified position.

reset (**args*, ***kwargs*)

Overloaded function.

1. reset(self: rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) -> rti.connextdds.DiscoveryConfigBuiltinChannelKindMask

Clear all bits in the mask.

2. reset(self: rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, pos: int) -> rti.connextdds.DiscoveryConfigBuiltinChannelKindMask

Clear the mask bit at the specified position.

set (**args*, ***kwargs*)

Overloaded function.

1. set(self: rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) -> rti.connextdds.DiscoveryConfigBuiltinChannelKindMask

Set all bits in the mask.

2. set(self: rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, pos: int, value: bool = True) -> rti.connextdds.DiscoveryConfigBuiltinChannelKindMask

Set the mask bit at the specified position to the provided value (default: true).

property size

Returns the number of bits in the mask type.

test (*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask, *pos: int*) → bool

Test whether the mask bit at position "pos" is set.

test_all (*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) → bool

Test if all bits are set.

test_any (*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) → bool

Test if any bits are set.

test_none (*self:* rti.connextdds.DiscoveryConfigBuiltinChannelKindMask) → bool

Test if none of the bits are set.

class rti.connextdds.**DiscoveryConfigBuiltinPluginKindMask**

Bases: pybind11_object

NONE = 00000

SDP = 00011

SDP2 = 10010

SEDP = 00010

SPDP = 00001

SPDP2 = 10000

__and__(*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, *arg0:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) → *rti.connextdds.DiscoveryConfigBuiltinPluginKindMask*

Bitwise logical AND of masks.

__bool__(*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) → bool

Test if any bits are set.

__contains__(*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, *arg0:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) → bool

__eq__(*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, *arg0:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) → bool

Compare masks for equality.

__getitem__(*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, *arg0: int*) → bool

Get individual mask bit.

__hash__ = None

__iand__(*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, *arg0:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) → *rti.connextdds.DiscoveryConfigBuiltinPluginKindMask*

Set mask to logical AND with another mask.

__ilshift__(*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, *arg0: int*) → *rti.connextdds.DiscoveryConfigBuiltinPluginKindMask*

Left shift bits in mask.

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) -> None

Create a DiscoveryConfigBuiltinPluginKindMask equivalent to DiscoveryConfigBuiltinPluginKindMask.NONE

2. __init__(self: rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, value: int) -> None

Creates a mask from the bits in an integer.

__int__ (*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) → int

Convert mask to int.

__ior__ (*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, *arg0:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) → *rti.connextdds.DiscoveryConfigBuiltinPluginKindMask*

Set mask to logical OR with another mask.

__irshift__ (*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, *arg0: int*) → *rti.connextdds.DiscoveryConfigBuiltinPluginKindMask*

Right shift bits in mask.

__ixor__ (*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, *arg0:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) → *rti.connextdds.DiscoveryConfigBuiltinPluginKindMask*

Set mask to logical XOR with another mask.

__lshift__ (*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, *arg0: int*) → *rti.connextdds.DiscoveryConfigBuiltinPluginKindMask*

Left shift bits in mask.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, *arg0:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) → bool

Compare masks for inequality.

__or__ (*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, *arg0:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) → *rti.connextdds.DiscoveryConfigBuiltinPluginKindMask*

Bitwise logical OR of masks.

__repr__ (*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) → str

Convert mask to string.

__rshift__ (*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, *arg0: int*) → *rti.connextdds.DiscoveryConfigBuiltinPluginKindMask*

Right shift bits in mask.

__setitem__ (*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, *arg0: int*, *arg1: bool*) → None

Set individual mask bit

__str__ (*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) → str

Convert mask to string.

__xor__ (*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, *arg0:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) → *rti.connextdds.DiscoveryConfigBuiltinPluginKindMask*

Bitwise logical XOR of masks.

property count

Returns the number of bits set in the mask.

flip(**args*, ***kwargs*)

Overloaded function.

1. flip(self: rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) -> rti.connextdds.DiscoveryConfigBuiltinPluginKindMask

Flip all bits in the mask.

2. flip(self: rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, pos: int) -> rti.connextdds.DiscoveryConfigBuiltinPluginKindMask

Flip the mask bit at the specified position.

reset(**args*, ***kwargs*)

Overloaded function.

1. reset(self: rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) -> rti.connextdds.DiscoveryConfigBuiltinPluginKindMask

Clear all bits in the mask.

2. reset(self: rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, pos: int) -> rti.connextdds.DiscoveryConfigBuiltinPluginKindMask

Clear the mask bit at the specified position.

set(**args*, ***kwargs*)

Overloaded function.

1. set(self: rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) -> rti.connextdds.DiscoveryConfigBuiltinPluginKindMask

Set all bits in the mask.

2. set(self: rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, pos: int, value: bool = True) -> rti.connextdds.DiscoveryConfigBuiltinPluginKindMask

Set the mask bit at the specified position to the provided value (default: true).

property size

Returns the number of bits in the mask type.

test(*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask, *pos: int*) → bool

Test whether the mask bit at position "pos" is set.

test_all(*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) → bool

Test if all bits are set.

test_any (*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) → bool

Test if any bits are set.

test_none (*self:* rti.connextdds.DiscoveryConfigBuiltinPluginKindMask) → bool

Test if none of the bits are set.

class rti.connextdds.**DomainParticipant**

Bases: *IEntity*

Container for all Entity objects.

- It acts as a container for all other Entity objects.
- It acts as a factory for the Publisher, Subscriber, Topic and Entity objects.
- It represents the participation of the application on a communication plane that isolates applications running on the same set of physical computers from each other. A domain establishes a virtual network linking all applications that share the same domainId and isolating them from applications running on different domains. In this way, several independent distributed applications can coexist in the same physical network without interfering, or even being aware of each other.
- It provides administration services in the domain, offering operations that allow the application to ignore locally any information about a given participant (DomainParticipant.ignore), publication (ignore_publication), subscription (ignore_subscription) or topic (ignore_topic).

__enter__ (*self:* rti.connextdds.DomainParticipant) → *rti.connextdds.DomainParticipant*

Enter a context for this Domain Participant, to be cleaned up on exiting context

__eq__ (*self:* rti.connextdds.DomainParticipant, *arg0:* rti.connextdds.DomainParticipant) → bool

Test for equality.

__exit__ (*self:* rti.connextdds.DomainParticipant, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for this Domain Participant, cleaning up resources.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DomainParticipant, domain_id: int) -> None

Create a new DomainParticipant with default QoS.

2. __init__(self: rti.connextdds.DomainParticipant, domain_id: int, qos: rti.connextdds.DomainParticipantQos, listener: rti.connextdds.DomainParticipantListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a new DomainParticipant

3. __init__(self: rti.connextdds.DomainParticipant, entity: rti.connextdds.IEntity) -> None

Downcast an IEntity to a DomainParticipant.

__lshift__ (*self:* rti.connextdds.DomainParticipant, *arg0:* rti.connextdds.DomainParticipantQos) → *rti.connextdds.DomainParticipant*

Set the domain participant's QoS.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DomainParticipant, *arg0:* rti.connextdds.DomainParticipant) → bool

Test for inequality.

__rshift__ (*self:* rti.connextdds.DomainParticipant, *arg0:* rti.connextdds.DomainParticipantQos) → *rti.connextdds.DomainParticipant*

Get the domain participant's QoS.

add_peer (*self:* rti.connextdds.DomainParticipant, *peer: str*) → None

Attempt to contact an additional peer participant.

add_peers (*self:* rti.connextdds.DomainParticipant, *peers:* rti.connextdds.StringSeq) → None

Add a sequence of peers to be contacted.

assert_liveliness (*self:* rti.connextdds.DomainParticipant) → None

Manually assert the liveliness of the DomainParticipant.

banish_ignored_participants (*self:* rti.connextdds.DomainParticipant) → None

Prevents ignored remote DomainParticipants from receiving traffic from the local DomainParticipant.

property builtin_subscriber

Get the built-in subscriber for the DomainParticipant.

close_contained_entities (*self:* rti.connextdds.DomainParticipant) → None

Destroy all contained entities.

contains_entity (*self:* rti.connextdds.DomainParticipant, *handle:* rti.connextdds.InstanceHandle) → bool

Check whether or not the given handle represents an Entity that was created from the DomainParticipant.

property current_time

Get the current time.

property default_datareader_qos

The default DataReaderQos.

This property's getter returns a deep copy.

property default_datawriter_qos

The default DataWriterQos.

This property's getter returns a deep copy.

default_participant_qos = <rti.connextdds.DomainParticipantQos object>

property default_publisher_qos

The default PublisherQos.

This property's getter returns a deep copy.

property default_subscriber_qos

The default SubscriberQos.

This property's getter returns a deep copy.

property default_topic_qos

The default TopicQos.

This property's getter returns a deep copy.

delete_durable_subscription(*self:* rti.connextdds.DomainParticipant, *group:* rti.connextdds.EndpointGroup) → None

Deletes an existing Durable Subscription on all Persistence Services.

discovered_participant_data(**args*, ***kwargs*)

Overloaded function.

1. discovered_participant_data(self: rti.connextdds.DomainParticipant, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.ParticipantBuiltinTopicData

Retrieve DomainParticipant information by handle.

2. discovered_participant_data(self: rti.connextdds.DomainParticipant, handles: rti.connextdds.InstanceHandleSeq) -> rti.connextdds.ParticipantBuiltinTopicDataSeq

Retrieve DomainParticipant information with a sequence of handles.

discovered_participant_subject_name(*self:* rti.connextdds.DomainParticipant, *handle:* rti.connextdds.InstanceHandle) → Optional[str]

Returns the entity name for the specified DomainParticipant InstanceHandle.

discovered_participants(*self:* rti.connextdds.DomainParticipant) → *rti.connextdds.InstanceHandleSeq*

Retrieves the instance handles of other DomainParticipants discovered by this one.

discovered_participants_from_subject_name(*self:* rti.connextdds.DomainParticipant, *subject_name: Optional[str]*) → *rti.connextdds.InstanceHandleSeq*

Returns the list of InstanceHandles corresponding toparticipants with the given entity name.

discovered_topic_data(**args*, ***kwargs*)

Overloaded function.

1. discovered_topic_data(self: rti.connextdds.DomainParticipant, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.TopicBuiltinTopicData

Get information about a discovered topic using its handle.

2. discovered_topic_data(self: rti.connextdds.DomainParticipant, topic: rti.connextdds.IEntity) -> rti.connextdds.TopicBuiltinTopicData

Get information about a discovered topic.

3. discovered_topic_data(self: rti.connextdds.DomainParticipant, handles: rti.connextdds.InstanceHandleSeq) -> rti.connextdds.TopicBuiltinTopicDataSeq

Get information about a discovered topics with their handles.

4. discovered_topic_data(self: rti.connextdds.DomainParticipant) -> rti.connextdds.TopicBuiltinTopicDataSeq

Get information about all discovered topics.

discovered_topics(*self:* rti.connextdds.DomainParticipant) → *rti.connextdds.InstanceHandleSeq*

Get all Topic handles discovered by this DomainParticipant.

property domain_id

The unique domain identifier.

static finalize_participant_factory() → None

Finalize the DomainParticipantFactory

static find(**args*, ***kwargs*)

Overloaded function.

1. find(name: str) -> Optional[rti.connextdds.DomainParticipant]

Find a local DomainParticipant by its name.

2. find() -> rti.connextdds.DomainParticipantSeq

Find all local DomainParticipants.

3. find(domain_id: int) -> Optional[rti.connextdds.DomainParticipant]

Find a local DomainParticipant with the given domain ID. If more than one DomainParticipant on the same domain exists in the application, it is not specified which will be returned.

find_contentfilter(*self:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.ContentFilterBase*]

Find content filter previously registered to this DomainParticipant.

find_datareader(*self:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.AnyDataReader*]

Find a DataReader by its name.

find_datawriter(*self:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.AnyDataWriter*]

Find a DataWriter by its name.

find_flow_controller (*self:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.FlowController*]

Find a FlowController configured in this DomainParticipant.

find_publisher (*self:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.Publisher*]

Lookup a Publisher within the DomainParticipant by its entity name.

find_publishers (*self:* rti.connextdds.DomainParticipant) → *rti.connextdds.PublisherSeq*

Find all Publishers within the DomainParticipant.

find_registered_content_filters (*self:* rti.connextdds.DomainParticipant) → *rti.connextdds.StringSeq*

Retrieve a list of all registered content filter names.

find_subscriber (*self:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.Subscriber*]

Find a Subscriber in the DomainParticipant by its entity name.

find_subscribers (*self:* rti.connextdds.DomainParticipant) → *rti.connextdds.SubscriberSeq*

Find all subscribers within the DomainParticipant.

find_topics (*self:* rti.connextdds.DomainParticipant) → *rti.connextdds.AnyTopicSeq*

Find all Topics in the DomainParticipant.

ignore_datareader (*self:* rti.connextdds.DomainParticipant, *handle:* rti.connextdds.InstanceHandle) → None

Ignore a DataReader matching the provided handle.

ignore_datareaders (*self:* rti.connextdds.DomainParticipant, *handles:* rti.connextdds.InstanceHandleSeq) → None

Ignore a list of DataReaders specified by their handles.

ignore_datawriter (*self:* rti.connextdds.DomainParticipant, *handle:* rti.connextdds.InstanceHandle) → None

Ignore a DataWriter matching the provided handle.

ignore_datawriters (*self:* rti.connextdds.DomainParticipant, *handles:* rti.connextdds.InstanceHandleSeq) → None

Ignore a list of DataWriters specified by their handles.

ignore_participant (*self:* rti.connextdds.DomainParticipant, *handle:* rti.connextdds.InstanceHandle) → None

Ignore a DomainParticipant given it's handle.

ignore_participants (*self:* rti.connextdds.DomainParticipant, *arg0:* rti.connextdds.InstanceHandleSeq) → None

Ignore DomainParticipants given a list of handles.

ignore_topic (*self:* rti.connextdds.DomainParticipant, *handle:* rti.connextdds.InstanceHandle) → None

Ignore a Topic matching the provided handle.

ignore_topics (*self:* rti.connextdds.DomainParticipant, *handles:* rti.connextdds.InstanceHandleSeq) → None

Ignore a list of Topics specified by their handles.

property implicit_publisher

Get the implicit Publisher for the DomainParticipant.

property implicit_subscriber

Get the implicit Subscriber for the DomainParticipant.

is_type_registered (*self:* rti.connextdds.DomainParticipant, *name: str*) → bool

Check if a type has been registered to this DomainParticipant.

property listener

Get the listener.

participant_factory_qos =
<rti.connextdds.DomainParticipantFactoryQos object>

property participant_protocol_status

Get the protocol status for this participant

property participant_reader

Get the DomainParticipant built-in topic reader.

property publication_reader

Get the publication built-in topic reader.

property qos

Get the domain participant's QoS.

This property's getter returns a deep copy.

register_contentfilter (*self:* rti.connextdds.DomainParticipant, *filter:* rti.connextdds.ContentFilterBase, *name: str*) → None

Register a content filter which can be used to create a ContentFiltertedTopic.

register_durable_subscription (*self:* rti.connextdds.DomainParticipant, *group:* rti.connextdds.EndpointGroup, *topic_name: str*) → None

Registers a Durable Subscription on the specified Topic on all Persistence Services

static register_idl_type (*type: object*, *registered_type_name: str*) → None

Registers a python class so it can be used in XML-based applications and referred to by its registered name.

register_type(*self:* rti.connextdds.DomainParticipant, *name: str*, *type:* rti.connextdds.DynamicType, *serialization_property:* rti.connextdds.DynamicDataTypeSerializationProperty = *DynamicDataTypeSerializationProperty.DEFAULT*) → None

Registers a DynamicType with specific serialization properties.

remove_peer(*self:* rti.connextdds.DomainParticipant, *peer: str*) → None

Remove a peer participant from this list that this DomainParticipant will attempt to communicate with.

remove_peers(*self:* rti.connextdds.DomainParticipant, *peers:* rti.connextdds.StringSeq) → None

Remove a sequence of peers from the contact list.

resume_endpoint_discovery(*self:* rti.connextdds.DomainParticipant, *handle:* rti.connextdds.InstanceHandle) → None

Initiates endpoint discovery with the remote DomainParticipant identified by its InstanceHandle.

property service_request_reader

Get the ServiceRequest built-in topic reader.

set_listener(*self:* rti.connextdds.DomainParticipant, *listener:* rti.connextdds.DomainParticipantListener, *event_mask:* rti.connextdds.StatusMask) → None

Bind the listener and event mask to the DomainParticipant.

property subscription_reader

Get the subscription built-in topic reader.

unregister_contentfilter(*self:* rti.connextdds.DomainParticipant, *name: str*) → None

Unregister content filter previously registered to this DomainParticipant.

unregister_type(*self:* rti.connextdds.DomainParticipant, *name: str*) → None

Unregister a type previously registered to this DomainParticipant.

class rti.connextdds.**DomainParticipantConfigParams**

Bases: pybind11_object

DOMAIN_ID_USE_XML_CONFIG = -1

ENTITY_NAME_USE_XML_CONFIG = 'com.rti.dds.domain.entity_name_use_xml_config'

QOS_ELEMENT_NAME_USE_XML_CONFIG = 'com.rti.dds.domain.qos_element_name_use_xml_config'

__eq__(*self:* rti.connextdds.DomainParticipantConfigParams, *arg0:* rti.connextdds.DomainParticipantConfigParams) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.DomainParticipantConfigParams, *domain_id: int* = *DomainParticipantConfigParams.DOMAIN_ID_USE_XML_CONFIG*, *participant_name: str* = *DomainParticipantConfigParams.ENTITY_NAME_USE_XML_CONFIG*, *qos_library_name: str* = *DomainParticipantConfigParams.QOS_ELEMENT_NAME_USE_XML_CONFIG*, *qos_profile_name: str* = *DomainParticipantConfigParams.QOS_ELEMENT_NAME_USE_XML_CONFIG*, *domain_entity_qos_library_name: str* = *DomainParticipantConfigParams.QOS_ELEMENT_NAME_USE_XML_CONFIG*, *domain_entity_qos_profile_name: str* = *DomainParticipantConfigParams.QOS_ELEMENT_NAME_USE_XML_CONFIG*) → None

Create a DomainParticipantConfigParams object with the specified values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DomainParticipantConfigParams, *arg0:* rti.connextdds.DomainParticipantConfigParams) → bool

Test for inequality.

property domain_entity_qos_library_name

The QoS library name containing the QoS profile from which the all the entities defined under the participant configuraton are created.

property domain_entity_qos_profile_name

The QoS profile name from which the all the entities defined under the participant configuraton are created.

property domain_id

The domain id from which the DomainParticipant is created.

property participant_name

The name assigned to the DomainParticipant.

property participant_qos_library_name

The name of the library containing the DomainParticipant's QoS.

property participant_qos_profile_name

The name of the profile containing the DomainParticipant's QoS.

class rti.connextdds.**DomainParticipantFactoryQos**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.DomainParticipantFactoryQos, *arg0:* rti.connextdds.DomainParticipantFactoryQos) → bool

Test for equality

__hash__ = None

__init__ (*self:* rti.connextdds.DomainParticipantFactoryQos) → None

Create a DomainParticipantFactoryQos with the default value for each policy.

__lshift__ (**args*, ***kwargs*)

Overloaded function.

1. __lshift__(self: rti.connextdds.DomainParticipantFactoryQos, arg0: rti.connextdds.EntityFactory) -> rti.connextdds.DomainParticipantFactoryQos

Set the EntityFactoryQoS.

2. __lshift__(self: rti.connextdds.DomainParticipantFactoryQos, arg0: rti.connextdds.SystemResourceLimits) -> rti.connextdds.DomainParticipantFactoryQos

Set the SystemResourceLimitsQoS.

3. __lshift__(self: rti.connextdds.DomainParticipantFactoryQos, arg0: rti.connextdds.Monitoring) -> rti.connextdds.DomainParticipantFactoryQos

Set the MonitoringQoS.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DomainParticipantFactoryQos, *arg0:* rti.connextdds.DomainParticipantFactoryQos) → bool

Test for inequality.

__rshift__ (**args*, ***kwargs*)

Overloaded function.

1. __rshift__(self: rti.connextdds.DomainParticipantFactoryQos, arg0: rti.connextdds.EntityFactory) -> rti.connextdds.DomainParticipantFactoryQos

Get the EntityFactoryQoS.

2. __rshift__(self: rti.connextdds.DomainParticipantFactoryQos, arg0: rti.connextdds.SystemResourceLimits) -> rti.connextdds.DomainParticipantFactoryQos

Get the SystemResourceLimitsQoS.

3. __rshift__(self: rti.connextdds.DomainParticipantFactoryQos, arg0: rti.connextdds.Monitoring) -> rti.connextdds.DomainParticipantFactoryQos

Get the MonitoringQoS.

__str__ (*self:* rti.connextdds.DomainParticipantFactoryQos) → str

property entity_factory

Get/set EntityFactory QoS.

property monitoring

Get/set Monitoring QoS.

property system_resource_limits

Get/set SystemResourceLimits QoS.

to_string(*self:* rti.connextdds.DomainParticipantFactoryQos, *format:* rti.connextdds.QosPrintFormat = *QosPrintFormat()*, *base:* *Optional[*rti.connextdds.DomainParticipantFactoryQos*]* = *None*, *print_all: bool* = *False*) → str

Convert QoS to string based on params.

class rti.connextdds.**DomainParticipantListener**

Bases: *PublisherListener*, *SubscriberListener*, *AnyTopicListener*

__init__(*self:* rti.connextdds.DomainParticipantListener) → None

__module__ = 'rti.connextdds'

on_invalid_local_identity_status_advance_notice(*self:* rti.connextdds.DomainParticipantListener, *arg0:* rti.connextdds.DomainParticipant, *arg1:* rti.connextdds.InvalidLocalIdentityAdvanceNoticeStatus) → None

On invalid local identity status advance notice callback

class rti.connextdds.**DomainParticipantProtocolStatus**

Bases: pybind11_object

Protocol status of a DomainParticipant

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property corrupted_rtps_message_count

The number of corrupted RTPS messages detected by the domain participant.

property corrupted_rtps_message_count_change

The incremental change in the number of corrupted RTPS messages detected by the domain participant since the last time the status was read.

property last_corrupted_message_timestamp

The timestamp when the last corrupted RTPS message was detected by the domain participant.

class rti.connextdds.**DomainParticipantQos**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.DomainParticipantQos, *arg0:* rti.connextdds.DomainParticipantQos) → bool

Test for equality

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DomainParticipantQos) -> None

Create a DomainParticipantQos with the default value for each policy.

2. __init__(self: rti.connextdds.DomainParticipantQos, participant: rti.connextdds.DomainParticipant) -> None

Create a DomainParticipantQos with settings equivalent to those of the provided DomainParticipant object.

3. __init__(self: rti.connextdds.DomainParticipantQos, other: rti.connextdds.DomainParticipantQos) -> None

Create a copy of a DomainParticipantQos object.

__lshift__ (**args*, ***kwargs*)

Overloaded function.

1. __lshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.UserData) -> rti.connextdds.DomainParticipantQos

Set the UserDataQoS.

2. __lshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.EntityFactory) -> rti.connextdds.DomainParticipantQos

Set the EntityFactoryQoS.

3. __lshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.WireProtocol) -> rti.connextdds.DomainParticipantQos

Set the WireProtocolQoS.

4. __lshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.TransportBuiltin) -> rti.connextdds.DomainParticipantQos

Set the TransportBuiltinQoS.

5. __lshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.Discovery) -> rti.connextdds.DomainParticipantQos

Set the DiscoveryQoS.

6. __lshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.DomainParticipantResourceLimits) -> rti.connextdds.DomainParticipantQos

Set the DomainParticipantResourceLimitsQoS.

7. __lshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.Event) -> rti.connextdds.DomainParticipantQos

Set the EventQoS.

8. __lshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.ReceiverPool) -> rti.connextdds.DomainParticipantQos

Set the ReceiverPoolQoS.

9. __lshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.Database) -> rti.connextdds.DomainParticipantQos

Set the DatabaseQoS.

10. __lshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.DiscoveryConfig) -> rti.connextdds.DomainParticipantQos

Set the DiscoveryConfigQoS.

11. __lshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.Property) -> rti.connextdds.DomainParticipantQos

Set the PropertyQoS.

12. __lshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.EntityName) -> rti.connextdds.DomainParticipantQos

Set the EntityNameQoS.

13. __lshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.TransportMulticastMapping) -> rti.connextdds.DomainParticipantQos

Set the TransportMulticastMappingQoS.

14. __lshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.Partition) -> rti.connextdds.DomainParticipantQos

Set the PartitionQoS.

15. __lshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.TransportUnicast) -> rti.connextdds.DomainParticipantQos

Set the TransportUnicastQoS.

16. __lshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.Service) -> rti.connextdds.DomainParticipantQos

Set the ServiceQoS.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DomainParticipantQos, *arg0:* rti.connextdds.DomainParticipantQos) → bool

Test for inequality.

__rshift__(**args*, ***kwargs*)

Overloaded function.

1. __rshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.UserData) -> rti.connextdds.DomainParticipantQos

Get the UserDataQoS.

2. __rshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.EntityFactory) -> rti.connextdds.DomainParticipantQos

Get the EntityFactoryQoS.

3. __rshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.WireProtocol) -> rti.connextdds.DomainParticipantQos

Get the WireProtocolQoS.

4. __rshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.TransportBuiltin) -> rti.connextdds.DomainParticipantQos

Get the TransportBuiltinQoS.

5. __rshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.Discovery) -> rti.connextdds.DomainParticipantQos

Get the DiscoveryQoS.

6. __rshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.DomainParticipantResourceLimits) -> rti.connextdds.DomainParticipantQos

Get the DomainParticipantResourceLimitsQoS.

7. __rshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.Event) -> rti.connextdds.DomainParticipantQos

Get the EventQoS.

8. __rshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.ReceiverPool) -> rti.connextdds.DomainParticipantQos

Get the ReceiverPoolQoS.

9. __rshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.Database) -> rti.connextdds.DomainParticipantQos

Get the DatabaseQoS.

10. __rshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.DiscoveryConfig) -> rti.connextdds.DomainParticipantQos

Get the DiscoveryConfigQoS.

11. __rshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.Property) -> rti.connextdds.DomainParticipantQos

Get the PropertyQoS.

12. __rshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.EntityName) -> rti.connextdds.DomainParticipantQos

Get the EntityNameQoS.

13. __rshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.TransportMulticastMapping) -> rti.connextdds.DomainParticipantQos

Get the TransportMulticastMappingQoS.

14. __rshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.Partition) -> rti.connextdds.DomainParticipantQos

Get the PartitionQoS.

15. __rshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.TransportUnicast) -> rti.connextdds.DomainParticipantQos

Get the TransportUnicastQoS.

16. __rshift__(self: rti.connextdds.DomainParticipantQos, arg0: rti.connextdds.Service) -> rti.connextdds.DomainParticipantQos

Get the ServiceQoS.

__str__ (*self:* rti.connextdds.DomainParticipantQos) → str

property database

Get/set Database QoS.

property default_unicast

Get/set TransportUnicast QoS.

property discovery

Get/set Discovery QoS.

property discovery_config

Get/set DiscoveryConfig QoS.

property entity_factory

Get/set EntityFactory QoS.

property event

Get/set Event QoS.

property participant_name

Get/set EntityName QoS.

property partition

Get/set Partition QoS.

property property

Get/set Property QoS.

property receiver_pool

Get/set ReceiverPool QoS.

property resource_limits

Get/set DomainParticipantResourceLimits QoS.

property service

Get/set Service QoS.

to_string (*self:* rti.connextdds.DomainParticipantQos, *format:* rti.connextdds.QosPrintFormat = *QosPrintFormat()*, *base: Optional[*rti.connextdds.DomainParticipantQos*]* = *None*, *print_all: bool* = *False*) → str

Convert QoS to string based on params.

property transport_builtin

Get/set TransportBuiltin QoS.

property transport_multicast_mapping

Get/set TransportMulticastMapping QoS.

property user_data

Get/set UserData QoS.

property wire_protocol

Get/set WireProtocol QoS.

class rti.connextdds.**DomainParticipantResourceLimits**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.DomainParticipantResourceLimits, *arg0:* rti.connextdds.DomainParticipantResourceLimits) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.DomainParticipantResourceLimits) → None

Create a default policy.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DomainParticipantResourceLimits, *arg0:* rti.connextdds.DomainParticipantResourceLimits) → bool

Test for inequality.

property channel_filter_expression_max_length

Maximum length of a channel ChannelSettings.filter_expression in a MultiChannel DataWriter.

property channel_seq_max_length

Maximum number of channels that can be specified in MultiChannel for MultiChannel DataWriters.

property content_filter_allocation

Allocation settings applied to content filter.

This property's getter returns a deep copy.

property content_filter_hash_buckets

Number of hash buckets for content filters.

property content_filtered_topic_allocation

Allocation settings applied to content filtered topic.

This property's getter returns a deep copy.

property content_filtered_topic_hash_buckets

Number of hash buckets for content filtered topics.

property contentfilter_property_max_length

This field is the maximum length of all data related to a Content-filtered topic.

property deserialized_type_object_dynamic_allocation_threshold

A threshold, in bytes, for dynamic memory allocation for the deserialized TypeObject.

property flow_controller_allocation

Allocation settings applied to flow controllers.

This property's getter returns a deep copy.

property flow_controller_hash_buckets

Number of hash buckets for flow controllers.

property ignored_entity_allocation

Allocation settings applied to ignored entities.

This property's getter returns a deep copy.

property ignored_entity_hash_buckets

Number of hash buckets for ignored entities.

property ignored_entity_replacement_kind

Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in DomainParticipantResourceLimits.ignored_entity_allocation are reached.

property local_publisher_allocation

Allocation settings applied to local Publisher.

This property's getter returns a deep copy.

property local_publisher_hash_buckets

Number of hash buckets for local Publisher.

property local_reader_allocation

Allocation settings applied to local DataReaders.

This property's getter returns a deep copy.

property local_reader_hash_buckets

Hash buckets settings applied to local DataReaders.

property local_subscriber_allocation

Allocation settings applied to local Subscriber.

This property's getter returns a deep copy.

property local_subscriber_hash_buckets

Number of hash buckets for local Subscriber.

property local_topic_allocation

Allocation settings applied to local Topic.

This property's getter returns a deep copy.

property local_topic_hash_buckets

Number of hash buckets for local Topic.

property local_writer_allocation

Allocation settings applied to local DataWriters.

This property's getter returns a deep copy.

property local_writer_hash_buckets

Hash buckets settings applied to local DataWriters.

property matching_reader_writer_pair_allocation

Allocation settings applied to matching local reader and remote/local writer pairs.

This property's getter returns a deep copy.

property matching_reader_writer_pair_hash_buckets

Number of hash buckets for matching local reader and remote/local writer pairs.

property matching_writer_reader_pair_allocation

Allocation settings applied to matching local writer and remote/local reader pairs.

This property's getter returns a deep copy.

property matching_writer_reader_pair_hash_buckets

Number of hash buckets for matching local writer and remote/local reader pairs.

property max_endpoint_group_cumulative_characters

Maximum number of combined role_name characters allowed in all EndpointGroup in a Availability.

property max_endpoint_groups

Maximum number of EndpointGroup allowable in a Availability.

property max_gather_destinations

Maximum number of destinations per RTI Connext send.

property max_partition_cumulative_characters

Maximum number of combined characters allowable in all partition names in a Partition.

property max_partitions

Maximum number of partition name strings allowable in a Partition.

property outstanding_asynchronous_sample_allocation

Allocation settings applied to the maximum number of samples (from DataWriter) waiting to be asynchronously written.

This property's getter returns a deep copy.

property participant_property_list_max_length

Maximum number of properties associated with the DomainParticipant.

property participant_property_string_max_length

Maximum string length of the properties associated with the DomainParticipant.

property participant_user_data_max_length

Maximum length of user data in DomainParticipantQos and ParticipantBuiltinTopicData.

property publisher_group_data_max_length

Maximum length of group data in PublisherQos and PublicationBuiltinTopicData.

property query_condition_allocation

Allocation settings applied to query condition pool.

This property's getter returns a deep copy.

property read_condition_allocation

Allocation settings applied to read condition pool.

This property's getter returns a deep copy.

property reader_data_tag_list_max_length

Maximum number of data tags associated with a DataReader.

property reader_data_tag_string_max_length

Maximum string length of the data tags associated with a DataReader.

property reader_property_list_max_length

Maximum number of properties associated with a DataReader.

property reader_property_string_max_length

Maximum string length of the properties associated with a DataReader.

property reader_user_data_max_length

Maximum length of user data in DataReaderQos and SubscriptionBuiltinTopicData.

property remote_participant_allocation

Allocation settings applied to remote DomainParticipants.

This property's getter returns a deep copy.

property remote_participant_hash_buckets

Number of hash buckets for remote DomainParticipants.

property remote_reader_allocation

Allocation settings applied to remote DataReaders.

This property's getter returns a deep copy.

property remote_reader_hash_buckets

Number of hash buckets for remote DataReaders.

property remote_topic_query_allocation

Allocation settings applied to remote TopicQueries.

This property's getter returns a deep copy.

property remote_topic_query_hash_buckets

Number of hash buckets for remote TopicQueries.

property remote_writer_allocation

Allocation settings applied to remote DataWriters.

This property's getter returns a deep copy.

property remote_writer_hash_buckets

Number of hash buckets for remote DataWriters.

property serialized_type_object_dynamic_allocation_threshold

A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject.

property shmem_ref_transfer_mode_max_segments

Maximum number of segments created by all DataWriters belonging to a DomainParticipant.

property subscriber_group_data_max_length

Maximum length of group data in SubscriberQos and SubscriptionBuiltinTopicData.

property topic_data_max_length

Maximum length of topic data in TopicQos, TopicBuiltinTopicData, PublicationBuiltinTopicData and SubscriptionBuiltinTopicData.

property transport_info_list_max_length

Maximum number of installed transports to send and receive information about in ParticipantBuiltinTopicData.transport_info.

property type_code_max_serialized_length

Maximum size of serialized string for type code.

property type_object_max_deserialized_length

The maximum number of bytes that a deserialized TypeObject can consume.

property type_object_max_serialized_length

The maximum length, in bytes, that the buffer to serialize a TypeObject can consume.

property writer_data_tag_list_max_length

Maximum number of data tags associated with a DataWriter.

property writer_data_tag_string_max_length

Maximum string length of the data tags associated with a DataWriter.

property writer_property_list_max_length

Maximum number of properties associated with a DataWriter.

property writer_property_string_max_length

Maximum string length of the properties associated with a DataWriter.

property writer_user_data_max_length

Maximum length of user data in DataWriterQos and PublicationBuiltinTopicData.

class rti.connextdds.**DomainParticipantSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.DomainParticipantSeq, *arg0:* rti.connextdds.DomainParticipantSeq) → *rti.connextdds.DomainParticipantSeq*

__bool__(*self:* rti.connextdds.DomainParticipantSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.DomainParticipantSeq, *x:* rti.connextdds.DomainParticipant) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.DomainParticipantSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.DomainParticipantSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.DomainParticipantSeq, *arg0:* rti.connextdds.DomainParticipantSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.DomainParticipantSeq, s: slice) -> rti.connextdds.DomainParticipantSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.DomainParticipantSeq, arg0: int) -> rti.connextdds.DomainParticipant

__hash__ = None

__iadd__(*self:* rti.connextdds.DomainParticipantSeq, *arg0:* rti.connextdds.DomainParticipantSeq) → *rti.connextdds.DomainParticipantSeq*

__imul__(*self:* rti.connextdds.DomainParticipantSeq, *arg0: int*) → *rti.connextdds.DomainParticipantSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DomainParticipantSeq) -> None

2. __init__(self: rti.connextdds.DomainParticipantSeq, arg0: rti.connextdds.DomainParticipantSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.DomainParticipantSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.DomainParticipantSeq) → Iterator

__len__ (*self:* rti.connextdds.DomainParticipantSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.DomainParticipantSeq, *arg0: int*) → *rti.connextdds.DomainParticipantSeq*

__ne__ (*self:* rti.connextdds.DomainParticipantSeq, *arg0:* rti.connextdds.DomainParticipantSeq) → bool

__rmul__ (*self:* rti.connextdds.DomainParticipantSeq, *arg0: int*) → *rti.connextdds.DomainParticipantSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.DomainParticipantSeq, arg0: int, arg1: rti.connextdds.DomainParticipant) -> None

2. __setitem__(self: rti.connextdds.DomainParticipantSeq, arg0: slice, arg1: rti.connextdds.DomainParticipantSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.DomainParticipantSeq, *x:* rti.connextdds.DomainParticipant) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.DomainParticipantSeq) → None

Clear the contents

count (*self:* rti.connextdds.DomainParticipantSeq, *x:* rti.connextdds.DomainParticipant) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.DomainParticipantSeq, L: rti.connextdds.DomainParticipantSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.DomainParticipantSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.DomainParticipantSeq, *i: int*, *x:* rti.connextdds.DomainParticipant) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.DomainParticipantSeq) -> rti.connextdds.DomainParticipant

Remove and return the last item

2. pop(self: rti.connextdds.DomainParticipantSeq, i: int) -> rti.connextdds.DomainParticipant

Remove and return the item at index `i`

remove (*self:* rti.connextdds.DomainParticipantSeq, *x:* rti.connextdds.DomainParticipant) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

rti.connextdds.**DoubleSeq**

alias of *Float64Seq*

rti.connextdds.**DoubleType**

alias of *Float64Type*

class rti.connextdds.**Durability**

Bases: pybind11_object

AUTO_WRITER_DEPTH = 0

__eq__ (*self:* rti.connextdds.Durability, *arg0:* rti.connextdds.Durability) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Durability) -> None

Create an empty Durability QoS policy.

2. __init__(self: rti.connextdds.Durability, kind: rti.connextdds.DurabilityKind) -> None

Create a Durability QoS policy with the given kind.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Durability, *arg0:* rti.connextdds.Durability) → bool

Test for inequality.

property direct_communication

Get/set whether a DataReader should receive samples directly from a TRANSIENT or PERSISTENT DataWriter.

property kind

Get/set the Durability kind.

persistent = <rti.connextdds.Durability object>

property storage_settings

Configures durable writer history and durable reader state.

transient = <rti.connextdds.Durability object>

transient_local = <rti.connextdds.Durability object>

volatile = <rti.connextdds.Durability object>

property writer_depth

The number of samples a durable DataWriter will send to a late joining DataReader.

class rti.connextdds.**DurabilityKind**

Bases: pybind11_object

class DurabilityKind

Bases: pybind11_object

Members:

VOLATILE : [default] RTI Connext does not need to keep any samples of data instances on behalf of any DataReader that is unknown by the DataWriter at the time the instance is written.

In other words, RTI Connext will only attempt to provide the data to existing subscribers.

This option does not require RTI Persistence Service.

TRANSIENT_LOCAL : RTI Connext will attempt to keep some samples so that they can be delivered to any potential late-joining DataReader.

Which particular samples are kept depends on other QoS such as History and ResourceLimits. RTI Connext is only required to keep the data in memory of the DataWriter that wrote the data.

Data is not required to survive the DataWriter.

For this setting to be effective, you must also set the Reliability.kind to ReliabilityKind.RELIABLE.

This option does not require RTI Persistence Service.

TRANSIENT : RTI Connext will attempt to keep some samples so that they can be delivered to any potential late-joining DataReader.

Which particular samples are kept depends on other QoS such as History and ResourceLimits. RTI Connext is only required to keep the data in memory and not in permanent storage.

Data is not tied to the lifecycle of the DataWriter.

Data will survive the DataWriter.

This option requires RTI Persistence Service.

PERSISTENT : Data is kept on permanent storage, so that they can outlive a system session.

This option requires RTI Persistence Service.

PERSISTENT = <DurabilityKind.PERSISTENT: 3>

TRANSIENT = <DurabilityKind.TRANSIENT: 2>

TRANSIENT_LOCAL = <DurabilityKind.TRANSIENT_LOCAL: 1>

VOLATILE = <DurabilityKind.VOLATILE: 0>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.DurabilityKind.DurabilityKind) → int

__init__(*self:* rti.connextdds.DurabilityKind.DurabilityKind, *value: int*) → None

__int__(*self:* rti.connextdds.DurabilityKind.DurabilityKind) → int

__members__ = {'PERSISTENT': <DurabilityKind.PERSISTENT: 3>, 'TRANSIENT': <DurabilityKind.TRANSIENT: 2>, 'TRANSIENT_LOCAL': <DurabilityKind.TRANSIENT_LOCAL: 1>, 'VOLATILE': <DurabilityKind.VOLATILE: 0>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.DurabilityKind.DurabilityKind, *state: int*) → None

__str__()
name(self: handle) -> str

property name

property value

PERSISTENT = <DurabilityKind.PERSISTENT: 3>

TRANSIENT = <DurabilityKind.TRANSIENT: 2>

TRANSIENT_LOCAL = <DurabilityKind.TRANSIENT_LOCAL: 1>

VOLATILE = <DurabilityKind.VOLATILE: 0>

__eq__(*self:* rti.connextdds.DurabilityKind, *arg0:* rti.connextdds.DurabilityKind) → bool
Apply operator to underlying enumerated values.

__ge__(*self:* rti.connextdds.DurabilityKind, *arg0:* rti.connextdds.DurabilityKind) → bool
Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.DurabilityKind, *arg0:* rti.connextdds.DurabilityKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DurabilityKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.DurabilityKind, arg0: rti.connextdds.DurabilityKind.DurabilityKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.DurabilityKind) → *rti.connextdds.DurabilityKind.DurabilityKind*

Int conversion.

__le__ (*self:* rti.connextdds.DurabilityKind, *arg0:* rti.connextdds.DurabilityKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.DurabilityKind, *arg0:* rti.connextdds.DurabilityKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DurabilityKind, *arg0:* rti.connextdds.DurabilityKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.DurabilityKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**DurabilityService**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.DurabilityService, *arg0:* rti.connextdds.DurabilityService) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DurabilityService) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.DurabilityService, service_cleanup_delay: rti.connextdds.Duration, history_kind: rti.connextdds.HistoryKind, history_depth: int, max_samples: int, max_instances: int, max_samples_per_instance: int) -> None

Creates an instance with all the parameters set.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DurabilityService, *arg0:* rti.connextdds.DurabilityService) → bool

Test for inequality.

property history_depth

Controls when the service is able to remove all information regarding a data instances.

property history_kind

Controls when the service is able to remove all information regarding a data instances.

property max_instances

Controls when the service is able to remove all information regarding a data instances.

property max_samples

Controls when the service is able to remove all information regarding a data instances.

property max_samples_per_instance

Controls when the service is able to remove all information regarding a data instances.

property service_cleanup_delay

Controls when the service is able to remove all information regarding a data instances.

class rti.connextdds.**Duration**

Bases: pybind11_object

__add__(*self:* rti.connextdds.Duration, *arg0:* rti.connextdds.Duration) → *rti.connextdds.Duration*

Add two Duration objects.

__eq__(*self:* rti.connextdds.Duration, *arg0:* rti.connextdds.Duration) → bool

Compare Duration objects for equality.

__float__(*self:* rti.connextdds.Duration) → float

Floating point value of Duration in seconds.

__ge__(*self:* rti.connextdds.Duration, *arg0:* rti.connextdds.Duration) → bool

Greater-than-or-equal comparison for Duration objects.

__gt__(*self:* rti.connextdds.Duration, *arg0:* rti.connextdds.Duration) → bool

Greater than comparison for Duration objects.

__hash__ = None

__iadd__(*self:* rti.connextdds.Duration, *arg0:* rti.connextdds.Duration) → *rti.connextdds.Duration*

Add a Duration to another.

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Duration) -> None

Create a Duration of 0 seconds.

2. __init__(self: rti.connextdds.Duration, seconds: int, nanoseconds: int) -> None

Create a Duration of the specified seconds and nanoseconds.

3. __init__(self: rti.connextdds.Duration, duration: datetime.timedelta) -> None

Create a Duration from a datetime.timedelta.

4. __init__(self: rti.connextdds.Duration, float_duration: float) -> None

Create a Duration from a floating point duration in seconds.

5. __init__(self: rti.connextdds.Duration, seconds: int) -> None

Create a Duration from an integer number of seconds.

6. __init__(self: rti.connextdds.Duration, time_tuple: Tuple[int, int]) -> None

Create a Duration from a tuple of (sec, nanosec).

7. __init__(self: rti.connextdds.Duration, other: rti.connextdds.Duration) -> None

Create a copy of a Duration.

__int__ (*self:* rti.connextdds.Duration) → int
Integer value of Duration in microseconds.

__isub__ (*self:* rti.connextdds.Duration, *arg0:* rti.connextdds.Duration) → *rti.connextdds.Duration*
Subtract a Duration from another.

__le__ (*self:* rti.connextdds.Duration, *arg0:* rti.connextdds.Duration) → bool
Less-than-or-equal comparison for Duration object.

__lt__ (*self:* rti.connextdds.Duration, *arg0:* rti.connextdds.Duration) → bool
Less than comparison for Duration objects.

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.Duration, *arg0: int*) → *rti.connextdds.Duration*
Multiply a Duration by an unsigned int.

__ne__ (*self:* rti.connextdds.Duration, *arg0:* rti.connextdds.Duration) → bool
Determine if Duration objects are unequal.

__rmul__ (*self:* rti.connextdds.Duration, *arg0: int*) → *rti.connextdds.Duration*
Multiply a Duration by an unsigned int.

__sub__ (*self:* rti.connextdds.Duration, *arg0:* rti.connextdds.Duration) → *rti.connextdds.Duration*
Subtract two Duration objects.

__truediv__ (*self:* rti.connextdds.Duration, *arg0: int*) → *rti.connextdds.Duration*
Divide a Duration by an unsigned int.

automatic = <rti.connextdds.Duration object>

compare (*self:* rti.connextdds.Duration, *other:* rti.connextdds.Duration) → int

Compare this Duration to another. Returns -1 if this Duration is less, 0 if they are equal, and 1 if this Duration is greate

static from_microseconds (*microseconds: int*) → *rti.connextdds.Duration*

Get a Duration from microseconds.

static from_milliseconds (*milliseconds: int*) → *rti.connextdds.Duration*

Get a Duration from milliseconds.

static from_seconds (*seconds: float*) → *rti.connextdds.Duration*

Get a Duration from seconds

infinite = <rti.connextdds.Duration object>

property nanosec

Get/set the number of nanoseconds in the Duration.

property sec

Get/set the number of seconds in the Duration.

to_microseconds (*self:* rti.connextdds.Duration) → int

Returns Duration in microseconds.

to_milliseconds (*self:* rti.connextdds.Duration) → int

Returns Duration in milliseconds.

to_seconds (*self:* rti.connextdds.Duration) → float

Return the Duration in seconds.

to_timedelta (*self:* rti.connextdds.Duration) → datetime.timedelta

Return the datetime.timedelta conversion of this Duration.

zero = <rti.connextdds.Duration object>

class rti.connextdds.**DynamicData**

Bases: pybind11_object

class ContentFilter

Bases: *ContentFilterBase*

__init__ (*self:* rti.connextdds.DynamicData.ContentFilter) → None

__module__ = 'rti.connextdds'

compile (*self:* rti.connextdds.DynamicData.ContentFilter, *expression: str*, *parameters:* rti.connextdds.StringSeq, *type_code: Optional[*rti.connextdds.DynamicType*]*, *type_class_name: str*, *old_compile_data: Optional[object]*) → Optional[object]

Compile an instance of the content filter according to the filter expression and parameters of the given data type.

evaluate (*self:* rti.connextdds.DynamicData.ContentFilter, *compile_data: Optional[object]*, *sample:* rti.connextdds.DynamicData, *meta_data:* rti.connextdds.FilterSampleInfo) → bool

Evaluate whether the sample is passing the filter or not according to the sample content.

finalize (*self:* rti.connextdds.DynamicData.ContentFilter, *compile_data: Optional[object]*) → None

A previously compiled instance of the content filter is no longer in use and resources can now be cleaned up.

class ContentFilteredTopic

Bases: *ITopicDescription*, *IAnyTopic*

__eq__ (*self:* rti.connextdds.DynamicData.ContentFilteredTopic, *arg0:* rti.connextdds.DynamicData.ContentFilteredTopic) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DynamicData.ContentFilteredTopic, topic: rti.connextdds.DynamicData.Topic, name: str, contentfilter: rti.connextdds.Filter) -> None

Create a ContentFilteredTopic with a name and Filter.

2. __init__(self: rti.connextdds.DynamicData.ContentFilteredTopic, topic_description: rti.connextdds.DynamicData.ITopicDescription) -> None

Cast a TopicDescription to a ContentFilteredTopic.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DynamicData.ContentFilteredTopic, *arg0:* rti.connextdds.DynamicData.ContentFilteredTopic) → bool

Test for inequality.

append_to_expression_parameter (*self:* rti.connextdds.DynamicData.ContentFilteredTopic, *index: int*, *extension: str*) → None

Append the extension to the end of parameter at the provided index, separated by a comma.

property filter_expression

Get the filter expression

property filter_parameters

Get/set the filter parameters.

static find (*participant:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.DynamicData.ContentFilteredTopic*]

Look up a ContentFilteredTopic by its name in the DomainParticipant.

remove_from_expression_parameter (*self:* rti.connextdds.DynamicData.ContentFilteredTopic, *index: int*, *remove_term: str*) → None

Removes the specified term from the parameter at the provided index.

set_filter (*self:* rti.connextdds.DynamicData.ContentFilteredTopic, *arg0:* rti.connextdds.Filter) → None

Set the filter.

property topic

Get the underlying Topic.

class ContentFilteredTopicSeq

Bases: pybind11_object

__add__ (*self:* rti.connextdds.DynamicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.DynamicData.ContentFilteredTopicSeq) → *rti.connextdds.DynamicData.ContentFilteredTopicSeq*

__bool__ (*self:* rti.connextdds.DynamicData.ContentFilteredTopicSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.DynamicData.ContentFilteredTopicSeq, *x:* rti.connextdds.DynamicData.ContentFilteredTopic) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.DynamicData.ContentFilteredTopicSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.DynamicData.ContentFilteredTopicSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.DynamicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.DynamicData.ContentFilteredTopicSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.DynamicData.ContentFilteredTopicSeq, s: slice) -> rti.connextdds.DynamicData.ContentFilteredTopicSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.DynamicData.ContentFilteredTopicSeq, arg0: int) -> rti.connextdds.DynamicData.ContentFilteredTopic

__hash__ = None

__iadd__ (*self:* rti.connextdds.DynamicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.DynamicData.ContentFilteredTopicSeq) → *rti.connextdds.DynamicData.ContentFilteredTopicSeq*

__imul__ (*self:* rti.connextdds.DynamicData.ContentFilteredTopicSeq, *arg0: int*) → *rti.connextdds.DynamicData.ContentFilteredTopicSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DynamicData.ContentFilteredTopicSeq) -> None
2. __init__(self: rti.connextdds.DynamicData.ContentFilteredTopicSeq, arg0: rti.connextdds.DynamicData.ContentFilteredTopicSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.DynamicData.ContentFilteredTopicSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.DynamicData.ContentFilteredTopicSeq) → Iterator

__len__ (*self:* rti.connextdds.DynamicData.ContentFilteredTopicSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.DynamicData.ContentFilteredTopicSeq, *arg0: int*) → *rti.connextdds.DynamicData.ContentFilteredTopicSeq*

__ne__ (*self:* rti.connextdds.DynamicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.DynamicData.ContentFilteredTopicSeq) → bool

__rmul__ (*self:* rti.connextdds.DynamicData.ContentFilteredTopicSeq, *arg0: int*) → *rti.connextdds.DynamicData.ContentFilteredTopicSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.DynamicData.ContentFilteredTopicSeq, arg0: int, arg1: rti.connextdds.DynamicData.ContentFilteredTopic) -> None
2. __setitem__(self: rti.connextdds.DynamicData.ContentFilteredTopicSeq, arg0: slice, arg1: rti.connextdds.DynamicData.ContentFilteredTopicSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.DynamicData.ContentFilteredTopicSeq, *x:* rti.connextdds.DynamicData.ContentFilteredTopic) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.DynamicData.ContentFilteredTopicSeq) → None

Clear the contents

count (*self:* rti.connextdds.DynamicData.ContentFilteredTopicSeq, *x:* rti.connextdds.DynamicData.ContentFilteredTopic) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.DynamicData.ContentFilteredTopicSeq, L: rti.connextdds.DynamicData.ContentFilteredTopicSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.DynamicData.ContentFilteredTopicSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.DynamicData.ContentFilteredTopicSeq, *i: int*, *x:* rti.connextdds.DynamicData.ContentFilteredTopic) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.DynamicData.ContentFilteredTopicSeq) -> rti.connextdds.DynamicData.ContentFilteredTopic

Remove and return the last item

2. pop(self: rti.connextdds.DynamicData.ContentFilteredTopicSeq, i: int) -> rti.connextdds.DynamicData.ContentFilteredTopic

Remove and return the item at index `i`

remove (*self:* rti.connextdds.DynamicData.ContentFilteredTopicSeq, *x:* rti.connextdds.DynamicData.ContentFilteredTopic) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class DataReader

Bases: *IDataReader*

class Selector

Bases: pybind11_object

__init__ (*self:* rti.connextdds.DynamicData.DataReader.Selector, *datareader:* rti.connextdds.DynamicData.DataReader) → None

Create a Selector for a DataReader to read/take based on selected conditions

__module__ = 'rti.connextdds'

condition (*self:* rti.connextdds.DynamicData.DataReader.Selector, *condition:* rti.connextdds.IReadCondition) → *rti.connextdds.DynamicData.DataReader.Selector*

Select samples based on a ReadCondition.

content (*self:* rti.connextdds.DynamicData.DataReader.Selector, *query:* rti.connextdds.Query) → *rti.connextdds.DynamicData.DataReader.Selector*

Select samples based on a Query.

instance (*self:* rti.connextdds.DynamicData.DataReader.Selector, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.DynamicData.DataReader.Selector*

Select a specific instance to read/take.

max_samples (*self:* rti.connextdds.DynamicData.DataReader.Selector, *max: int*) → *rti.connextdds.DynamicData.DataReader.Selector*

Limit the number of samples read/taken by the Select.

next_instance (*self:* rti.connextdds.DynamicData.DataReader.Selector, *previous:* rti.connextdds.InstanceHandle) → *rti.connextdds.DynamicData.DataReader.Selector*

Select the instance after the specified instance to read/take.

read (*self:* rti.connextdds.DynamicData.DataReader.Selector) → list

Read copies of available samples (data and info) based on the Selector settings.

read_data (*self:* rti.connextdds.DynamicData.DataReader.Selector) → list

Read copies of available valid data based on the Selector settings.

read_loaned (*self:* rti.connextdds.DynamicData.DataReader.Selector) → *rti.connextdds.DynamicData.LoanedSamples*

Take available samples (data and info) based on the Selector settings and return them in a loaned container.

state (*self:* rti.connextdds.DynamicData.DataReader.Selector, *state:* rti.connextdds.DataState) → *rti.connextdds.DynamicData.DataReader.Selector*

Select samples with a specified data state.

take (*self:* rti.connextdds.DynamicData.DataReader.Selector) → list

Take copies of available samples (data and info) based on the Selector settings.

take_data (*self:* rti.connextdds.DynamicData.DataReader.Selector) → list

Take copies of available valid data based on the Selector settings.

take_loaned (*self:* rti.connextdds.DynamicData.DataReader.Selector) → *rti.connextdds.DynamicData.LoanedSamples*

Read available samples (data and info) based on the Selector settings and return them in a loaned container.

__enter__ (*self:* rti.connextdds.DynamicData.DataReader) → *rti.connextdds.DynamicData.DataReader*

Enter a context for this DataReader, to be cleaned up on exiting context

__eq__ (*self:* rti.connextdds.DynamicData.DataReader, *arg0:* rti.connextdds.DynamicData.DataReader) → bool

Test for equality.

__exit__ (*self:* rti.connextdds.DynamicData.DataReader, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for this DataReader, cleaning up resources.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DynamicData.DataReader, topic: rti.connextdds.DynamicData.Topic) -> None

Create a DataReader in the implicit subscriber with default QoS.

2. __init__(self: rti.connextdds.DynamicData.DataReader, topic: rti.connextdds.DynamicData.Topic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.DynamicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader in the implicit subscriber with specific QoS and a listener.

3. __init__(self: rti.connextdds.DynamicData.DataReader, cft: rti.connextdds.DynamicData.ContentFilteredTopic) -> None

Create a DataReader with a ContentFilteredTopic in the implicit subscriber with default QoS.

4. __init__(self: rti.connextdds.DynamicData.DataReader, cft: rti.connextdds.DynamicData.ContentFilteredTopic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.DynamicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader with a ContentFilteredTopic in the implicit subscriber with specific QoS.

5. __init__(self: rti.connextdds.DynamicData.DataReader, subscriber: rti.connextdds.Subscriber, topic: rti.connextdds.DynamicData.Topic) -> None

Create a DataReader.

6. __init__(self: rti.connextdds.DynamicData.DataReader, subscriber: rti.connextdds.Subscriber, topic: rti.connextdds.DynamicData.Topic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.DynamicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader in a subscriber with specific QoS and a listener.

7. __init__(self: rti.connextdds.DynamicData.DataReader, subscriber: rti.connextdds.Subscriber, cft: rti.connextdds.DynamicData.ContentFilteredTopic) -> None

Create a DataReader with a ContentFilteredTopic in a subscriber with default QoS.

8. __init__(self: rti.connextdds.DynamicData.DataReader, subscriber: rti.connextdds.Subscriber, cft: rti.connextdds.DynamicData.ContentFilteredTopic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.DynamicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader with a ContentFilteredTopic in a subscriber with specific QoS.

9. __init__(self: rti.connextdds.DynamicData.DataReader, reader: rti.connextdds.IAnyDataReader) -> None

Get a typed DataReader from an AnyDataReader.

10. __init__(self: rti.connextdds.DynamicData.DataReader, entity: rti.connextdds.IEntity) -> None

Get a typed DataReader from an Entity.

__lshift__ (*self:* rti.connextdds.DynamicData.DataReader, *arg0:* rti.connextdds.DataReaderQos) → *rti.connextdds.DynamicData.DataReader*

Set the DataReaderQos for this DataReader.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DynamicData.DataReader, *arg0:* rti.connextdds.DynamicData.DataReader) → bool

Test for inequality.

__rshift__(*self:* rti.connextdds.DynamicData.DataReader, *arg0:* rti.connextdds.DataReaderQos) → *rti.connextdds.DynamicData.DataReader*

Get the DataReaderQos from this DataReader

acknowledge_all(**args*, ***kwargs*)

Overloaded function.

1. acknowledge_all(self: rti.connextdds.DynamicData.DataReader) -> None

Acknowledge all previously accessed samples.

2. acknowledge_all(self: rti.connextdds.DynamicData.DataReader, arg0: rti.connextdds.AckResponseData) -> None

Acknowledge all previously accessed samples.

acknowledge_sample(**args*, ***kwargs*)

Overloaded function.

1. acknowledge_sample(self: rti.connextdds.DynamicData.DataReader, sample_info: rti.connextdds.SampleInfo) -> None

Acknowledge a single sample.

2. acknowledge_sample(self: rti.connextdds.DynamicData.DataReader, sample_info: rti.connextdds.SampleInfo, ack_response_data: rti.connextdds.AckResponseData) -> None

Acknowledge a single sample with ack response data.

close(*self:* rti.connextdds.DynamicData.DataReader) → None

Close this DataReader.

property datareader_cache_status

Get the DataReaderCacheStatus for the DataReader.

property datareader_protocol_status

Get the DataReaderProtocolStatus for the DataReader.

property default_filter_state

Returns the filter state for the read/take operations.

static find_all_by_topic(*subscriber:* rti.connextdds.Subscriber, *topic_name: str*) → *rti.connextdds.DynamicData.DataReaderSeq*

Retrieve all DataReaders for the given topic name in the subscriber.

static find_by_name(**args*, ***kwargs*)

Overloaded function.

1. find_by_name(participant: rti.connextdds.DomainParticipant, name: str) -> Optional[rti.connextdds.DynamicData.DataReader]

Find DataReader in DomainParticipant with the DataReader's name, returning the first found.

2. find_by_name(subscriber: rti.connextdds.Subscriber, name: str) -> Optional[rti.connextdds.DynamicData.DataReader]

Find DataReader in Subscriber with the DataReader's name, returning the first found.

static find_by_topic(*subscriber:* rti.connextdds.Subscriber, *name: str*) → Optional[*rti.connextdds.DynamicData.DataReader*]

Find DataReader in Subscriber with a topic name, returning the first found.

is_matched_publication_alive(*self:* rti.connextdds.DynamicData.DataReader, *arg0:* rti.connextdds.InstanceHandle) → bool

Check if a matched publication is alive.

key_value(*self:* rti.connextdds.DynamicData.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.DynamicData*

Retrieve the instance key that corresponds to an instance handle.

property listener

Gets or sets the listener with StatusMask.ALL

property liveliness_changed_status

Get the LivelinessChangedStatus for this DataReader.

lookup_instance(*self:* rti.connextdds.DynamicData.DataReader, *key_holder:* rti.connextdds.DynamicData) → *rti.connextdds.InstanceHandle*

Retrieve the instance handle that corresponds to an instance key_holder

matched_publication_data(*self:* rti.connextdds.DynamicData.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.PublicationBuiltinTopicData*

Get the PublicationBuiltinTopicData for a publication matched to this DataReader.

matched_publication_datareader_protocol_status(*self:* rti.connextdds.DynamicData.DataReader, *publication_handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.DataReaderProtocolStatus*

Get the DataReaderProtocolStatus for the DataReader based on the matched publication handle.

matched_publication_participant_data(*self:* rti.connextdds.DynamicData.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ParticipantBuiltinTopicData*

Get the ParticipantBuiltinTopicData for a publication matched to this DataReader.

property matched_publications

Get a copy of the list of the currently matched publication handles.

property qos

The DataReaderQos for this DataReader.

This property's getter returns a deep copy.

read (*self:* rti.connextdds.DynamicData.DataReader) → list

Read copies of all available samples (data and info).

read_data (*self:* rti.connextdds.DynamicData.DataReader) → list

Read copies of all available valid data.

read_loaned (*self:* rti.connextdds.DynamicData.DataReader) → *rti.connextdds.DynamicData.LoanedSamples*

Read all available samples (data and info) and return them in a loaned container.

property requested_deadline_missed_status

Get the RequestedDeadlineMissed status for the DataReader

property requested_incompatible_qos_status

Get the RequestedIncompatibleQosStatus for the DataReader.

property sample_lost_status

Get the SampleLostStatus for the DataReader.

property sample_rejected_status

Get the SampleRejectedStatus for the DataReader.

select (*self:* rti.connextdds.DynamicData.DataReader) → dds::sub::DataReader<rti::core::xtypes::DynamicDataImpl, rti::sub::DataReaderImpl>::Selector

Get a Selector to perform complex data selections, such as per-instance selection, content, and status filtering.

set_listener (*self:* rti.connextdds.DynamicData.DataReader, *listener:* rti.connextdds.DynamicData.DataReaderListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and associated event mask.

property subscriber

Returns the parent Subscriber of the DataReader.

property subscription_matched_status

Get the SubscriptionMatchedStatus for the DataReader.

take (*self:* rti.connextdds.DynamicData.DataReader) → list

Take copies of all available samples (data and info).

take_async (*condition: Optional[*ReadCondition*] = None*)

take_data (*self:* rti.connextdds.DynamicData.DataReader) → list

Take copies of all available valid data.

take_data_async (*condition: Optional[*ReadCondition*] = None*)

take_loaned (*self:* rti.connextdds.DynamicData.DataReader) → *rti.connextdds.DynamicData.LoanedSamples*

Take all available samples (data and info) and return them in a loaned container.

property topic_description

Returns the TopicDescription associated with the DataReader.

property topic_name

Get the topic name associated with this DataReader.

property type_name

Get the type name associated with this DataReader.

wait_for_historical_data(*self:* rti.connextdds.DynamicData.DataReader, *max_wait:* rti.connextdds.Duration) → None

Waits until all "historical" data is received for DataReaders that have a non-VOLATILE Durability Qos kind.

wait_for_historical_data_async(*self:* rti.connextdds.DynamicData.DataReader, *max_wait:* rti.connextdds.Duration) → object

Waits until all "historical" data is received for DataReaders that have a non-VOLATILE Durability Qos kind. This call is awaitable and only for use with asyncio.

class DataReaderListener

Bases: `pybind11_object`

__init__(*self:* rti.connextdds.DynamicData.DataReaderListener) → None

__module__ = 'rti.connextdds'

on_data_available(*self:* rti.connextdds.DynamicData.DataReaderListener, *arg0:* rti.connextdds.DynamicData.DataReader) → None

Data available callback.

on_liveliness_changed(*self:* rti.connextdds.DynamicData.DataReaderListener, *arg0:* rti.connextdds.DynamicData.DataReader, *arg1:* rti.connextdds.LivelinessChangedStatus) → None

Liveliness changed callback.

on_requested_deadline_missed(*self:* rti.connextdds.DynamicData.DataReaderListener, *arg0:* rti.connextdds.DynamicData.DataReader, *arg1:* rti.connextdds.RequestedDeadlineMissedStatus) → None

Requested deadline missed callback.

on_requested_incompatible_qos(*self:* rti.connextdds.DynamicData.DataReaderListener, *arg0:* rti.connextdds.DynamicData.DataReader, *arg1:* rti.connextdds.RequestedIncompatibleQosStatus) → None

Requested incompatible QoS callback.

on_sample_lost (*self:* rti.connextdds.DynamicData.DataReaderListener, *arg0:* rti.connextdds.DynamicData.DataReader, *arg1:* rti.connextdds.SampleLostStatus) → None

Sample lost callback.

on_sample_rejected (*self:* rti.connextdds.DynamicData.DataReaderListener, *arg0:* rti.connextdds.DynamicData.DataReader, *arg1:* rti.connextdds.SampleRejectedStatus) → None

Sample rejected callback.

on_subscription_matched (*self:* rti.connextdds.DynamicData.DataReaderListener, *arg0:* rti.connextdds.DynamicData.DataReader, *arg1:* rti.connextdds.SubscriptionMatchedStatus) → None

Subscription matched callback.

class DataReaderSeq

Bases: `pybind11_object`

__add__ (*self:* rti.connextdds.DynamicData.DataReaderSeq, *arg0:* rti.connextdds.DynamicData.DataReaderSeq) → *rti.connextdds.DynamicData.DataReaderSeq*

__bool__ (*self:* rti.connextdds.DynamicData.DataReaderSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.DynamicData.DataReaderSeq, *x:* rti.connextdds.DynamicData.DataReader) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.DynamicData.DataReaderSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.DynamicData.DataReaderSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.DynamicData.DataReaderSeq, *arg0:* rti.connextdds.DynamicData.DataReaderSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.DynamicData.DataReaderSeq, s: slice) -> rti.connextdds.DynamicData.DataReaderSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.DynamicData.DataReaderSeq, arg0: int) -> rti.connextdds.DynamicData.DataReader

__hash__ = None

__iadd__ (*self:* rti.connextdds.DynamicData.DataReaderSeq, *arg0:* rti.connextdds.DynamicData.DataReaderSeq) → *rti.connextdds.DynamicData.DataReaderSeq*

__imul__ (*self:* rti.connextdds.DynamicData.DataReaderSeq, *arg0: int*) → *rti.connextdds.DynamicData.DataReaderSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DynamicData.DataReaderSeq) -> None
2. __init__(self: rti.connextdds.DynamicData.DataReaderSeq, arg0: rti.connextdds.DynamicData.DataReaderSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.DynamicData.DataReaderSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.DynamicData.DataReaderSeq) → Iterator

__len__ (*self:* rti.connextdds.DynamicData.DataReaderSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.DynamicData.DataReaderSeq, *arg0: int*) → *rti.connextdds.DynamicData.DataReaderSeq*

__ne__ (*self:* rti.connextdds.DynamicData.DataReaderSeq, *arg0:* rti.connextdds.DynamicData.DataReaderSeq) → bool

__rmul__ (*self:* rti.connextdds.DynamicData.DataReaderSeq, *arg0: int*) → *rti.connextdds.DynamicData.DataReaderSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.DynamicData.DataReaderSeq, arg0: int, arg1: rti.connextdds.DynamicData.DataReader) -> None
2. __setitem__(self: rti.connextdds.DynamicData.DataReaderSeq, arg0: slice, arg1: rti.connextdds.DynamicData.DataReaderSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.DynamicData.DataReaderSeq, *x:* rti.connextdds.DynamicData.DataReader) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.DynamicData.DataReaderSeq) → None

Clear the contents

count (*self:* rti.connextdds.DynamicData.DataReaderSeq, *x:* rti.connextdds.DynamicData.DataReader) → int

Return the number of times x appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.DynamicData.DataReaderSeq, L: rti.connextdds.DynamicData.DataReaderSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.DynamicData.DataReaderSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert(*self:* rti.connextdds.DynamicData.DataReaderSeq, *i: int*, *x:* rti.connextdds.DynamicData.DataReader) → None

Insert an item at a given position.

pop(**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.DynamicData.DataReaderSeq) -> rti.connextdds.DynamicData.DataReader

Remove and return the last item

2. pop(self: rti.connextdds.DynamicData.DataReaderSeq, i: int) -> rti.connextdds.DynamicData.DataReader

Remove and return the item at index `i`

remove(*self:* rti.connextdds.DynamicData.DataReaderSeq, *x:* rti.connextdds.DynamicData.DataReader) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class DataWriter

Bases: *IEntity*, *IAnyDataWriter*

__enter__(*self:* rti.connextdds.DynamicData.DataWriter) → *rti.connextdds.DynamicData.DataWriter*

Enter a context for this DataWriter, to be cleaned up on exiting context

__eq__(*self:* rti.connextdds.DynamicData.DataWriter, *arg0:* rti.connextdds.DynamicData.DataWriter) → bool

Test for equality.

__exit__(*self:* rti.connextdds.DynamicData.DataWriter, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for this DataWriter, cleaning up resources.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DynamicData.DataWriter, topic: rti.connextdds.DynamicData.Topic) -> None

Creates a DataWriter in the implicit publisher with default QoS.

2. __init__(self: rti.connextdds.DynamicData.DataWriter, topic: rti.connextdds.DynamicData.Topic, qos: rti.connextdds.DataWriterQos, listener: rti.connextdds.Dynamic-

Data.DataWriterListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Creates a DataWriter in the implicit publisher with specific QoS and optionally a listener.

3. __init__(self: rti.connextdds.DynamicData.DataWriter, pub: rti.connextdds.Publisher, topic: rti.connextdds.DynamicData.Topic) -> None

Creates a DataWriter in a publisher with default QoS.

4. __init__(self: rti.connextdds.DynamicData.DataWriter, pub: rti.connextdds.Publisher, topic: rti.connextdds.DynamicData.Topic, qos: rti.connextdds.DataWriterQos, listener: rti.connextdds.DynamicData.DataWriterListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Creates a DataWriter in a publisher with specific QoS and optionally a listener.

5. __init__(self: rti.connextdds.DynamicData.DataWriter, writer: rti.connextdds.IAnyDataWriter) -> None

Create a typed DataWriter from an AnyDataWriter.

6. __init__(self: rti.connextdds.DynamicData.DataWriter, entity: rti.connextdds.IEntity) -> None

Create a typed DataWriter from an Entity.

__lshift__(**args*, ***kwargs*)

Overloaded function.

1. __lshift__(self: rti.connextdds.DynamicData.DataWriter, arg0: rti.connextdds.DataWriterQos) -> rti.connextdds.DynamicData.DataWriter

Sets the DataWriterQos.

2. __lshift__(self: rti.connextdds.DynamicData.DataWriter, arg0: Tuple[rti.connextdds.DynamicData, rti.connextdds.Time]) -> rti.connextdds.DynamicData.DataWriter

Writes a paired sample with a timestamp.

3. __lshift__(self: rti.connextdds.DynamicData.DataWriter, arg0: Tuple[rti.connextdds.DynamicData, rti.connextdds.InstanceHandle]) -> rti.connextdds.DynamicData.DataWriter

Writes a paired sample with an instance handle.

4. __lshift__(self: rti.connextdds.DynamicData.DataWriter, arg0: rti.connextdds.DynamicDataTimestampedSeq) -> rti.connextdds.DynamicData.DataWriter

Writes a sequence of pairs of samples with timestamps.

5. __lshift__(self: rti.connextdds.DynamicData.DataWriter, arg0: rti.connextdds.DynamicDataSeq) -> rti.connextdds.DynamicData.DataWriter

Writes a sequence of samples.

6. __lshift__(self: rti.connextdds.DynamicData.DataWriter, arg0: rti.connextdds.DynamicData) -> rti.connextdds.DynamicData.DataWriter

Writes a sample.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DynamicData.DataWriter, *arg0:* rti.connextdds.DynamicData.DataWriter) → bool

Test for inequality.

__rshift__(*self:* rti.connextdds.DynamicData.DataWriter, *arg0:* rti.connextdds.DataWriterQos) → *rti.connextdds.DynamicData.DataWriter*

Get the DataWriterQos.

assert_liveliness (*self:* rti.connextdds.DynamicData.DataWriter) → None

Manually asserts the liveliness of the DataWriter.

close (*self:* rti.connextdds.DynamicData.DataWriter) → None

Close this DataWriter.

create_data (*self:* rti.connextdds.DynamicData.DataWriter) → *rti.connextdds.DynamicData*

Create data of the writer's associated type and initialize it.

property datawriter_cache_status

Get a copy of the cache status for this writer.

property datawriter_protocol_status

Get a copy of the protocol status for this writer.

dispose_instance (**args*, ***kwargs*)

Overloaded function.

1. dispose_instance(self: rti.connextdds.DynamicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.DynamicData.DataWriter

Dispose an instance.

2. dispose_instance(self: rti.connextdds.DynamicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> rti.connextdds.DynamicData.DataWriter

Dispose an instance with a timestamp.

3. dispose_instance(self: rti.connextdds.DynamicData.DataWriter, params: rti.connextdds.WriteParams) -> None

Dispose an instance with params.

dispose_instance_async (**args*, ***kwargs*)

Overloaded function.

1. dispose_instance_async(self: rti.connextdds.DynamicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> object

Dispose an instance.

2. dispose_instance_async(self: rti.connextdds.DynamicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> object

Dispose an instance with a timestamp.

3. dispose_instance_async(self: rti.connextdds.DynamicData.DataWriter, key_holder: rti.connextdds.DynamicData) -> object

Dispose the instance associated with key_holder.

4. dispose_instance_async(self: rti.connextdds.DynamicData.DataWriter, key_holder: rti.connextdds.DynamicData, timestamp: rti.connextdds.Time) -> object

Dispose the instance associated with key_holder using a timestamp

5. dispose_instance_async(self: rti.connextdds.DynamicData.DataWriter, params: rti.connextdds.WriteParams) -> object

Dispose an instance with params.

static find_all_by_topic(*publisher:* rti.connextdds.Publisher, *topic_name: str*) → *rti.connextdds.DynamicData.DataWriterSeq*

Retrieve all DataWriters for the given topic name in the publisher.

static find_by_name(**args*, ***kwargs*)

Overloaded function.

1. find_by_name(participant: rti.connextdds.DomainParticipant, name: str) -> Optional[rti.connextdds.DynamicData.DataWriter]

Find DataWriter in DomainParticipant with the provided name, returning the first found.

2. find_by_name(publisher: rti.connextdds.Publisher, name: str) -> Optional[rti.connextdds.DynamicData.DataWriter]

Find DataWriter in Publisher with the DataReader's name, returning the first found.

static find_by_topic(*publisher:* rti.connextdds.Publisher, *name: str*) → Optional[*rti.connextdds.DynamicData.DataWriter*]

Find DataWriter in publisher with a topic name, returning the first found.

flush(*self:* rti.connextdds.DynamicData.DataWriter) → None

Flushes the batch in progress in the context of thecalling thread.

is_matched_subscription_active(*self:* rti.connextdds.DynamicData.DataWriter, *arg0:* rti.connextdds.InstanceHandle) → bool

A boolean indicating whether or not the matched subscription is active.

is_sample_app_acknowledged(*self:* rti.connextdds.DynamicData.DataWriter, *sample_id:* rti.connextdds.SampleIdentity) → bool

Indicates if a sample is considered application-acknowledged.

key_value(*self:* rti.connextdds.DynamicData.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.DynamicData*

Retrieve the instance key that corresponds to an instance handle.

property listener

Get the listener associated with the DataWriter or set the listener.

property liveliness_lost_status

Get a copy of the LivelinessLostStatus.

lookup_instance(*self:* rti.connextdds.DynamicData.DataWriter, *key_holder:* rti.connextdds.DynamicData) → *rti.connextdds.InstanceHandle*

Retrieve the instance handle that corresponds to an instance key_holder

matched_subscription_data(*self:* rti.connextdds.DynamicData.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.SubscriptionBuiltinTopicData*

Get the SubscriptionBuiltinTopicData for a subscription matched to this DataWriter.

matched_subscription_datawriter_protocol_status(**args*, ***kwargs*)

Overloaded function.

1. matched_subscription_datawriter_protocol_status(self: rti.connextdds.DynamicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.DataWriterProtocolStatus

Get a copy of the protocol status for this writer per a matched subscription handle.

2. matched_subscription_datawriter_protocol_status(self: rti.connextdds.DynamicData.DataWriter, locator: rti.connextdds.Locator) -> rti.connextdds.DataWriterProtocolStatus

Get a copy of the protocol status for this writer per a matched subscription locator.

matched_subscription_participant_data (*self:* rti.connextdds.DynamicData.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ParticipantBuiltinTopicData*

Get the ParticipantBuiltinTopicData for a subscription matched to this DataWriter.

property matched_subscriptions

Get a copy of the list of the currently matched subscription handles.

property matched_subscriptions_locators

The locators used to communicate with matched DataReaders.

property offered_deadline_missed_status

Get a copy of the OfferedDeadlineMissedStatus.

property offered_incompatible_qos_status

Get a copy of the OfferedIncompatibleQosStatus

property publication_matched_status

Get a copy of the PublicationMatchedStatus

property publisher

Get the Publisher that owns this DataWriter.

property qos

The DataWriterQos for this DataWriter.This property's getter returns a deep copy.

register_instance (**args*, ***kwargs*)

Overloaded function.

1. register_instance(self: rti.connextdds.DynamicData.DataWriter, key_holder: rti.connextdds.DynamicData) -> rti.connextdds.InstanceHandle

Informs RTI Connext that the application will be modifying a particular instance.

2. register_instance(self: rti.connextdds.DynamicData.DataWriter, key_holder: rti.connextdds.DynamicData, timestamp: rti.connextdds.Time) -> rti.connextdds.InstanceHandle

Informs RTI Connext that the application will be modifying a particular instance and specified the timestamp.

3. register_instance(self: rti.connextdds.DynamicData.DataWriter, key_holder: rti.connextdds.DynamicData, params: rti.connextdds.WriteParams) -> rti.connextdds.InstanceHandle

Registers instance with parameters.

property reliable_reader_activity_changed_status

Get a copy of the reliable reader activity changed status for this writer.

property reliable_writer_cache_changed_status

Get a copy of the reliable cache status for this writer.

property service_request_accepted_status

Get a copy of the service request accepted status for this writer.

set_listener(*self:* rti.connextdds.DynamicData.DataWriter, *listener:* rti.connextdds.DynamicData.DataWriterListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and event mask for the DataWriter.

property topic

Get the Topic object associated with this DataWriter.

property topic_name

Get the topic name associated with this DataWriter.

property type_name

Get the type name for the topic object associated with this DataWriter.

unregister_instance(**args*, ***kwargs*)

Overloaded function.

1. unregister_instance(self: rti.connextdds.DynamicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.DynamicData.DataWriter

Unregister an instance.

2. unregister_instance(self: rti.connextdds.DynamicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> rti.connextdds.DynamicData.DataWriter

Unregister an instance with timestamp.

3. unregister_instance(self: rti.connextdds.DynamicData.DataWriter, params: rti.connextdds.WriteParams) -> None

Unregister an instance with parameters.

unregister_instance_async(**args*, ***kwargs*)

Overloaded function.

1. unregister_instance_async(self: rti.connextdds.DynamicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> object

Unregister an instance.

2. unregister_instance_async(self: rti.connextdds.DynamicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> object

Unregister an instance with timestamp.

3. unregister_instance_async(self: rti.connextdds.DynamicData.DataWriter, key_holder: rti.connextdds.DynamicData) -> object

Unregister the instance associated with key_holder.

4. unregister_instance_async(self: rti.connextdds.DynamicData.DataWriter, key_holder: rti.connextdds.DynamicData, timestamp: rti.connextdds.Time) -> object

Unregister the instance associate with key_holder using a timestamp.

5. unregister_instance_async(self: rti.connextdds.DynamicData.DataWriter, params: rti.connextdds.WriteParams) -> object

Unregister an instance with parameters.

wait_for_acknowledgments (*self:* rti.connextdds.DynamicData.DataWriter, *max_wait:* rti.connextdds.Duration) → None

Blocks the calling thread until all data written by a reliable DataWriter is acknowledged or until the timeout expires.

wait_for_asynchronous_publishing (*self:* rti.connextdds.DynamicData.DataWriter, *max_wait:* rti.connextdds.Duration) → None

This operation blocks the calling thread (up to max_wait) until all data written by the asynchronous DataWriter is sent and acknowledged (if reliable) by all matched DataReader entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; a time out indicates that max_wait elapsed before all the data was sent and/or acknowledged.

In other words, this guarantees that sending to best effort DataReader is complete in addition to what DataWriter.wait_for_acknowledgments() provides.

If the DataWriter does not have PublishMode kind set to PublishModeKind.ASYNCHRONOUS the operation will complete immediately

wait_for_asynchronous_publishing_async (*self:* rti.connextdds.DynamicData.DataWriter, *max_wait:* rti.connextdds.Duration) → object

This function is awaitable until either a timeout of max_wait or all data written by the asynchronous DataWriter is sent and acknowledged (if reliable) by all matched DataReader entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; a time out indicates that max_wait elapsed before all the data was sent and/or acknowledged.This function works with asyncio.

In other words, this guarantees that sending to best effort DataReader is complete in addition to what DataWriter.wait_for_acknowledgments() provides.

If the DataWriter does not have PublishMode kind set to PublishModeKind.ASYNCHRONOUS the operation will complete immediately

write (**args*, ***kwargs*)

Overloaded function.

1. write(self: rti.connextdds.DynamicData.DataWriter, samples: rti.connextdds.DynamicDataSeq) -> None

Write a sequence of samples.

2. write(self: rti.connextdds.DynamicData.DataWriter, samples: rti.connextdds.DynamicDataSeq, timestamp: rti.connextdds.Time) -> None

Write a sequence of samples with a timestamp.

3. write(self: rti.connextdds.DynamicData.DataWriter, sample: rti.connextdds.DynamicData) -> None

Write a sample.

4. write(self: rti.connextdds.DynamicData.DataWriter, sample: rti.connextdds.DynamicData, timestamp: rti.connextdds.Time) -> None

Write a sample with a specified timestamp.

5. write(self: rti.connextdds.DynamicData.DataWriter, sample: rti.connextdds.DynamicData, handle: rti.connextdds.InstanceHandle) -> None

Write a sample with an instance handle.

6. write(self: rti.connextdds.DynamicData.DataWriter, sample: rti.connextdds.DynamicData, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> None

Write a sample with an instance handle and specified timestamp.

7. write(self: rti.connextdds.DynamicData.DataWriter, sample: rti.connextdds.DynamicData, params: rti.connextdds.WriteParams) -> None

Write with advanced parameters.

8. write(self: rti.connextdds.DynamicData.DataWriter, sample_data: dict) -> None

Create a DynamicData object and write it with the given dictionary containing field names as keys.

write_async(**args*, ***kwargs*)

Overloaded function.

1. write_async(self: rti.connextdds.DynamicData.DataWriter, sample: rti.connextdds.DynamicData) -> object

Write a sample. This method is awaitable and is only for use with asyncio.

2. write_async(self: rti.connextdds.DynamicData.DataWriter, sample: rti.connextdds.DynamicData, timestamp: rti.connextdds.Time) -> object

Write a sample with a specified timestamp. This methods is awaitable and only for use with asyncio.

3. write_async(self: rti.connextdds.DynamicData.DataWriter, sample: rti.connextdds.DynamicData, handle: rti.connextdds.InstanceHandle) -> object

Write a sample with an instance handle. This method is awaitable and only for use with asyncio.

4. write_async(self: rti.connextdds.DynamicData.DataWriter, sample: rti.connextdds.DynamicData, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> object

Write a sample with an instance handle and specified timestamp. This method is awaitable and only for use with asyncio.

5. write_async(self: rti.connextdds.DynamicData.DataWriter, samples: rti.connextdds.DynamicDataSeq) -> object

Write a sequence of samples. This method is awaitable and only for use with asyncio.

6. write_async(self: rti.connextdds.DynamicData.DataWriter, samples: rti.connextdds.DynamicDataSeq, timestamp: rti.connextdds.Time) -> object

Write a sequence of samples with a timestamp. This method is awaitable and only for use with asyncio.

7. write_async(self: rti.connextdds.DynamicData.DataWriter, samples: rti.connextdds.DynamicDataSeq, handles: rti.connextdds.InstanceHandleSeq) -> object

Write a sequence of samples with their instance handles. This method is awaitable and only for use with asyncio.

8. write_async(self: rti.connextdds.DynamicData.DataWriter, samples: rti.connextdds.DynamicDataSeq, handles: rti.connextdds.InstanceHandleSeq, timestamp: rti.connextdds.Time) -> object

Write a sequence of samples with their instance handles and a timestamp. This method is awaitable and only for use with asyncio.

9. write_async(self: rti.connextdds.DynamicData.DataWriter, sample: rti.connextdds.DynamicData, params: rti.connextdds.WriteParams) -> object

Write with advanced parameters.

10. write_async(self: rti.connextdds.DynamicData.DataWriter, sample_data: dict) -> object

Create a DynamicData object and write it with the given dictionary containing field names as keys. This method is awaitable and is only for use with asyncio.

class DataWriterListener

Bases: `pybind11_object`

__init__ (*self:* rti.connextdds.DynamicData.DataWriterListener) → None

__module__ = 'rti.connextdds'

on_application_acknowledgment (*self:* rti.connextdds.DynamicData.DataWriterListener, *arg0:* rti.connextdds.DynamicData.DataWriter, *arg1:* rti.connextdds.AcknowledgmentInfo) → None

On application acknowledgment callback

on_instance_replaced (*self:* rti.connextdds.DynamicData.DataWriterListener, *arg0:* rti.connextdds.DynamicData.DataWriter, *arg1:* rti.connextdds.InstanceHandle) → None

On instance replaced callback.

on_liveliness_lost (*self:* rti.connextdds.DynamicData.DataWriterListener, *arg0:* rti.connextdds.DynamicData.DataWriter, *arg1:* rti.connextdds.LivelinessLostStatus) → None

Liveliness lost callback.

on_offered_deadline_missed (*self:* rti.connextdds.DynamicData.DataWriterListener, *arg0:* rti.connextdds.DynamicData.DataWriter, *arg1:* rti.connextdds.OfferedDeadlineMissedStatus) → None

Offered deadline missed callback.

on_offered_incompatible_qos (*self:* rti.connextdds.DynamicData.DataWriterListener, *arg0:* rti.connextdds.DynamicData.DataWriter, *arg1:* rti.connextdds.OfferedIncompatibleQosStatus) → None

Offered incompatible QoS callback.

on_publication_matched (*self:* rti.connextdds.DynamicData.DataWriterListener, *arg0:* rti.connextdds.DynamicData.DataWriter, *arg1:* rti.connextdds.PublicationMatchedStatus) → None

Publication matched callback.

on_reliable_reader_activity_changed(*self:* rti.connextdds.DynamicData.DataWriterListener, *arg0:* rti.connextdds.DynamicData.DataWriter, *arg1:* rti.connextdds.ReliableReaderActivityChangedStatus) → None

Reliable reader activity changed callback.

on_reliable_writer_cache_changed(*self:* rti.connextdds.DynamicData.DataWriterListener, *arg0:* rti.connextdds.DynamicData.DataWriter, *arg1:* rti.connextdds.ReliableWriterCacheChangedStatus) → None

Reliable writer cache changed callback.

on_service_request_accepted(*self:* rti.connextdds.DynamicData.DataWriterListener, *arg0:* rti.connextdds.DynamicData.DataWriter, *arg1:* rti.connextdds.ServiceRequestAcceptedStatus) → None

On service request accepted callback.

class DataWriterSeq

Bases: `pybind11_object`

__add__(*self:* rti.connextdds.DynamicData.DataWriterSeq, *arg0:* rti.connextdds.DynamicData.DataWriterSeq) → *rti.connextdds.DynamicData.DataWriterSeq*

__bool__(*self:* rti.connextdds.DynamicData.DataWriterSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.DynamicData.DataWriterSeq, *x:* rti.connextdds.DynamicData.DataWriter) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.DynamicData.DataWriterSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.DynamicData.DataWriterSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.DynamicData.DataWriterSeq, *arg0:* rti.connextdds.DynamicData.DataWriterSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.DynamicData.DataWriterSeq, s: slice) -> rti.connextdds.DynamicData.DataWriterSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.DynamicData.DataWriterSeq, arg0: int) -> rti.connextdds.DynamicData.DataWriter

__hash__ = None

__iadd__(*self:* rti.connextdds.DynamicData.DataWriterSeq, *arg0:* rti.connextdds.DynamicData.DataWriterSeq) → *rti.connextdds.DynamicData.DataWriterSeq*

__imul__(*self:* rti.connextdds.DynamicData.DataWriterSeq, *arg0: int*) → *rti.connextdds.DynamicData.DataWriterSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DynamicData.DataWriterSeq) -> None
2. __init__(self: rti.connextdds.DynamicData.DataWriterSeq, arg0: rti.connextdds.DynamicData.DataWriterSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.DynamicData.DataWriterSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.DynamicData.DataWriterSeq) → Iterator

__len__(*self:* rti.connextdds.DynamicData.DataWriterSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.DynamicData.DataWriterSeq, *arg0: int*) → *rti.connextdds.DynamicData.DataWriterSeq*

__ne__(*self:* rti.connextdds.DynamicData.DataWriterSeq, *arg0:* rti.connextdds.DynamicData.DataWriterSeq) → bool

__rmul__(*self:* rti.connextdds.DynamicData.DataWriterSeq, *arg0: int*) → *rti.connextdds.DynamicData.DataWriterSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.DynamicData.DataWriterSeq, arg0: int, arg1: rti.connextdds.DynamicData.DataWriter) -> None
2. __setitem__(self: rti.connextdds.DynamicData.DataWriterSeq, arg0: slice, arg1: rti.connextdds.DynamicData.DataWriterSeq) -> None

Assign list elements using a slice object

append(*self:* rti.connextdds.DynamicData.DataWriterSeq, *x:* rti.connextdds.DynamicData.DataWriter) → None

Add an item to the end of the list

clear(*self:* rti.connextdds.DynamicData.DataWriterSeq) → None

Clear the contents

count (*self:* rti.connextdds.DynamicData.DataWriterSeq, *x:* rti.connextdds.DynamicData.DataWriter) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.DynamicData.DataWriterSeq, L: rti.connextdds.DynamicData.DataWriterSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.DynamicData.DataWriterSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.DynamicData.DataWriterSeq, *i: int*, *x:* rti.connextdds.DynamicData.DataWriter) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.DynamicData.DataWriterSeq) -> rti.connextdds.DynamicData.DataWriter

Remove and return the last item

2. pop(self: rti.connextdds.DynamicData.DataWriterSeq, i: int) -> rti.connextdds.DynamicData.DataWriter

Remove and return the item at index `i`

remove (*self:* rti.connextdds.DynamicData.DataWriterSeq, *x:* rti.connextdds.DynamicData.DataWriter) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class FieldsIterator

Bases: `pybind11_object`

__init__ (*self:* rti.connextdds.DynamicData.FieldsIterator, *arg0:* rti.connextdds.DynamicData, *arg1: bool*) → None

__iter__ (*self:* rti.connextdds.DynamicData.FieldsIterator) → *rti.connextdds.DynamicData.FieldsIterator*

__module__ = 'rti.connextdds'

__next__ (*self:* rti.connextdds.DynamicData.FieldsIterator) → str

class FieldsView

Bases: `pybind11_object`

__contains__ (*self:* rti.connextdds.DynamicData.FieldsView, *arg0: str*) → bool

__init__ (*self:* rti.connextdds.DynamicData.FieldsView, *arg0:* rti.connextdds.DynamicData) → None

__iter__(*self:* rti.connextdds.DynamicData.FieldsView) → *rti.connextdds.DynamicData.FieldsIterator*

__len__(*self:* rti.connextdds.DynamicData.FieldsView) → int

__module__ = 'rti.connextdds'

__reversed__(*self:* rti.connextdds.DynamicData.FieldsView) → *rti.connextdds.DynamicData.FieldsIterator*

class ITopicDescription

Bases: *IEntity*

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property name

The name of the entity conforming to the ITopicDescription interface.

property participant

The parent DomainParticipant.

property type_name

The name of the associated type.

class IndexIterator

Bases: pybind11_object

__init__(*self:* rti.connextdds.DynamicData.IndexIterator, *arg0:* rti.connextdds.DynamicData, *arg1: bool*) → None

__iter__(*self:* rti.connextdds.DynamicData.IndexIterator) → *rti.connextdds.DynamicData.IndexIterator*

__module__ = 'rti.connextdds'

__next__(*self:* rti.connextdds.DynamicData.IndexIterator) → object

class ItemsIterator

Bases: pybind11_object

__init__(*self:* rti.connextdds.DynamicData.ItemsIterator, *arg0:* rti.connextdds.DynamicData, *arg1: bool*) → None

__iter__(*self:* rti.connextdds.DynamicData.ItemsIterator) → *rti.connextdds.DynamicData.ItemsIterator*

__module__ = 'rti.connextdds'

__next__(*self:* rti.connextdds.DynamicData.ItemsIterator) → Tuple[str, object]

class ItemsView

Bases: pybind11_object

__contains__(*self:* rti.connextdds.DynamicData.ItemsView, *arg0: Tuple[str, object]*) → bool

__init__(*self:* rti.connextdds.DynamicData.ItemsView, *arg0:* rti.connextdds.DynamicData) → None

__iter__(*self:* rti.connextdds.DynamicData.ItemsView) → *rti.connextdds.DynamicData.ItemsIterator*

__len__(*self:* rti.connextdds.DynamicData.ItemsView) → int

__module__ = 'rti.connextdds'

__reversed__(*self:* rti.connextdds.DynamicData.ItemsView) → *rti.connextdds.DynamicData.ItemsIterator*

class LoanedSample

Bases: pybind11_object

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DynamicData.LoanedSample) -> None

Basic constructor

2. __init__(self: rti.connextdds.DynamicData.LoanedSample, data: rti.connextdds.DynamicData, info: rti.connextdds.SampleInfo) -> None

Construct LoanedSample with data and info.

__iter__(*self:* rti.connextdds.DynamicData.LoanedSample) → object

__module__ = 'rti.connextdds'

property data

Get the data associated with the sample.

property info

Get the info associated with the sample.

class LoanedSamples

Bases: pybind11_object

__enter__(*self:* rti.connextdds.DynamicData.LoanedSamples) → *rti.connextdds.DynamicData.LoanedSamples*

Enter a context for the loaned samples, loan returned on context exit.

__exit__(*self:* rti.connextdds.DynamicData.LoanedSamples, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for the loaned samples, returning the resources.

__getitem__(*self:* rti.connextdds.DynamicData.LoanedSamples, *arg0: int*) → *rti.connextdds.DynamicData.LoanedSample*

Access a LoanedSample object in an array-like syntax

__init__(*self:* rti.connextdds.DynamicData.LoanedSamples) → None

Create an empty LoanedSamples object.

__iter__(*self:* rti.connextdds.DynamicData.LoanedSamples) → Iterator

__len__(*self:* rti.connextdds.DynamicData.LoanedSamples) → int

Get the number of samples in the loan.

__module__ = 'rti.connextdds'

property length

Get the number of samples in the loan.

return_loan(*self:* rti.connextdds.DynamicData.LoanedSamples) → None

Returns the loan to the DataReader.

class NoOpDataReaderListener

Bases: *DataReaderListener*

__init__(*self:* rti.connextdds.DynamicData.NoOpDataReaderListener) → None

__module__ = 'rti.connextdds'

on_data_available(*self:* rti.connextdds.DynamicData.NoOpDataReaderListener, *arg0:* rti.connextdds.DynamicData.DataReader) → None

Data available callback.

on_liveliness_changed(*self:* rti.connextdds.DynamicData.NoOpDataReaderListener, *arg0:* rti.connextdds.DynamicData.DataReader, *arg1:* rti.connextdds.LivelinessChangedStatus) → None

Liveliness changed callback.

on_requested_deadline_missed(*self:* rti.connextdds.DynamicData.NoOpDataReaderListener, *arg0:* rti.connextdds.DynamicData.DataReader, *arg1:* rti.connextdds.RequestedDeadlineMissedStatus) → None

Requested deadline missed callback.

on_requested_incompatible_qos(*self:* rti.connextdds.DynamicData.NoOpDataReaderListener, *arg0:* rti.connextdds.DynamicData.DataReader, *arg1:* rti.connextdds.RequestedIncompatibleQosStatus) → None

Requested incompatible QoS callback.

on_sample_lost(*self:* rti.connextdds.DynamicData.NoOpDataReaderListener, *arg0:* rti.connextdds.DynamicData.DataReader, *arg1:* rti.connextdds.SampleLostStatus) → None

Sample lost callback.

on_sample_rejected(*self:* rti.connextdds.DynamicData.NoOpDataReaderListener, *arg0:* rti.connextdds.DynamicData.DataReader, *arg1:* rti.connextdds.SampleRejectedStatus) → None

Sample rejected callback.

on_subscription_matched(*self:* rti.connextdds.DynamicData.NoOpDataReaderListener, *arg0:* rti.connextdds.DynamicData.DataReader, *arg1:* rti.connextdds.SubscriptionMatchedStatus) → None

Subscription matched callback.

class NoOpDataWriterListener

Bases: *DataWriterListener*

__init__(*self:* rti.connextdds.DynamicData.NoOpDataWriterListener) → None

__module__ = 'rti.connextdds'

on_application_acknowledgment(*self:* rti.connextdds.DynamicData.NoOpDataWriterListener, *arg0:* rti.connextdds.DynamicData.DataWriter, *arg1:* rti.connextdds.AcknowledgmentInfo) → None

On application acknowledgment callback

on_instance_replaced(*self:* rti.connextdds.DynamicData.NoOpDataWriterListener, *arg0:* rti.connextdds.DynamicData.DataWriter, *arg1:* rti.connextdds.InstanceHandle) → None

On instance replaced callback.

on_liveliness_lost(*self:* rti.connextdds.DynamicData.NoOpDataWriterListener, *arg0:* rti.connextdds.DynamicData.DataWriter, *arg1:* rti.connextdds.LivelinessLostStatus) → None

Liveliness lost callback.

on_offered_deadline_missed(*self:* rti.connextdds.DynamicData.NoOpDataWriterListener, *arg0:* rti.connextdds.DynamicData.DataWriter, *arg1:* rti.connextdds.OfferedDeadlineMissedStatus) → None

Offered deadline missed callback.

on_offered_incompatible_qos(*self:* rti.connextdds.DynamicData.NoOpDataWriterListener, *arg0:* rti.connextdds.DynamicData.DataWriter, *arg1:* rti.connextdds.OfferedIncompatibleQosStatus) → None

Offered incompatible QoS callback.

on_publication_matched(*self:* rti.connextdds.DynamicData.NoOpDataWriterListener, *arg0:* rti.connextdds.DynamicData.DataWriter, *arg1:* rti.connextdds.PublicationMatchedStatus) → None

Publication matched callback.

on_reliable_reader_activity_changed(*self:* rti.connextdds.DynamicData.NoOpDataWriterListener, *arg0:* rti.connextdds.DynamicData.DataWriter, *arg1:* rti.connextdds.ReliableReaderActivityChangedStatus) → None

Reliable reader activity changed callback.

on_reliable_writer_cache_changed(*self:* rti.connextdds.DynamicData.NoOpDataWriterListener, *arg0:* rti.connextdds.DynamicData.DataWriter, *arg1:* rti.connextdds.ReliableWriterCacheChangedStatus) → None

Reliable writer cache changed callback.

on_service_request_accepted(*self:* rti.connextdds.DynamicData.NoOpDataWriterListener, *arg0:* rti.connextdds.DynamicData.DataWriter, *arg1:* rti.connextdds.ServiceRequestAcceptedStatus) → None

On service request accepted callback.

class NoOpTopicListener

Bases: *TopicListener*

__init__(*self:* rti.connextdds.DynamicData.NoOpTopicListener) → None

__module__ = 'rti.connextdds'

on_inconsistent_topic(*self:* rti.connextdds.DynamicData.NoOpTopicListener, *arg0:* rti.connextdds.DynamicData.Topic, *arg1:* rti.connextdds.InconsistentTopicStatus) → None

Inconsistent topic callback.

class Sample

Bases: pybind11_object

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DynamicData.Sample, data: rti.connextdds.DynamicData, info: rti.connextdds.SampleInfo) -> None

Construct Sample with data and info.

2. __init__(self: rti.connextdds.DynamicData.Sample, sample: rti.connextdds.DynamicData.Sample) -> None

Copy constructor.

3. __init__(self: rti.connextdds.DynamicData.Sample, loaned_sample: rti.connextdds.DynamicData.LoanedSample) -> None

Copy constructor.

__iter__ (*self:* rti.connextdds.DynamicData.Sample) → object

__module__ = 'rti.connextdds'

property data

The data associated with the sample.

property info

Get the info associated with the sample.

class SharedSamples

Bases: pybind11_object

__getitem__ (*self:* rti.connextdds.DynamicData.SharedSamples, *arg0: int*) → *rti.connextdds.DynamicData.LoanedSample*

Get the sample at the specified index.

__init__ (*self:* rti.connextdds.DynamicData.SharedSamples, *loaned_samples:* rti.connextdds.DynamicData.LoanedSamples) → None

Constructs an instance of SharedSamples, removing ownership of the loan from the Loaned Samples.

__iter__ (*self:* rti.connextdds.DynamicData.SharedSamples) → Iterator

Make a sample iterator

__len__ (*self:* rti.connextdds.DynamicData.SharedSamples) → int

Returns the number of samples.

__module__ = 'rti.connextdds'

class Topic

Bases: *ITopicDescription*, *IAnyTopic*

__eq__ (*self:* rti.connextdds.DynamicData.Topic, *arg0:* rti.connextdds.DynamicData.Topic) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DynamicData.Topic, entity: rti.connextdds.IEntity) -> None

Cast an Entity to a Topic.

2. __init__(self: rti.connextdds.DynamicData.Topic, topic_description: rti.connextdds.DynamicData.ITopicDescription) -> None

Cast an ITopicDescription to a Topic.

3. __init__(self: rti.connextdds.DynamicData.Topic, topic: rti.connextdds.IAnyTopic) -> None

Create a typed Topic from an AnyTopic.

4. __init__(self: rti.connextdds.DynamicData.Topic, participant: rti.connextdds.DomainParticipant, topic_name: str, topic_type: rti.connextdds.DynamicType) -> None

Create a Topic with the given type.

5. __init__(self: rti.connextdds.DynamicData.Topic, participant: rti.connextdds.DomainParticipant, topic_name: str, topic_type: rti.connextdds.DynamicType, qos: rti.connextdds.TopicQos, listener: Optional[rti.connextdds.DynamicData.TopicListener] = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a Topic with the given type.

6. __init__(self: rti.connextdds.DynamicData.Topic, participant: rti.connextdds.DomainParticipant, topic_name: str, type_name: str, qos: rti.connextdds.TopicQos, listener: Optional[rti.connextdds.DynamicData.TopicListener] = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Creates a new Topic.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DynamicData.Topic, *arg0:* rti.connextdds.DynamicData.Topic) → bool

Test for inequality.

static find(*participant:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.DynamicData.Topic*]

Look up a Topic by its name in the DomainParticipant.

property inconsistent_topic_status

Get a copy of the current InconsistentTopicStatus for this Topic.

property listener

The listener.

property qos

Get the TopicQos for this Topic.

This property's getter returns a deep copy.

set_listener(*self:* rti.connextdds.DynamicData.Topic, *listener:* rti.connextdds.DynamicData.TopicListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and event mask.

class TopicDescription

Bases: *ITopicDescription*

__eq__(*self:* rti.connextdds.DynamicData.TopicDescription, *arg0:* rti.connextdds.DynamicData.TopicDescription) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DynamicData.TopicDescription, topic: rti.connextdds.DynamicData.ITopicDescription) -> None

Cast a Topic to a TopicDescription.

2. __init__(self: rti.connextdds.DynamicData.TopicDescription, entity: rti.connextdds.IEntity) -> None

Cast a Topic to a TopicDescription.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DynamicData.TopicDescription, *arg0:* rti.connextdds.DynamicData.TopicDescription) → bool

Test for inequality.

class TopicListener

Bases: pybind11_object

__init__(*self:* rti.connextdds.DynamicData.TopicListener) → None

__module__ = 'rti.connextdds'

on_inconsistent_topic(*self:* rti.connextdds.DynamicData.TopicListener, *arg0:* rti.connextdds.DynamicData.Topic, *arg1:* rti.connextdds.InconsistentTopicStatus) → None

Inconsistent topic callback.

class TopicSeq

Bases: pybind11_object

__add__(*self:* rti.connextdds.DynamicData.TopicSeq, *arg0:* rti.connextdds.DynamicData.TopicSeq) → *rti.connextdds.DynamicData.TopicSeq*

__bool__(*self:* rti.connextdds.DynamicData.TopicSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.DynamicData.TopicSeq, *x:* rti.connextdds.DynamicData.Topic) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.DynamicData.TopicSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.DynamicData.TopicSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.DynamicData.TopicSeq, *arg0:* rti.connextdds.DynamicData.TopicSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.DynamicData.TopicSeq, s: slice) -> rti.connextdds.DynamicData.TopicSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.DynamicData.TopicSeq, arg0: int) -> rti.connextdds.DynamicData.Topic

__hash__ = None

__iadd__(*self:* rti.connextdds.DynamicData.TopicSeq, *arg0:* rti.connextdds.DynamicData.TopicSeq) → *rti.connextdds.DynamicData.TopicSeq*

__imul__(*self:* rti.connextdds.DynamicData.TopicSeq, *arg0: int*) → *rti.connextdds.DynamicData.TopicSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DynamicData.TopicSeq) -> None
2. __init__(self: rti.connextdds.DynamicData.TopicSeq, arg0: rti.connextdds.DynamicData.TopicSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.DynamicData.TopicSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.DynamicData.TopicSeq) → Iterator

__len__(*self:* rti.connextdds.DynamicData.TopicSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.DynamicData.TopicSeq, *arg0: int*) → *rti.connextdds.DynamicData.TopicSeq*

__ne__(*self:* rti.connextdds.DynamicData.TopicSeq, *arg0:* rti.connextdds.DynamicData.TopicSeq) → bool

__rmul__(*self:* rti.connextdds.DynamicData.TopicSeq, *arg0: int*) → *rti.connextdds.DynamicData.TopicSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.DynamicData.TopicSeq, arg0: int, arg1: rti.connextdds.DynamicData.Topic) -> None
2. __setitem__(self: rti.connextdds.DynamicData.TopicSeq, arg0: slice, arg1: rti.connextdds.DynamicData.TopicSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.DynamicData.TopicSeq, *x:* rti.connextdds.DynamicData.Topic) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.DynamicData.TopicSeq) → None

Clear the contents

count (*self:* rti.connextdds.DynamicData.TopicSeq, *x:* rti.connextdds.DynamicData.Topic) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.DynamicData.TopicSeq, L: rti.connextdds.DynamicData.TopicSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.DynamicData.TopicSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.DynamicData.TopicSeq, *i: int*, *x:* rti.connextdds.DynamicData.Topic) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.DynamicData.TopicSeq) -> rti.connextdds.DynamicData.Topic

Remove and return the last item

2. pop(self: rti.connextdds.DynamicData.TopicSeq, i: int) -> rti.connextdds.DynamicData.Topic

Remove and return the item at index `i`

remove (*self:* rti.connextdds.DynamicData.TopicSeq, *x:* rti.connextdds.DynamicData.Topic) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class TopicTypeSupport

Bases: `pybind11_object`

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

static from_cdr_buffer (*sample:* rti.connextdds.DynamicData, *buffer:* rti.connextdds.CharSeq) → None

Deserialize a sample from a CDR buffer.

static initialize_sample (*arg0:* rti.connextdds.DynamicData) → None

static register_type (*participant:* rti.connextdds.DomainParticipant, *type_name: str*) → None

Register a type with a participant.

static to_cdr_buffer(*buffer:* rti.connextdds.CharSeq, *sample:* rti.connextdds.DynamicData, *representation_id: int = -1*) → *rti.connextdds.CharSeq*

Serialize sample to a CDR buffer.

class ValidLoanedSamples

Bases: pybind11_object

__enter__(*self:* rti.connextdds.DynamicData.ValidLoanedSamples) → *rti.connextdds.DynamicData.ValidLoanedSamples*

__exit__(*self:* rti.connextdds.DynamicData.ValidLoanedSamples, *arg0: object*, *arg1: object*, *arg2: object*) → None

__init__(**args*, ***kwargs*)

__iter__(*self:* rti.connextdds.DynamicData.ValidLoanedSamples) → Iterator

__module__ = 'rti.connextdds'

class WriterContentFilter

Bases: *ContentFilter*

__init__(*self:* rti.connextdds.DynamicData.WriterContentFilter) → None

__module__ = 'rti.connextdds'

writer_attach(*self:* rti.connextdds.DynamicData.WriterContentFilter) → Optional[object]

A writer-side filtering API to create some state that can facilitate filtering on the writer side.

writer_compile(*self:* rti.connextdds.DynamicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *property:* rti.connextdds.ExpressionProperty, *expression: str*, *parameters:* rti.connextdds.StringSeq, *type_code: Optional[*rti.connextdds.DynamicType*]*, *type_class_name: str*, *cookie:* rti.connextdds.Cookie) → None

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching DataReader.

writer_detach(*self:* rti.connextdds.DynamicData.WriterContentFilter, *writer_filter_data: Optional[object]*) → None

A writer-side filtering API to clean up a previously created state using writer_attach.

writer_evaluate(*self:* rti.connextdds.DynamicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *sample:* rti.connextdds.DynamicData, *meta_data:* rti.connextdds.FilterSampleInfo) → *rti.connextdds.CookieVector*

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching DataReader.

writer_finalize(*self:* rti.connextdds.DynamicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *cookie:* rti.connextdds.Cookie) → None

A writer-side filtering API to clean up a previously compiled instance of the content filter.

writer_return_loan(*self:* rti.connextdds.DynamicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *cookies:* rti.connextdds.CookieVector) → None

A writer-side filtering API to return the loan on the list of DataReaders returned by writer_evaluate.

class WriterContentFilterHelper

Bases: *WriterContentFilter*

__init__(*self:* rti.connextdds.DynamicData.WriterContentFilterHelper) → None

__module__ = 'rti.connextdds'

add_cookie(*self:* rti.connextdds.DynamicData.WriterContentFilterHelper, *cookie:* rti.connextdds.Cookie) → None

A helper function which will add a Cookie to the Cookie sequence that is then returned by the writer_evaluate function.

writer_evaluate_helper(*self:* rti.connextdds.DynamicData.WriterContentFilterHelper, *writer_filter_data: Optional[object]*, *sample:* rti.connextdds.DynamicData, *meta_data:* rti.connextdds.FilterSampleInfo) → None

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching DataReader.

__contains__(**args*, ***kwargs*)

Overloaded function.

1. __contains__(self: rti.connextdds.DynamicData, arg0: str) -> bool
2. __contains__(self: rti.connextdds.DynamicData, arg0: object) -> bool

__eq__(**args*, ***kwargs*)

Overloaded function.

1. __eq__(self: rti.connextdds.DynamicData, arg0: rti.connextdds.DynamicData) -> bool

Test for equality.

2. __eq__(self: rti.connextdds.DynamicData, arg0: dict) -> bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.DynamicData, arg0: str) -> object
2. __getitem__(self: rti.connextdds.DynamicData, arg0: int) -> object
3. __getitem__(self: rti.connextdds.DynamicData, arg0: tuple) -> object

4. __getitem__(self: rti.connextdds.DynamicData, arg0: slice) -> list

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DynamicData, dynamic_type: rti.connextdds.DynamicType) -> None

Create a DynamicData object from a DynamicType.

2. __init__(self: rti.connextdds.DynamicData, dynamic_type: rti.connextdds.DynamicType, property: rti.connextdds.DynamicDataProperty) -> None

Create a DynamicData object from a DynamicType with properties.

3. __init__(self: rti.connextdds.DynamicData, dynamic_type: rti.connextdds.DynamicType, data: dict) -> None

Create a DynamicData object from a DynamicType with a dict to initialize the fields

4. __init__(self: rti.connextdds.DynamicData, other: rti.connextdds.DynamicData) -> None

Create a copy of a DynamicData object.

__iter__(*self:* rti.connextdds.DynamicData) → object

__len__(*self:* rti.connextdds.DynamicData) → int

Number of members in the DynamicData object

__module__ = 'rti.connextdds'

__ne__(**args*, ***kwargs*)

Overloaded function.

1. __ne__(self: rti.connextdds.DynamicData, arg0: rti.connextdds.DynamicData) -> bool

Test for inequality.

2. __ne__(self: rti.connextdds.DynamicData, arg0: dict) -> bool

__reversed__(*self:* rti.connextdds.DynamicData) → object

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.DynamicData, arg0: str, arg1: object) -> None
2. __setitem__(self: rti.connextdds.DynamicData, arg0: int, arg1: object) -> None
3. __setitem__(self: rti.connextdds.DynamicData, arg0: tuple, arg1: object) -> None
4. __setitem__(self: rti.connextdds.DynamicData, arg0: slice, arg1: Iterable) -> None

__str__(*self:* rti.connextdds.DynamicData) → str

DynamicData value to string.

append (*self:* rti.connextdds.DynamicData, *value: object*) → None

Append a value to a sequence.

clear_all_members (*self:* rti.connextdds.DynamicData) → None

Clear the contents of all data members of this object, including key members.

clear_member (**args*, ***kwargs*)

Overloaded function.

1. clear_member(self: rti.connextdds.DynamicData, name: str) -> None

Clear the contents of a single data member of this object.

2. clear_member(self: rti.connextdds.DynamicData, index: int) -> None

Clear the contents of a single data member of this object.

clear_optional_member (**args*, ***kwargs*)

Overloaded function.

1. clear_optional_member(self: rti.connextdds.DynamicData, name: str) -> None

Clear the contents of a single optional data member of this object.

2. clear_optional_member(self: rti.connextdds.DynamicData, index: int) -> None

Clear the contents of a single optional data member of this object.

count (*self:* rti.connextdds.DynamicData, *arg0: object*) → int

property discriminator_value

Obtains the value of the union discriminator (valid for UnionType only).

extend (*self:* rti.connextdds.DynamicData, *values: Iterable*) → None

Extend a sequence.

fields (*self:* rti.connextdds.DynamicData) → *rti.connextdds.DynamicData.FieldsView*

from_cdr_buffer (**args*, ***kwargs*)

Overloaded function.

1. from_cdr_buffer(self: rti.connextdds.DynamicData, buffer: rti.connextdds.Uint8Seq) -> rti.connextdds.DynamicData

Populates a DynamicData sample by deserializing a CDR buffer.

2. from_cdr_buffer(self: rti.connextdds.DynamicData, signed_buffer: rti.connextdds.CharSeq) -> rti.connextdds.DynamicData

Populates a DynamicData sample by deserializing a CDR buffer.

from_json (*self:* rti.connextdds.DynamicData, *str: str*) → None

Populates a DynamicData object from a JSON string representation.

from_string (*self:* rti.connextdds.DynamicData, *str: str*, *format_kind:* rti.connextdds.PrintFormatKind) → None

Populates a DynamicData object from a JSON string representation. At the moment the only supported value for format_kind is dds.PrintFormatKind.JSON, and from_json() is preferred.

get_boolean (**args*, ***kwargs*)

Overloaded function.

1. get_boolean(self: rti.connextdds.DynamicData, name: str) -> bool

Get a boolean value by field name.

2. get_boolean(self: rti.connextdds.DynamicData, name: int) -> bool

Get a boolean value by field name.

get_cdr_buffer (*self:* rti.connextdds.DynamicData) → memoryview

Gets the CDR buffer of this DynamicData sample as a memoryview.

get_char (**args*, ***kwargs*)

Overloaded function.

1. get_char(self: rti.connextdds.DynamicData, name: str) -> str

Get a character value by field name.

2. get_char(self: rti.connextdds.DynamicData, name: int) -> str

Get a character value by field name.

get_char_values (**args*, ***kwargs*)

Overloaded function.

1. get_char_values(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.CharSeq

Get multiple character values by field name.

2. get_char_values(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.CharSeq

Get multiple character values by field name.

get_complex (**args*, ***kwargs*)

Overloaded function.

1. get_complex(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.DynamicData

Get a complex data type value by field name.

2. get_complex(self: rti.connextdds.DynamicData, name: int) -> rti.connextdds.DynamicData

Get a complex data type value by field name.

get_complex_values (**args*, ***kwargs*)

Overloaded function.

1. get_complex_values(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.DynamicDataSeq

Get a list of complex values by field name.

2. get_complex_values(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.DynamicDataSeq

Get a list of complex fields by index.

get_double(**args*, ***kwargs*)

Overloaded function.

1. get_double(self: rti.connextdds.DynamicData, name: str) -> float

Get a 64-bit floating point value by field name.

2. get_double(self: rti.connextdds.DynamicData, name: int) -> float

Get a 64-bit floating point value by field name.

get_double_values(**args*, ***kwargs*)

Overloaded function.

1. get_double_values(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.Float64Seq

Get multiple 64-bit floating point values by field name.

2. get_double_values(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.Float64Seq

Get multiple 64-bit floating point values by field name.

get_float(**args*, ***kwargs*)

Overloaded function.

1. get_float(self: rti.connextdds.DynamicData, name: str) -> float

Get a 32-bit floating point value by field name.

2. get_float(self: rti.connextdds.DynamicData, name: int) -> float

Get a 32-bit floating point value by field name.

get_float128(**args*, ***kwargs*)

Overloaded function.

1. get_float128(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.LongDouble

Get a 128-bit floating point value by field name.

2. get_float128(self: rti.connextdds.DynamicData, name: int) -> rti.connextdds.LongDouble

Get a 128-bit floating point value by field name.

get_float32(**args*, ***kwargs*)

Overloaded function.

1. get_float32(self: rti.connextdds.DynamicData, name: str) -> float

Get a 32-bit floating point value by field name.

2. get_float32(self: rti.connextdds.DynamicData, name: int) -> float

Get a 32-bit floating point value by field name.

get_float32_values(**args*, ***kwargs*)

Overloaded function.

1. get_float32_values(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.Float32Seq

Get multiple 32-bit floating point values by field name.

2. get_float32_values(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.Float32Seq

Get multiple 32-bit floating point values by field name.

get_float64(**args*, ***kwargs*)

Overloaded function.

1. get_float64(self: rti.connextdds.DynamicData, name: str) -> float

Get a 64-bit floating point value by field name.

2. get_float64(self: rti.connextdds.DynamicData, name: int) -> float

Get a 64-bit floating point value by field name.

get_float64_values(**args*, ***kwargs*)

Overloaded function.

1. get_float64_values(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.Float64Seq

Get multiple 64-bit floating point values by field name.

2. get_float64_values(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.Float64Seq

Get multiple 64-bit floating point values by field name.

get_float_values(**args*, ***kwargs*)

Overloaded function.

1. get_float_values(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.Float32Seq

Get multiple 32-bit floating point values by field name.

2. get_float_values(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.Float32Seq

Get multiple 32-bit floating point values by field name.

get_int(**args*, ***kwargs*)

Overloaded function.

1. get_int(self: rti.connextdds.DynamicData, name: str) -> int

Get a int (signed) value by field name.

2. get_int(self: rti.connextdds.DynamicData, name: int) -> int

Get a int (signed) value by field name.

get_int16(**args*, ***kwargs*)

Overloaded function.

1. get_int16(self: rti.connextdds.DynamicData, name: str) -> int

Get a 16-bit signed int value by field name.

2. get_int16(self: rti.connextdds.DynamicData, name: int) -> int

Get a 16-bit signed int value by field name.

get_int16_values(**args*, ***kwargs*)

Overloaded function.

1. get_int16_values(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.Int16Seq

Get multiple 16-bit signed int values by field name.

2. get_int16_values(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.Int16Seq

Get multiple 16-bit signed int values by field name.

get_int32(**args*, ***kwargs*)

Overloaded function.

1. get_int32(self: rti.connextdds.DynamicData, name: str) -> int

Get a 32-bit signed int value by field name.

2. get_int32(self: rti.connextdds.DynamicData, name: int) -> int

Get a 32-bit signed int value by field name.

get_int32_values(**args*, ***kwargs*)

Overloaded function.

1. get_int32_values(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.Int32Seq

Get multiple 32-bit signed int values by field name.

2. get_int32_values(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.Int32Seq

Get multiple 32-bit signed int values by field name.

get_int64(**args*, ***kwargs*)

Overloaded function.

1. get_int64(self: rti.connextdds.DynamicData, name: str) -> int

Get a 64-bit signed int value by field name.

2. get_int64(self: rti.connextdds.DynamicData, name: int) -> int

Get a 64-bit signed int value by field name.

get_int64_values (**args*, ***kwargs*)

Overloaded function.

1. get_int64_values(self: rti.connextdds.DynamicData, name: str) -> List[int]

Get multiple 64-bit signed int values by field name.

2. get_int64_values(self: rti.connextdds.DynamicData, index: int) -> List[int]

Get multiple 64-bit signed int values by field name.

get_int8 (**args*, ***kwargs*)

Overloaded function.

1. get_int8(self: rti.connextdds.DynamicData, name: str) -> int

Get an int8 value by field name.

2. get_int8(self: rti.connextdds.DynamicData, index: int) -> int

Get an int8 value by field index.

get_int_values (**args*, ***kwargs*)

Overloaded function.

1. get_int_values(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.Int32Seq

Get multiple 32-bit signed int values by field name.

2. get_int_values(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.Int32Seq

Get multiple 32-bit signed int values by field name.

get_long (**args*, ***kwargs*)

Overloaded function.

1. get_long(self: rti.connextdds.DynamicData, name: str) -> int

Get a 32-bit signed int value by field name.

2. get_long(self: rti.connextdds.DynamicData, name: int) -> int

Get a 32-bit signed int value by field name.

get_long_values (**args*, ***kwargs*)

Overloaded function.

1. get_long_values(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.Int32Seq

Get multiple 32-bit signed int values by field name.

2. get_long_values(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.Int32Seq

Get multiple 32-bit signed int values by field name.

get_longdouble (**args*, ***kwargs*)

Overloaded function.

1. get_longdouble(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.LongDouble

Get a 128-bit floating point value by field name.

2. get_longdouble(self: rti.connextdds.DynamicData, name: int) -> rti.connextdds.LongDouble

Get a 128-bit floating point value by field name.

get_longlong(**args*, ***kwargs*)

Overloaded function.

1. get_longlong(self: rti.connextdds.DynamicData, name: str) -> int

Get a 64-bit signed int value by field name.

2. get_longlong(self: rti.connextdds.DynamicData, name: int) -> int

Get a 64-bit signed int value by field name.

get_longlong_values(**args*, ***kwargs*)

Overloaded function.

1. get_longlong_values(self: rti.connextdds.DynamicData, name: str) -> List[int]

Get multiple 64-bit signed int values by field name.

2. get_longlong_values(self: rti.connextdds.DynamicData, index: int) -> List[int]

Get multiple 64-bit signed int values by field name.

get_octet(**args*, ***kwargs*)

Overloaded function.

1. get_octet(self: rti.connextdds.DynamicData, name: str) -> int

Get a 8-bit unsigned int value by field name.

2. get_octet(self: rti.connextdds.DynamicData, name: int) -> int

Get a 8-bit unsigned int value by field name.

get_octet_values(**args*, ***kwargs*)

Overloaded function.

1. get_octet_values(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.Uint8Seq

Get multiple 8-bit unsigned int values by field name.

2. get_octet_values(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.Uint8Seq

Get multiple 8-bit unsigned int values by field name.

get_short(**args*, ***kwargs*)

Overloaded function.

1. get_short(self: rti.connextdds.DynamicData, name: str) -> int

Get a 16-bit signed int value by field name.

2. get_short(self: rti.connextdds.DynamicData, name: int) -> int

Get a 16-bit signed int value by field name.

get_short_values(**args*, ***kwargs*)

Overloaded function.

1. get_short_values(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.Int16Seq

Get multiple 16-bit signed int values by field name.

2. get_short_values(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.Int16Seq

Get multiple 16-bit signed int values by field name.

get_string(**args*, ***kwargs*)

Overloaded function.

1. get_string(self: rti.connextdds.DynamicData, name: str) -> str

Get a string value by field name.

2. get_string(self: rti.connextdds.DynamicData, name: int) -> str

Get a string value by field name.

get_uint(**args*, ***kwargs*)

Overloaded function.

1. get_uint(self: rti.connextdds.DynamicData, name: str) -> int

Get a 32-bit unsigned int value by field name.

2. get_uint(self: rti.connextdds.DynamicData, name: int) -> int

Get a 32-bit unsigned int value by field name.

get_uint16(**args*, ***kwargs*)

Overloaded function.

1. get_uint16(self: rti.connextdds.DynamicData, name: str) -> int

Get a 16-bit unsigned int value by field name.

2. get_uint16(self: rti.connextdds.DynamicData, name: int) -> int

Get a 16-bit unsigned int value by field name.

get_uint16_values(**args*, ***kwargs*)

Overloaded function.

1. get_uint16_values(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.Uint16Seq

Get multiple 16-bit unsigned int values by field name.

2. get_uint16_values(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.Uint16Seq

Get multiple 16-bit unsigned int values by field name.

get_uint32 (**args*, ***kwargs*)

Overloaded function.

1. get_uint32(self: rti.connextdds.DynamicData, name: str) -> int

Get a 32-bit unsigned int value by field name.

2. get_uint32(self: rti.connextdds.DynamicData, name: int) -> int

Get a 32-bit unsigned int value by field name.

get_uint32_values (**args*, ***kwargs*)

Overloaded function.

1. get_uint32_values(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.Uint32Seq

Get multiple 32-bit unsigned int values by field name.

2. get_uint32_values(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.Uint32Seq

Get multiple 32-bit unsigned int values by field name.

get_uint64 (**args*, ***kwargs*)

Overloaded function.

1. get_uint64(self: rti.connextdds.DynamicData, name: str) -> int

Get a 64-bit unsigned int value by field name.

2. get_uint64(self: rti.connextdds.DynamicData, name: int) -> int

Get a 64-bit unsigned int value by field name.

get_uint64_values (**args*, ***kwargs*)

Overloaded function.

1. get_uint64_values(self: rti.connextdds.DynamicData, name: str) -> List[int]

Get multiple 64-bit unsigned int values by field name.

2. get_uint64_values(self: rti.connextdds.DynamicData, index: int) -> List[int]

Get multiple 64-bit unsigned int values by field name.

get_uint8 (**args*, ***kwargs*)

Overloaded function.

1. get_uint8(self: rti.connextdds.DynamicData, name: str) -> int

Get a 8-bit unsigned int value by field name.

2. get_uint8(self: rti.connextdds.DynamicData, name: int) -> int

Get a 8-bit unsigned int value by field name.

get_uint8_values(**args*, ***kwargs*)

Overloaded function.

1. get_uint8_values(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.Uint8Seq

Get multiple 8-bit unsigned int values by field name.

2. get_uint8_values(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.Uint8Seq

Get multiple 8-bit unsigned int values by field name.

get_uint_values(**args*, ***kwargs*)

Overloaded function.

1. get_uint_values(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.Uint32Seq

Get multiple 32-bit unsigned int values by field name.

2. get_uint_values(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.Uint32Seq

Get multiple 32-bit unsigned int values by field name.

get_ulong(**args*, ***kwargs*)

Overloaded function.

1. get_ulong(self: rti.connextdds.DynamicData, name: str) -> int

Get a 32-bit unsigned int value by field name.

2. get_ulong(self: rti.connextdds.DynamicData, name: int) -> int

Get a 32-bit unsigned int value by field name.

get_ulong_values(**args*, ***kwargs*)

Overloaded function.

1. get_ulong_values(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.Uint32Seq

Get multiple 32-bit unsigned int values by field name.

2. get_ulong_values(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.Uint32Seq

Get multiple 32-bit unsigned int values by field name.

get_ulonglong(**args*, ***kwargs*)

Overloaded function.

1. get_ulonglong(self: rti.connextdds.DynamicData, name: str) -> int

Get a 64-bit unsigned int value by field name.

2. get_ulonglong(self: rti.connextdds.DynamicData, name: int) -> int

Get a 64-bit unsigned int value by field name.

get_ulonglong_values(**args*, ***kwargs*)

Overloaded function.

1. get_ulonglong_values(self: rti.connextdds.DynamicData, name: str) -> List[int]

Get multiple 64-bit unsigned int values by field name.

2. get_ulonglong_values(self: rti.connextdds.DynamicData, index: int) -> List[int]

Get multiple 64-bit unsigned int values by field name.

get_ushort(**args*, ***kwargs*)

Overloaded function.

1. get_ushort(self: rti.connextdds.DynamicData, name: str) -> int

Get a 16-bit unsigned int value by field name.

2. get_ushort(self: rti.connextdds.DynamicData, name: int) -> int

Get a 16-bit unsigned int value by field name.

get_ushort_values(**args*, ***kwargs*)

Overloaded function.

1. get_ushort_values(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.Uint16Seq

Get multiple 16-bit unsigned int values by field name.

2. get_ushort_values(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.Uint16Seq

Get multiple 16-bit unsigned int values by field name.

get_value(**args*, ***kwargs*)

Overloaded function.

1. get_value(self: rti.connextdds.DynamicData, field_path: str) -> object

Automatically resolve type and return value for a field.

2. get_value(self: rti.connextdds.DynamicData, field_path: int) -> object

Automatically resolve type and return value for a field.

get_values(**args*, ***kwargs*)

Overloaded function.

1. get_values(self: rti.connextdds.DynamicData, field_path: str) -> object

Automatically resolve type and return collection for a field.

2. get_values(self: rti.connextdds.DynamicData, field_path: int) -> object

Automatically resolve type and return collection for a field.

get_wchar(**args*, ***kwargs*)

Overloaded function.

1. get_wchar(self: rti.connextdds.DynamicData, name: str) -> str

Get a wchar value by field name.

2. get_wchar(self: rti.connextdds.DynamicData, index: int) -> str

Get a wchar value by field index.

get_wstring(**args*, ***kwargs*)

Overloaded function.

1. get_wstring(self: rti.connextdds.DynamicData, name: str) -> str

Get a wstring value by field name.

2. get_wstring(self: rti.connextdds.DynamicData, index: int) -> str

Get a wstring value by field index.

property info

Returns info about this sample

property is_cdr

Determines if this DynamicData sample is represented as CDR

is_member_key(**args*, ***kwargs*)

Overloaded function.

1. is_member_key(self: rti.connextdds.DynamicData, name: str) -> bool

Returns whether a member is a key.

2. is_member_key(self: rti.connextdds.DynamicData, index: int) -> bool

Returns whether a member is a key.

items(*self:* rti.connextdds.DynamicData) → *rti.connextdds.DynamicData.ItemsView*

loan_value(**args*, ***kwargs*)

Overloaded function.

1. loan_value(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.LoanedDynamicData

Gets a view of a complex member.

2. loan_value(self: rti.connextdds.DynamicData, data: rti.connextdds.LoanedDynamicData, index: int) -> rti.connextdds.LoanedDynamicData

Gets a view of a complex member.

3. loan_value(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.LoanedDynamicData

Gets a view of a complex member.

4. loan_value(self: rti.connextdds.DynamicData, data: rti.connextdds.LoanedDynamicData, name: str) -> rti.connextdds.LoanedDynamicData

Gets a view of a complex member.

property member_count

Get the number of members in this sample.

member_exists(**args*, ***kwargs*)

Overloaded function.

1. member_exists(self: rti.connextdds.DynamicData, name: str) -> bool

Determine if an optional member is set by member name.

2. member_exists(self: rti.connextdds.DynamicData, index: int) -> bool

Determine if an optional member is set by member index.

member_exists_in_type(**args*, ***kwargs*)

Overloaded function.

1. member_exists_in_type(self: rti.connextdds.DynamicData, name: str) -> bool

Determine if a member with a particular name exists in the type.

2. member_exists_in_type(self: rti.connextdds.DynamicData, index: int) -> bool

Determine if a member with a particular index exists in the type.

member_index(*self:* rti.connextdds.DynamicData, *name: str*) → int

Translates from member name to member index.

member_info(**args*, ***kwargs*)

Overloaded function.

1. member_info(self: rti.connextdds.DynamicData, name: str) -> rti.connextdds.DynamicDataMemberInfo

Returns info about a member.

2. member_info(self: rti.connextdds.DynamicData, index: int) -> rti.connextdds.DynamicDataMemberInfo

Returns info about a member.

set_boolean(**args*, ***kwargs*)

Overloaded function.

1. set_boolean(self: rti.connextdds.DynamicData, name: str, value: bool) -> None

Set a boolean value by name.

2. set_boolean(self: rti.connextdds.DynamicData, index: int, value: bool) -> None

Set a boolean value by index.

set_cdr_buffer(*self:* rti.connextdds.DynamicData, *buffer: buffer*) → None

Sets the CDR buffer of this DynamicData sample.

buffer can be any object that supports the buffer protocol and produces a one-dimensional contiguous byte buffer.

set_char(**args*, ***kwargs*)

Overloaded function.

1. set_char(self: rti.connextdds.DynamicData, name: str, value: str) -> None

Set a character value by name.

2. set_char(self: rti.connextdds.DynamicData, index: int, value: str) -> None

Set a character value by index.

3. set_char(self: rti.connextdds.DynamicData, name: str, value: int) -> None

Set a char value by field name using an int.

4. set_char(self: rti.connextdds.DynamicData, index: int, value: int) -> None

Set a char value by field index using an int.

set_char_values(**args*, ***kwargs*)

Overloaded function.

1. set_char_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple character values by field name.

2. set_char_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple character values by field name.

set_complex(**args*, ***kwargs*)

Overloaded function.

1. set_complex(self: rti.connextdds.DynamicData, name: str, value: rti.connextdds.DynamicData) -> None

Set a complex data type value by name.

2. set_complex(self: rti.connextdds.DynamicData, index: int, value: rti.connextdds.DynamicData) -> None

Set a complex data type value by index.

set_complex_values(**args*, ***kwargs*)

Overloaded function.

1. set_complex_values(self: rti.connextdds.DynamicData, name: str, values: Iterable) -> None

Set a list of complex values by field name.

2. set_complex_values(self: rti.connextdds.DynamicData, index: int, values: Iterable) -> None

Set a list of complex values by index.

set_double(**args*, ***kwargs*)

Overloaded function.

1. set_double(self: rti.connextdds.DynamicData, name: str, value: float) -> None

Set a 64-bit floating point value by name.

2. set_double(self: rti.connextdds.DynamicData, index: int, value: float) -> None

Set a 64-bit floating point value by index.

set_double_values(**args*, ***kwargs*)

Overloaded function.

1. set_double_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 64-bit floating point values by field name.

2. set_double_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 64-bit floating point values by field name.

set_float(**args*, ***kwargs*)

Overloaded function.

1. set_float(self: rti.connextdds.DynamicData, name: str, value: float) -> None

Set a 32-bit floating point value by name.

2. set_float(self: rti.connextdds.DynamicData, index: int, value: float) -> None

Set a 32-bit floating point value by index.

set_float128(**args*, ***kwargs*)

Overloaded function.

1. set_float128(self: rti.connextdds.DynamicData, name: str, value: rti.connextdds.LongDouble) -> None

Set a 128-bit floating point value by name.

2. set_float128(self: rti.connextdds.DynamicData, index: int, value: rti.connextdds.LongDouble) -> None

Set a 128-bit floating point value by index.

set_float32(**args*, ***kwargs*)

Overloaded function.

1. set_float32(self: rti.connextdds.DynamicData, name: str, value: float) -> None

Set a 32-bit floating point value by name.

2. set_float32(self: rti.connextdds.DynamicData, index: int, value: float) -> None

Set a 32-bit floating point value by index.

set_float32_values(**args*, ***kwargs*)

Overloaded function.

1. set_float32_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 32-bit floating point values by field name.

2. set_float32_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 32-bit floating point values by field name.

set_float64 (**args*, ***kwargs*)

Overloaded function.

1. set_float64(self: rti.connextdds.DynamicData, name: str, value: float) -> None

Set a 64-bit floating point value by name.

2. set_float64(self: rti.connextdds.DynamicData, index: int, value: float) -> None

Set a 64-bit floating point value by index.

set_float64_values (**args*, ***kwargs*)

Overloaded function.

1. set_float64_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 64-bit floating point values by field name.

2. set_float64_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 64-bit floating point values by field name.

set_float_values (**args*, ***kwargs*)

Overloaded function.

1. set_float_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 32-bit floating point values by field name.

2. set_float_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 32-bit floating point values by field name.

set_int (**args*, ***kwargs*)

Overloaded function.

1. set_int(self: rti.connextdds.DynamicData, name: str, value: int) -> None

Set a int (signed) value by name.

2. set_int(self: rti.connextdds.DynamicData, index: int, value: int) -> None

Set a int (signed) value by index.

set_int16 (**args*, ***kwargs*)

Overloaded function.

1. set_int16(self: rti.connextdds.DynamicData, name: str, value: int) -> None

Set a 16-bit signed int value by name.

2. set_int16(self: rti.connextdds.DynamicData, index: int, value: int) -> None

Set a 16-bit signed int value by index.

set_int16_values (**args*, ***kwargs*)

Overloaded function.

1. set_int16_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 16-bit signed int values by field name.

2. set_int16_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 16-bit signed int values by field name.

set_int32 (**args*, ***kwargs*)

Overloaded function.

1. set_int32(self: rti.connextdds.DynamicData, name: str, value: int) -> None

Set a 32-bit signed int value by name.

2. set_int32(self: rti.connextdds.DynamicData, index: int, value: int) -> None

Set a 32-bit signed int value by index.

set_int32_values (**args*, ***kwargs*)

Overloaded function.

1. set_int32_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 32-bit signed int values by field name.

2. set_int32_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 32-bit signed int values by field name.

set_int64 (**args*, ***kwargs*)

Overloaded function.

1. set_int64(self: rti.connextdds.DynamicData, name: str, value: int) -> None

Set a 64-bit signed int value by name.

2. set_int64(self: rti.connextdds.DynamicData, index: int, value: int) -> None

Set a 64-bit signed int value by index.

set_int64_values (**args*, ***kwargs*)

Overloaded function.

1. set_int64_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 64-bit signed int values by field name.

2. set_int64_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 64-bit signed int values by field name.

set_int8 (**args*, ***kwargs*)

Overloaded function.

1. set_int8(self: rti.connextdds.DynamicData, name: str, value: int) -> None

Set an int8 value by field name using an int.

2. set_int8(self: rti.connextdds.DynamicData, index: int, value: int) -> None

Set an int8 value by field index using an int.

3. set_int8(self: rti.connextdds.DynamicData, name: str, value: str) -> None

Set an int8 value by field name using an char.

4. set_int8(self: rti.connextdds.DynamicData, index: int, value: str) -> None

Set an int8 value by field index using a char.

set_int_values(**args*, ***kwargs*)

Overloaded function.

1. set_int_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 32-bit signed int values by field name.

2. set_int_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 32-bit signed int values by field name.

set_long(**args*, ***kwargs*)

Overloaded function.

1. set_long(self: rti.connextdds.DynamicData, name: str, value: int) -> None

Set a 32-bit signed int value by name.

2. set_long(self: rti.connextdds.DynamicData, index: int, value: int) -> None

Set a 32-bit signed int value by index.

set_long_values(**args*, ***kwargs*)

Overloaded function.

1. set_long_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 32-bit signed int values by field name.

2. set_long_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 32-bit signed int values by field name.

set_longdouble(**args*, ***kwargs*)

Overloaded function.

1. set_longdouble(self: rti.connextdds.DynamicData, name: str, value: rti.connextdds.LongDouble) -> None

Set a 128-bit floating point value by name.

2. set_longdouble(self: rti.connextdds.DynamicData, index: int, value: rti.connextdds.LongDouble) -> None

Set a 128-bit floating point value by index.

set_longlong(**args*, ***kwargs*)

Overloaded function.

1. set_longlong(self: rti.connextdds.DynamicData, name: str, value: int) -> None

Set a 64-bit signed int value by name.

2. set_longlong(self: rti.connextdds.DynamicData, index: int, value: int) -> None

Set a 64-bit signed int value by index.

set_longlong_values(**args*, ***kwargs*)

Overloaded function.

1. set_longlong_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 64-bit signed int values by field name.

2. set_longlong_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 64-bit signed int values by field name.

set_octet(**args*, ***kwargs*)

Overloaded function.

1. set_octet(self: rti.connextdds.DynamicData, name: str, value: int) -> None

Set a 8-bit unsigned int value by name.

2. set_octet(self: rti.connextdds.DynamicData, index: int, value: int) -> None

Set a 8-bit unsigned int value by index.

set_octet_values(**args*, ***kwargs*)

Overloaded function.

1. set_octet_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 8-bit unsigned int values by field name.

2. set_octet_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 8-bit unsigned int values by field name.

set_short(**args*, ***kwargs*)

Overloaded function.

1. set_short(self: rti.connextdds.DynamicData, name: str, value: int) -> None

Set a 16-bit signed int value by name.

2. set_short(self: rti.connextdds.DynamicData, index: int, value: int) -> None

Set a 16-bit signed int value by index.

set_short_values(**args*, ***kwargs*)

Overloaded function.

1. set_short_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 16-bit signed int values by field name.

2. set_short_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 16-bit signed int values by field name.

set_string(**args*, ***kwargs*)

Overloaded function.

1. set_string(self: rti.connextdds.DynamicData, name: str, value: str) -> None

Set a string value by name.

2. set_string(self: rti.connextdds.DynamicData, index: int, value: str) -> None

Set a string value by index.

set_uint(**args*, ***kwargs*)

Overloaded function.

1. set_uint(self: rti.connextdds.DynamicData, name: str, value: int) -> None

Set a 32-bit unsigned int value by name.

2. set_uint(self: rti.connextdds.DynamicData, index: int, value: int) -> None

Set a 32-bit unsigned int value by index.

set_uint16(**args*, ***kwargs*)

Overloaded function.

1. set_uint16(self: rti.connextdds.DynamicData, name: str, value: int) -> None

Set a 16-bit unsigned int value by name.

2. set_uint16(self: rti.connextdds.DynamicData, index: int, value: int) -> None

Set a 16-bit unsigned int value by index.

set_uint16_values(**args*, ***kwargs*)

Overloaded function.

1. set_uint16_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 16-bit unsigned int values by field name.

2. set_uint16_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 16-bit unsigned int values by field name.

set_uint32(**args*, ***kwargs*)

Overloaded function.

1. set_uint32(self: rti.connextdds.DynamicData, name: str, value: int) -> None

Set a 32-bit unsigned int value by name.

2. set_uint32(self: rti.connextdds.DynamicData, index: int, value: int) -> None

Set a 32-bit unsigned int value by index.

set_uint32_values(**args*, ***kwargs*)

Overloaded function.

1. set_uint32_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 32-bit unsigned int values by field name.

2. set_uint32_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 32-bit unsigned int values by field name.

set_uint64 (**args*, ***kwargs*)

Overloaded function.

1. set_uint64(self: rti.connextdds.DynamicData, name: str, value: int) -> None

Set a 64-bit unsigned int value by name.

2. set_uint64(self: rti.connextdds.DynamicData, index: int, value: int) -> None

Set a 64-bit unsigned int value by index.

set_uint64_values (**args*, ***kwargs*)

Overloaded function.

1. set_uint64_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 64-bit unsigned int values by field name.

2. set_uint64_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 64-bit unsigned int values by field name.

set_uint8 (**args*, ***kwargs*)

Overloaded function.

1. set_uint8(self: rti.connextdds.DynamicData, name: str, value: int) -> None

Set a 8-bit unsigned int value by name.

2. set_uint8(self: rti.connextdds.DynamicData, index: int, value: int) -> None

Set a 8-bit unsigned int value by index.

set_uint8_values (**args*, ***kwargs*)

Overloaded function.

1. set_uint8_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 8-bit unsigned int values by field name.

2. set_uint8_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 8-bit unsigned int values by field name.

set_uint_values (**args*, ***kwargs*)

Overloaded function.

1. set_uint_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 32-bit unsigned int values by field name.

2. set_uint_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 32-bit unsigned int values by field name.

set_ulong(**args*, ***kwargs*)

Overloaded function.

1. set_ulong(self: rti.connextdds.DynamicData, name: str, value: int) -> None

Set a 32-bit unsigned int value by name.

2. set_ulong(self: rti.connextdds.DynamicData, index: int, value: int) -> None

Set a 32-bit unsigned int value by index.

set_ulong_values(**args*, ***kwargs*)

Overloaded function.

1. set_ulong_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 32-bit unsigned int values by field name.

2. set_ulong_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 32-bit unsigned int values by field name.

set_ulonglong(**args*, ***kwargs*)

Overloaded function.

1. set_ulonglong(self: rti.connextdds.DynamicData, name: str, value: int) -> None

Set a 64-bit unsigned int value by name.

2. set_ulonglong(self: rti.connextdds.DynamicData, index: int, value: int) -> None

Set a 64-bit unsigned int value by index.

set_ulonglong_values(**args*, ***kwargs*)

Overloaded function.

1. set_ulonglong_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 64-bit unsigned int values by field name.

2. set_ulonglong_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 64-bit unsigned int values by field name.

set_ushort(**args*, ***kwargs*)

Overloaded function.

1. set_ushort(self: rti.connextdds.DynamicData, name: str, value: int) -> None

Set a 16-bit unsigned int value by name.

2. set_ushort(self: rti.connextdds.DynamicData, index: int, value: int) -> None

Set a 16-bit unsigned int value by index.

set_ushort_values(**args*, ***kwargs*)

Overloaded function.

1. set_ushort_values(self: rti.connextdds.DynamicData, name: str, values: object) -> None

Get multiple 16-bit unsigned int values by field name.

2. set_ushort_values(self: rti.connextdds.DynamicData, index: int, values: object) -> None

Get multiple 16-bit unsigned int values by field name.

set_value(**args*, ***kwargs*)

Overloaded function.

1. set_value(self: rti.connextdds.DynamicData, field_path: str, value: object) -> None

Automatically resolve type and set value for a field.

2. set_value(self: rti.connextdds.DynamicData, field_path: int, value: object) -> None

Automatically resolve type and set value for a field.

set_values(**args*, ***kwargs*)

Overloaded function.

1. set_values(self: rti.connextdds.DynamicData, field_path: str, values: object) -> None

Automatically resolve type and set collection for a field.

2. set_values(self: rti.connextdds.DynamicData, field_path: int, values: object) -> None

Automatically resolve type and set collection for a field.

set_wchar(**args*, ***kwargs*)

Overloaded function.

1. set_wchar(self: rti.connextdds.DynamicData, name: str, value: str) -> None

Set a wchar value by field name.

2. set_wchar(self: rti.connextdds.DynamicData, index: int, value: str) -> None

Set a wchar value by field index.

set_wstring(**args*, ***kwargs*)

Overloaded function.

1. set_wstring(self: rti.connextdds.DynamicData, name: str, value: str) -> None

Set a wstring by field name.

2. set_wstring(self: rti.connextdds.DynamicData, index: int, value: str) -> None

Set a wstring value by field index.

to_cdr_buffer(*self:* rti.connextdds.DynamicData) → *rti.connextdds.CharSeq*

Serializes a DynamicData sample to CDR format

to_json(*self:* rti.connextdds.DynamicData) → str

Convert DynamicData object to a JSON string representation

to_string (*self:* rti.connextdds.DynamicData, *format:* rti.connextdds.PrintFormatProperty = *PrintFormatProperty.default*) → str

Convert DynamicData object to string with print format.

property type

Gets the data type of this DynamicData.

property type_kind

Gets this data type kind of this DynamicData.

update (*self:* rti.connextdds.DynamicData, *arg0: dict*) → None

class rti.connextdds.**DynamicDataEncapsulationKind**

Bases: pybind11_object

CDR_BIG_ENDIAN = <DynamicDataEncapsulationKind.CDR_BIG_ENDIAN: 0>

CDR_LITTLE_ENDIAN = <DynamicDataEncapsulationKind.CDR_LITTLE_ENDIAN: 1>

DEFAULT = <DynamicDataEncapsulationKind.DEFAULT: 2147483647>

class DynamicDataEncapsulationKind

Bases: pybind11_object

Members:

CDR_BIG_ENDIAN : CDR big endian encapsulation.

CDR_LITTLE_ENDIAN : CDR little endian encapsulation.

PL_CDR_BIG_ENDIAN : PL CDR big endian encapsulation.

PL_CDR_LITTLE_ENDIAN : PL CDR little endian encapsulation.

DEFAULT : Default encapsulation.

CDR_BIG_ENDIAN = <DynamicDataEncapsulationKind.CDR_BIG_ENDIAN: 0>

CDR_LITTLE_ENDIAN = <DynamicDataEncapsulationKind.CDR_LITTLE_ENDIAN: 1>

DEFAULT = <DynamicDataEncapsulationKind.DEFAULT: 2147483647>

PL_CDR_BIG_ENDIAN = <DynamicDataEncapsulationKind.PL_CDR_BIG_ENDIAN: 2>

PL_CDR_LITTLE_ENDIAN = <DynamicDataEncapsulationKind.PL_CDR_LITTLE_ENDIAN: 3>

__eq__ (*self: object*, *other: object*) → bool

__getstate__ (*self: object*) → int

__hash__ (*self: object*) → int

__index__ (*self:* rti.connextdds.DynamicDataEncapsulationKind.DynamicDataEncapsulationKind) → int

__init__ (*self:* rti.connextdds.DynamicDataEncapsulationKind.DynamicDataEncapsulationKind, *value: int*) → None

__int__ (*self:* rti.connextdds.DynamicDataEncapsulationKind.DynamicDataEncapsulationKind) → int

__members__ = {'CDR_BIG_ENDIAN': <DynamicDataEncapsulationKind.CDR_BIG_ENDIAN: 0>, 'CDR_LITTLE_ENDIAN': <DynamicDataEncapsulationKind.CDR_LITTLE_ENDIAN: 1>, 'DEFAULT': <DynamicDataEncapsulationKind.DEFAULT: 2147483647>, 'PL_CDR_BIG_ENDIAN': <DynamicDataEncapsulationKind.PL_CDR_BIG_ENDIAN: 2>, 'PL_CDR_LITTLE_ENDIAN': <DynamicDataEncapsulationKind.PL_CDR_LITTLE_ENDIAN: 3>}

__module__ = 'rti.connextdds'

__ne__ (*self: object*, *other: object*) → bool

__repr__ (*self: object*) → str

__setstate__ (*self:* rti.connextdds.DynamicDataEncapsulationKind.DynamicDataEncapsulationKind, *state: int*) → None

__str__ ()

name(self: handle) -> str

property name

property value

PL_CDR_BIG_ENDIAN = <DynamicDataEncapsulationKind.PL_CDR_BIG_ENDIAN: 2>

PL_CDR_LITTLE_ENDIAN = <DynamicDataEncapsulationKind.PL_CDR_LITTLE_ENDIAN: 3>

__eq__ (*self:* rti.connextdds.DynamicDataEncapsulationKind, *arg0:* rti.connextdds.DynamicDataEncapsulationKind) → bool

Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.DynamicDataEncapsulationKind, *arg0:* rti.connextdds.DynamicDataEncapsulationKind) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.DynamicDataEncapsulationKind, *arg0:* rti.connextdds.DynamicDataEncapsulationKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DynamicDataEncapsulationKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.DynamicDataEncapsulationKind, arg0: rti.connextdds.DynamicDataEncapsulationKind.DynamicDataEncapsulationKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.DynamicDataEncapsulationKind) → *rti.connextdds.DynamicDataEncapsulationKind.DynamicDataEncapsulationKind*

Int conversion.

__le__ (*self:* rti.connextdds.DynamicDataEncapsulationKind, *arg0:* rti.connextdds.DynamicDataEncapsulationKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.DynamicDataEncapsulationKind, *arg0:* rti.connextdds.DynamicDataEncapsulationKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DynamicDataEncapsulationKind, *arg0:* rti.connextdds.DynamicDataEncapsulationKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.DynamicDataEncapsulationKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**DynamicDataInfo**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.DynamicDataInfo, *arg0:* rti.connextdds.DynamicDataInfo) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DynamicDataInfo, *arg0:* rti.connextdds.DynamicDataInfo) → bool

Test for inequality.

property encapsulation_kind

The encapsulation kind.

property is_optimized_storage

Flag indicating whether storage is optimized.

property member_count

The number of members of this sample.

property stored_size

The number of bytes currently used to store the data of this sample.

class rti.connextdds.**DynamicDataMemberInfo**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.DynamicDataMemberInfo, *arg0:* rti.connextdds.DynamicDataMemberInfo) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.DynamicDataMemberInfo) → None

Create a DynamicDataMemberInfo object.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DynamicDataMemberInfo, *arg0:* rti.connextdds.DynamicDataMemberInfo) → bool

Test for inequality.

property element_count

The number of elements in the member.

property element_kind

The type kind of the elements in the member

property index

The member index

property kind

The member type kind

property name

The member name

class rti.connextdds.**DynamicDataProperty**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.DynamicDataProperty, *arg0:* rti.connextdds.DynamicDataProperty) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DynamicDataProperty) -> None

Construct with default settings.

2. __init__(self: rti.connextdds.DynamicDataProperty, initial_buffer_size: int, max_buffer_size: int, max_buffer_size_increment: int, check_buffer_size: bool) -> None

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DynamicDataProperty, *arg0:* rti.connextdds.DynamicDataProperty) → bool

Test for inequality.

property check_buffer_size

Check buffer size.

property initial_buffer_size

Initial buffer size.

property max_buffer_size

Max buffer size.

property max_buffer_size_increment

Max buffer size.

class rti.connextdds.**DynamicDataSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.DynamicDataSeq, *arg0:* rti.connextdds.DynamicDataSeq) → *rti.connextdds.DynamicDataSeq*

__bool__(*self:* rti.connextdds.DynamicDataSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.DynamicDataSeq, *x:* rti.connextdds.DynamicData) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.DynamicDataSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.DynamicDataSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.DynamicDataSeq, *arg0:* rti.connextdds.DynamicDataSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.DynamicDataSeq, s: slice) -> rti.connextdds.DynamicDataSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.DynamicDataSeq, arg0: int) -> rti.connextdds.DynamicData

__hash__ = None

__iadd__ (*self:* rti.connextdds.DynamicDataSeq, *arg0:* rti.connextdds.DynamicDataSeq) → *rti.connextdds.DynamicDataSeq*

__imul__ (*self:* rti.connextdds.DynamicDataSeq, *arg0: int*) → *rti.connextdds.DynamicDataSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DynamicDataSeq) -> None

2. __init__(self: rti.connextdds.DynamicDataSeq, arg0: rti.connextdds.DynamicDataSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.DynamicDataSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.DynamicDataSeq) → Iterator

__len__ (*self:* rti.connextdds.DynamicDataSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.DynamicDataSeq, *arg0: int*) → *rti.connextdds.DynamicDataSeq*

__ne__ (*self:* rti.connextdds.DynamicDataSeq, *arg0:* rti.connextdds.DynamicDataSeq) → bool

__repr__ (*self:* rti.connextdds.DynamicDataSeq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.DynamicDataSeq, *arg0: int*) → *rti.connextdds.DynamicDataSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.DynamicDataSeq, arg0: int, arg1: rti.connextdds.DynamicData) -> None

2. __setitem__(self: rti.connextdds.DynamicDataSeq, arg0: slice, arg1: rti.connextdds.DynamicDataSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.DynamicDataSeq, *x:* rti.connextdds.DynamicData) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.DynamicDataSeq) → None

Clear the contents

count (*self:* rti.connextdds.DynamicDataSeq, *x:* rti.connextdds.DynamicData) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.DynamicDataSeq, L: rti.connextdds.DynamicDataSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.DynamicDataSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.DynamicDataSeq, *i: int*, *x:* rti.connextdds.DynamicData) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.DynamicDataSeq) -> rti.connextdds.DynamicData

Remove and return the last item

2. pop(self: rti.connextdds.DynamicDataSeq, i: int) -> rti.connextdds.DynamicData

Remove and return the item at index `i`

remove (*self:* rti.connextdds.DynamicDataSeq, *x:* rti.connextdds.DynamicData) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**DynamicDataTimestampedSeq**

Bases: `pybind11_object`

__add__ (*self:* rti.connextdds.DynamicDataTimestampedSeq, *arg0:* rti.connextdds.DynamicDataTimestampedSeq) → *rti.connextdds.DynamicDataTimestampedSeq*

__bool__ (*self:* rti.connextdds.DynamicDataTimestampedSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.DynamicDataTimestampedSeq, *x:* *Tuple[*rti.connextdds.DynamicData, rti.connextdds.Time*]*) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.DynamicDataTimestampedSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.DynamicDataTimestampedSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.DynamicDataTimestampedSeq, *arg0:* rti.connextdds.DynamicDataTimestampedSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.DynamicDataTimestampedSeq, s: slice) -> rti.connextdds.DynamicDataTimestampedSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.DynamicDataTimestampedSeq, arg0: int) -> Tuple[rti.connextdds.DynamicData, rti.connextdds.Time]

__hash__ = None

__iadd__ (*self:* rti.connextdds.DynamicDataTimestampedSeq, *arg0:* rti.connextdds.DynamicDataTimestampedSeq) → *rti.connextdds.DynamicDataTimestampedSeq*

__imul__ (*self:* rti.connextdds.DynamicDataTimestampedSeq, *arg0: int*) → *rti.connextdds.DynamicDataTimestampedSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DynamicDataTimestampedSeq) -> None

2. __init__(self: rti.connextdds.DynamicDataTimestampedSeq, arg0: rti.connextdds.DynamicDataTimestampedSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.DynamicDataTimestampedSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.DynamicDataTimestampedSeq) → Iterator

__len__ (*self:* rti.connextdds.DynamicDataTimestampedSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.DynamicDataTimestampedSeq, *arg0: int*) → *rti.connextdds.DynamicDataTimestampedSeq*

__ne__ (*self:* rti.connextdds.DynamicDataTimestampedSeq, *arg0:* rti.connextdds.DynamicDataTimestampedSeq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__rmul__ (*self:* rti.connextdds.DynamicDataTimestampedSeq, *arg0: int*) → *rti.connextdds.DynamicDataTimestampedSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.DynamicDataTimestampedSeq, arg0: int, arg1: Tuple[rti.connextdds.DynamicData, rti.connextdds.Time]) -> None

2. __setitem__(self: rti.connextdds.DynamicDataTimestampedSeq, arg0: slice, arg1: rti.connextdds.DynamicDataTimestampedSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.DynamicDataTimestampedSeq, *x: Tuple[*rti.connextdds.DynamicData, rti.connextdds.Time*]*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.DynamicDataTimestampedSeq) → None

Clear the contents

count (*self:* rti.connextdds.DynamicDataTimestampedSeq, *x: Tuple[*rti.connextdds.DynamicData, rti.connextdds.Time*]*) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.DynamicDataTimestampedSeq, L: rti.connextdds.DynamicDataTimestampedSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.DynamicDataTimestampedSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.DynamicDataTimestampedSeq, *i: int*, *x: Tuple[*rti.connextdds.DynamicData, rti.connextdds.Time*]*) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.DynamicDataTimestampedSeq) -> Tuple[rti.connextdds.DynamicData, rti.connextdds.Time]

Remove and return the last item

2. pop(self: rti.connextdds.DynamicDataTimestampedSeq, i: int) -> Tuple[rti.connextdds.DynamicData, rti.connextdds.Time]

Remove and return the item at index `i`

remove (*self:* rti.connextdds.DynamicDataTimestampedSeq, *x: Tuple[*rti.connextdds.DynamicData, rti.connextdds.Time*]*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**DynamicDataTypeSerializationProperty**

Bases: pybind11_object

DEFAULT = <rti.connextdds.DynamicDataTypeSerializationProperty object>

__eq__ (*self:* rti.connextdds.DynamicDataTypeSerializationProperty, *arg0:* rti.connextdds.DynamicDataTypeSerializationProperty) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.DynamicDataTypeSerializationProperty) -> None

Construct with default settings.

2. __init__(self: rti.connextdds.DynamicDataTypeSerializationProperty, max_serialized_size: int, min_serialized_size: int, trim_to_size: bool, skip_deserialization: bool) -> None

Specify all serialization settings at object creation time.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.DynamicDataTypeSerializationProperty, *arg0:* rti.connextdds.DynamicDataTypeSerializationProperty) → bool

Test for inequality.

property max_serialized_size

The maximum number of bytes that objects of a given type could consume when serialized on the network. [DEPRECATED]

property min_serialized_size

The minimum number of bytes that objects of a given type could consume when serialized on the network. [DEPRECATED]

property skip_deserialization

Controls whether the DynamicData object will be deserialized by a DynamicDataReader.

property trim_to_size

Controls the growth of the buffer in a DynamicData object.

class rti.connextdds.**DynamicType**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.DynamicType, *arg0:* rti.connextdds.DynamicType) → bool

Compare DynamicType objects for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DynamicType, *arg0:* rti.connextdds.DynamicType) → bool

Compare DynamicType objects for inequality.

__str__(*self:* rti.connextdds.DynamicType) → str

DynamicData value to string.

is_aggregation_type(*self:* rti.connextdds.DynamicType) → bool

Determines if this DynamicType is an aggregation type.

is_collection_type(*self:* rti.connextdds.DynamicType) → bool

Determins if this DynamicType is a CollectionType.

is_constructed_type(*self:* rti.connextdds.DynamicType) → bool

Determines if this DynamicType is a constructed type.

is_keyed(*self:* rti.connextdds.DynamicType) → bool

Determines if this DynamicType has a key.

is_primitive_type(*self:* rti.connextdds.DynamicType) → bool

Determines if this DynamicType is a PrimitiveType

property kind

Get the type kind.

property name

Gets the name.

print_idl(*self:* rti.connextdds.DynamicType, *index: int = 0*) → None

Prints the IDL representation of this type to the standard output.

to_string(*self:* rti.connextdds.DynamicType, *format:* rti.connextdds.DynamicTypePrintFormatProperty = *DynamicTypePrintFormatProperty()*) → str

Convert DynamicType to string with print format.

class rti.connextdds.**DynamicTypePrintFormatProperty**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.DynamicTypePrintFormatProperty, *arg0:* rti.connextdds.DynamicTypePrintFormatProperty) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.DynamicTypePrintFormatProperty, *indent: int = 0*, *print_ordinals: bool = False*) → None

Construct with default settings.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.DynamicTypePrintFormatProperty, *arg0:* rti.connextdds.DynamicTypePrintFormatProperty) → bool

Test for inequality.

property indent

The amount of additional indent to be included when converting a DynamicType to a string.

property min_serialized_size

The minimum number of bytes that objects of a given type could consume when serialized on the network. [DEPRECATED]

class rti.connextdds.**EndpointGroup**

Bases: pybind11_object

__init__(*self:* rti.connextdds.EndpointGroup, *role_name: str*, *quorum_count: int*) → None

Create an EndpointGroup with the provided parameters.

__module__ = 'rti.connextdds'

property quorum_count

Get/set the EndpointGroup's quorum count.

property role_name

Get/set the EndpointGroup's role name.

class rti.connextdds.**EndpointGroupSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.EndpointGroupSeq, *arg0:* rti.connextdds.EndpointGroupSeq) → *rti.connextdds.EndpointGroupSeq*

__bool__(*self:* rti.connextdds.EndpointGroupSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.EndpointGroupSeq, *x:* rti.connextdds.EndpointGroup) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.EndpointGroupSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.EndpointGroupSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.EndpointGroupSeq, *arg0:* rti.connextdds.EndpointGroupSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.EndpointGroupSeq, s: slice) -> rti.connextdds.EndpointGroupSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.EndpointGroupSeq, arg0: int) -> rti.connextdds.EndpointGroup

__hash__ = None

__iadd__ (*self:* rti.connextdds.EndpointGroupSeq, *arg0:* rti.connextdds.EndpointGroupSeq) → *rti.connextdds.EndpointGroupSeq*

__imul__ (*self:* rti.connextdds.EndpointGroupSeq, *arg0: int*) → *rti.connextdds.EndpointGroupSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.EndpointGroupSeq) -> None

2. __init__(self: rti.connextdds.EndpointGroupSeq, arg0: rti.connextdds.EndpointGroupSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.EndpointGroupSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.EndpointGroupSeq) → Iterator

__len__ (*self:* rti.connextdds.EndpointGroupSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.EndpointGroupSeq, *arg0: int*) → *rti.connextdds.EndpointGroupSeq*

__ne__ (*self:* rti.connextdds.EndpointGroupSeq, *arg0:* rti.connextdds.EndpointGroupSeq) → bool

__rmul__ (*self:* rti.connextdds.EndpointGroupSeq, *arg0: int*) → *rti.connextdds.EndpointGroupSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.EndpointGroupSeq, arg0: int, arg1: rti.connextdds.EndpointGroup) -> None

2. __setitem__(self: rti.connextdds.EndpointGroupSeq, arg0: slice, arg1: rti.connextdds.EndpointGroupSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.EndpointGroupSeq, *x:* rti.connextdds.EndpointGroup) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.EndpointGroupSeq) → None

Clear the contents

count (*self:* rti.connextdds.EndpointGroupSeq, *x:* rti.connextdds.EndpointGroup) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.EndpointGroupSeq, L: rti.connextdds.EndpointGroupSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.EndpointGroupSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.EndpointGroupSeq, *i: int*, *x:* rti.connextdds.EndpointGroup) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.EndpointGroupSeq) -> rti.connextdds.EndpointGroup

Remove and return the last item

2. pop(self: rti.connextdds.EndpointGroupSeq, i: int) -> rti.connextdds.EndpointGroup

Remove and return the item at index `i`

remove (*self:* rti.connextdds.EndpointGroupSeq, *x:* rti.connextdds.EndpointGroup) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**EndpointGroupVector**

Bases: `pybind11_object`

A DDS standard container with functionality similar to a C++ vector.

__eq__ (*self:* rti.connextdds.EndpointGroupVector, *arg0:* rti.connextdds.EndpointGroupVector) → bool

Compare EndpointGroupVectors for equality.

__getitem__ (*self:* rti.connextdds.EndpointGroupVector, *arg0: int*) → *rti.connextdds.EndpointGroup*

Get the value at the specified index.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.EndpointGroupVector) -> None

Create an empty EndpointGroupVector

2. __init__(self: rti.connextdds.EndpointGroupVector, size: int) -> None

Create a EndpointGroupVector with a preallocated size.

3. __init__(self: rti.connextdds.EndpointGroupVector, vector: rti.connextdds.EndpointGroupVector) -> None

Create a copy from another EndpointGroupVector.

4. __init__(self: rti.connextdds.EndpointGroupVector, arg0: Iterable) -> None

5. __init__(self: rti.connextdds.EndpointGroupVector, list: rti.connextdds.EndpointGroupSeq) -> None

Create a EndpointGroupVector from a list of values.

__iter__ (*self:* rti.connextdds.EndpointGroupVector) → Iterator

Iterate over the contents of the vector.

__len__ (*self:* rti.connextdds.EndpointGroupVector) → int

Get the length of the EndpointGroupVector.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.EndpointGroupVector, *arg0:* rti.connextdds.EndpointGroupVector) → bool

Compare EndpointGroupVectors for inequality.

__setitem__ (*self:* rti.connextdds.EndpointGroupVector, *arg0: int*, *arg1:* rti.connextdds.EndpointGroup) → None

Set the value at the specified index.

clear (*self:* rti.connextdds.EndpointGroupVector) → None

Resize EndpointGroupVector to 0.

resize (*self:* rti.connextdds.EndpointGroupVector, *size: int*) → None

Resize EndpointGroupVector.

class rti.connextdds.**Entity**

Bases: *IEntity*

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

class rti.connextdds.**EntityFactory**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.EntityFactory, *arg0:* rti.connextdds.EntityFactory) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.EntityFactory) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.EntityFactory, auto_enable: bool) -> None

Specifies whether the entity acting as a factory automatically enables the instances it creates.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.EntityFactory, *arg0:* rti.connextdds.EntityFactory) → bool

Test for inequality.

auto_enable = <rti.connextdds.EntityFactory object>

property autoenable_created_entities

Whether the entity acting as a factory automatically enables the instances it creates.

manually_enable = <rti.connextdds.EntityFactory object>

class rti.connextdds.**EntityName**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.EntityName, *arg0:* rti.connextdds.EntityName) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.EntityName) -> None

Creates the default policy (no name).

2. __init__(self: rti.connextdds.EntityName, name: str) -> None

Creates an instance that specifies an entity name.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.EntityName, *arg0:* rti.connextdds.EntityName) → bool

Test for inequality.

property name

Sets the entity name.

property role_name

Sets the role name.

class rti.connextdds.**EntitySeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.EntitySeq, *arg0:* rti.connextdds.EntitySeq) → *rti.connextdds.EntitySeq*

__bool__(*self:* rti.connextdds.EntitySeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.EntitySeq, *x:* rti.connextdds.Entity) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.EntitySeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.EntitySeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.EntitySeq, *arg0:* rti.connextdds.EntitySeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.EntitySeq, s: slice) -> rti.connextdds.EntitySeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.EntitySeq, arg0: int) -> rti.connextdds.Entity

__hash__ = None

__iadd__(*self:* rti.connextdds.EntitySeq, *arg0:* rti.connextdds.EntitySeq) → *rti.connextdds.EntitySeq*

__imul__(*self:* rti.connextdds.EntitySeq, *arg0: int*) → *rti.connextdds.EntitySeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.EntitySeq) -> None

2. __init__(self: rti.connextdds.EntitySeq, arg0: rti.connextdds.EntitySeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.EntitySeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.EntitySeq) → Iterator

__len__ (*self:* rti.connextdds.EntitySeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.EntitySeq, *arg0: int*) → *rti.connextdds.EntitySeq*

__ne__ (*self:* rti.connextdds.EntitySeq, *arg0:* rti.connextdds.EntitySeq) → bool

__rmul__ (*self:* rti.connextdds.EntitySeq, *arg0: int*) → *rti.connextdds.EntitySeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.EntitySeq, arg0: int, arg1: rti.connextdds.Entity) -> None

2. __setitem__(self: rti.connextdds.EntitySeq, arg0: slice, arg1: rti.connextdds.EntitySeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.EntitySeq, *x:* rti.connextdds.Entity) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.EntitySeq) → None

Clear the contents

count (*self:* rti.connextdds.EntitySeq, *x:* rti.connextdds.Entity) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.EntitySeq, L: rti.connextdds.EntitySeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.EntitySeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.EntitySeq, *i: int*, *x:* rti.connextdds.Entity) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.EntitySeq) -> rti.connextdds.Entity

Remove and return the last item

2. pop(self: rti.connextdds.EntitySeq, i: int) -> rti.connextdds.Entity

Remove and return the item at index `i`

remove (*self:* rti.connextdds.EntitySeq, *x:* rti.connextdds.Entity) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**EnumMember**

Bases: pybind11_object

__eq__ (**args*, ***kwargs*)

Overloaded function.

1. __eq__(self: rti.connextdds.EnumMember, arg0: rti.connextdds.EnumMember) -> bool

Test for equality.

2. __eq__(self: rti.connextdds.EnumMember, arg0: rti.connextdds.EnumMember) -> bool

Test for equality.

3. __eq__(self: rti.connextdds.EnumMember, arg0: int) -> bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.EnumMember, *name: str*, *ordinal: int*) → None

Create an EnumMember with a given name and ordinal value.

__int__ (*self:* rti.connextdds.EnumMember) → int

__long__ (*self:* rti.connextdds.EnumMember) → int

__module__ = 'rti.connextdds'

__ne__ (**args*, ***kwargs*)

Overloaded function.

1. __ne__(self: rti.connextdds.EnumMember, arg0: rti.connextdds.EnumMember) -> bool

Test for inequality.

2. __ne__(self: rti.connextdds.EnumMember, arg0: int) -> bool

Test for inequality.

__str__ (*self:* rti.connextdds.EnumMember) → str

property name

The member name.

property ordinal

The member's ordinal.

class rti.connextdds.**EnumMemberSeq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.EnumMemberSeq, *arg0:* rti.connextdds.EnumMemberSeq) → *rti.connextdds.EnumMemberSeq*

__bool__ (*self:* rti.connextdds.EnumMemberSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.EnumMemberSeq, *x:* rti.connextdds.EnumMember) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.EnumMemberSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.EnumMemberSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.EnumMemberSeq, *arg0:* rti.connextdds.EnumMemberSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.EnumMemberSeq, s: slice) -> rti.connextdds.EnumMemberSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.EnumMemberSeq, arg0: int) -> rti.connextdds.EnumMember

__hash__ = None

__iadd__ (*self:* rti.connextdds.EnumMemberSeq, *arg0:* rti.connextdds.EnumMemberSeq) → *rti.connextdds.EnumMemberSeq*

__imul__ (*self:* rti.connextdds.EnumMemberSeq, *arg0: int*) → *rti.connextdds.EnumMemberSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.EnumMemberSeq) -> None

2. __init__(self: rti.connextdds.EnumMemberSeq, arg0: rti.connextdds.EnumMemberSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.EnumMemberSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.EnumMemberSeq) → Iterator

__len__ (*self:* rti.connextdds.EnumMemberSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.EnumMemberSeq, *arg0: int*) → *rti.connextdds.EnumMemberSeq*

__ne__ (*self:* rti.connextdds.EnumMemberSeq, *arg0:* rti.connextdds.EnumMemberSeq) → bool

__rmul__ (*self:* rti.connextdds.EnumMemberSeq, *arg0: int*) → *rti.connextdds.EnumMemberSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.EnumMemberSeq, arg0: int, arg1: rti.connextdds.EnumMember) -> None

2. __setitem__(self: rti.connextdds.EnumMemberSeq, arg0: slice, arg1: rti.connextdds.EnumMemberSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.EnumMemberSeq, *x:* rti.connextdds.EnumMember) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.EnumMemberSeq) → None

Clear the contents

count (*self:* rti.connextdds.EnumMemberSeq, *x:* rti.connextdds.EnumMember) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.EnumMemberSeq, L: rti.connextdds.EnumMemberSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.EnumMemberSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.EnumMemberSeq, *i: int*, *x:* rti.connextdds.EnumMember) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.EnumMemberSeq) -> rti.connextdds.EnumMember

Remove and return the last item

2. pop(self: rti.connextdds.EnumMemberSeq, i: int) -> rti.connextdds.EnumMember

Remove and return the item at index `i`

remove (*self:* rti.connextdds.EnumMemberSeq, *x:* rti.connextdds.EnumMember) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**EnumType**

Bases: *ACTEnumMember*

__eq__(*self:* rti.connextdds.EnumType, *arg0:* rti.connextdds.EnumType) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.EnumType, name: str) -> None

Creates an empty EnumType.

2. __init__(self: rti.connextdds.EnumType, name: str, members: rti.connextdds.EnumMemberSeq) -> None

Creates a enum with the members in the list.

3. __init__(self: rti.connextdds.EnumType, type: rti.connextdds.DynamicType) -> None

Cast a DynamicType to an EnumType.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.EnumType, *arg0:* rti.connextdds.EnumType) → bool

Test for inequality.

add_member(*self:* rti.connextdds.EnumType, *member:* rti.connextdds.EnumMember) → *rti.connextdds.EnumType*

Adds a member at the end.

add_members(*self:* rti.connextdds.EnumType, *members:* rti.connextdds.EnumMemberSeq) → *rti.connextdds.EnumType*

Adds a member at the end.

property extensibility_kind

Enum's extensibility kind.

find_member_by_ordinal(*self:* rti.connextdds.EnumType, *ordinal: int*) → int

Gets the index of the member with this ordinal value.

exception rti.connextdds.**Error**

Bases: *Exception*

__module__ = 'rti.connextdds'

class rti.connextdds.**Event**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.Event, *arg0:* rti.connextdds.Event) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Event) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.Event, thread: rti.connextdds.ThreadSettings, initial_count: int, max_count: int) -> None

Creates an instance with all the parameters set.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.Event, *arg0:* rti.connextdds.Event) → bool

Test for inequality.

property initial_count

Event thread QoS.

property max_count

Event thread QoS.

property thread

Event thread QoS.

class rti.connextdds.**EventCount32**

Bases: pybind11_object

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property change

The incremental count.

property total

The total count.

class rti.connextdds.**EventCount64**

Bases: pybind11_object

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property change

The incremental count.

property total

The total count.

exception rti.connextdds.**Exception**

Bases: *Exception*

__module__ = 'rti.connextdds'

__weakref__

list of weak references to the object (if defined)

class rti.connextdds.**ExclusiveArea**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.ExclusiveArea, *arg0:* rti.connextdds.ExclusiveArea) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ExclusiveArea) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.ExclusiveArea, use_shared_exclusive_area: bool) -> None

Creates an instance specifying the use of shared or exclusive area.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.ExclusiveArea, *arg0:* rti.connextdds.ExclusiveArea) → bool

Test for inequality.

exclusive_ea = <rti.connextdds.ExclusiveArea object>

shared_ea = <rti.connextdds.ExclusiveArea object>

property use_shared_exclusive_area

Whether the Entity is protected by its own exclusive area or the shared one.

class rti.connextdds.**ExpressionProperty**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.ExpressionProperty, *arg0:* rti.connextdds.ExpressionProperty) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ExpressionProperty) -> None

Create a default ExpressionProperty with key_only_filter = false and writer_side_filter_optimization = false.

2. __init__(self: rti.connextdds.ExpressionProperty, key_only_filter: bool, writer_side_filter_optimization: bool) -> None

Create an ExpressionProperty with the provided key_only_filter, and writer_side_filter_optimization.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ExpressionProperty, *arg0:* rti.connextdds.ExpressionProperty) → bool

Test for inequality.

property key_only_filter

The value for key_only_filter, indicating if the filter expression is based only on key fields. In this case, RTI Connext itself can cache the filtering results.

property writer_side_filter_optimization

The value for writer_side_filter_optimization, indicating if the filter implementation can cache the filtering result for the provided expression.

class rti.connextdds.**ExtensibilityKind**

Bases: pybind11_object

EXTENSIBLE = <ExtensibilityKind.EXTENSIBLE: 1>

class ExtensibilityKind

Bases: pybind11_object

Members:

FINAL : Final extensibility.

EXTENSIBLE : Extensible extensibility.

MUTABLE : Mutable extensibility.

EXTENSIBLE = <ExtensibilityKind.EXTENSIBLE: 1>

FINAL = <ExtensibilityKind.FINAL: 0>

MUTABLE = <ExtensibilityKind.MUTABLE: 2>

__eq__ (*self: object*, *other: object*) → bool

__getstate__ (*self: object*) → int

__hash__ (*self: object*) → int

__index__ (*self:* rti.connextdds.ExtensibilityKind.ExtensibilityKind) → int

__init__ (*self:* rti.connextdds.ExtensibilityKind.ExtensibilityKind, *value: int*) → None

__int__ (*self:* rti.connextdds.ExtensibilityKind.ExtensibilityKind) → int

__members__ = {'EXTENSIBLE': <ExtensibilityKind.EXTENSIBLE: 1>, 'FINAL': <ExtensibilityKind.FINAL: 0>, 'MUTABLE': <ExtensibilityKind.MUTABLE: 2>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.ExtensibilityKind.ExtensibilityKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

FINAL = <ExtensibilityKind.FINAL: 0>

MUTABLE = <ExtensibilityKind.MUTABLE: 2>

__eq__(*self:* rti.connextdds.ExtensibilityKind, *arg0:* rti.connextdds.ExtensibilityKind) → bool

Apply operator to underlying enumerated values.

__ge__(*self:* rti.connextdds.ExtensibilityKind, *arg0:* rti.connextdds.ExtensibilityKind) → bool

Apply operator to underlying enumerated values.

__gt__(*self:* rti.connextdds.ExtensibilityKind, *arg0:* rti.connextdds.ExtensibilityKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ExtensibilityKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.ExtensibilityKind, arg0: rti.connextdds.ExtensibilityKind.ExtensibilityKind) -> None

Copy constructor.

__int__(*self:* rti.connextdds.ExtensibilityKind) → *rti.connextdds.ExtensibilityKind.ExtensibilityKind*

Int conversion.

__le__(*self:* rti.connextdds.ExtensibilityKind, *arg0:* rti.connextdds.ExtensibilityKind) → bool

Apply operator to underlying enumerated values.

__lt__(*self:* rti.connextdds.ExtensibilityKind, *arg0:* rti.connextdds.ExtensibilityKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ExtensibilityKind, *arg0:* rti.connextdds.ExtensibilityKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.ExtensibilityKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**Filter**

Bases: pybind11_object

SQL_FILTER_NAME = 'DDSSQL'

STRINGMATCH_FILTER_NAME = 'DDSSTRINGMATCH'

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Filter, expression: str) -> None

Create a Filter with the specified expression.

2. __init__(self: rti.connextdds.Filter, expression: str, parameters: rti.connextdds.StringSeq) -> None

Create a Filter with the specified expression and parameters.

__iter__ (*self:* rti.connextdds.Filter) → Iterator

__module__ = 'rti.connextdds'

add_parameters (*self:* rti.connextdds.Filter, *parameter: str*) → *rti.connextdds.Filter*

Append a parameter to the Filter.

property expression

Get the Filter's expression.

property name

The name of the type of Filter.

property parameter_count

Get the number of parameters.

set_parameters (*self:* rti.connextdds.Filter, *arg0:* rti.connextdds.StringSeq) → None

Set the parameters for the Filter.

class rti.connextdds.**FilterSampleInfo**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.FilterSampleInfo, *arg0:* rti.connextdds.FilterSampleInfo) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.FilterSampleInfo, *arg0:* rti.connextdds.FilterSampleInfo) → bool

Test for inequality.

property priority

Get a positive integer designating the relative priority of the sample, used to determine the transmission order of pending transmissions.

property related_sample_identity

The Identity of another sample related to this one.

class rti.connextdds.**Float128Seq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.Float128Seq, *arg0:* rti.connextdds.Float128Seq) → *rti.connextdds.Float128Seq*

__bool__(*self:* rti.connextdds.Float128Seq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.Float128Seq, *x:* rti.connextdds.LongDouble) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.Float128Seq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.Float128Seq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.Float128Seq, *arg0:* rti.connextdds.Float128Seq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.Float128Seq, s: slice) -> rti.connextdds.Float128Seq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.Float128Seq, arg0: int) -> rti.connextdds.LongDouble

__hash__ = None

__iadd__(*self:* rti.connextdds.Float128Seq, *arg0:* rti.connextdds.Float128Seq) → *rti.connextdds.Float128Seq*

__imul__ (*self:* rti.connextdds.Float128Seq, *arg0: int*) → *rti.connextdds.Float128Seq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Float128Seq) -> None

2. __init__(self: rti.connextdds.Float128Seq, arg0: rti.connextdds.Float128Seq) -> None

Copy constructor

3. __init__(self: rti.connextdds.Float128Seq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.Float128Seq) → Iterator

__len__ (*self:* rti.connextdds.Float128Seq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.Float128Seq, *arg0: int*) → *rti.connextdds.Float128Seq*

__ne__ (*self:* rti.connextdds.Float128Seq, *arg0:* rti.connextdds.Float128Seq) → bool

__repr__ (*self:* rti.connextdds.Float128Seq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.Float128Seq, *arg0: int*) → *rti.connextdds.Float128Seq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.Float128Seq, arg0: int, arg1: rti.connextdds.LongDouble) -> None

2. __setitem__(self: rti.connextdds.Float128Seq, arg0: slice, arg1: rti.connextdds.Float128Seq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.Float128Seq, *x:* rti.connextdds.LongDouble) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.Float128Seq) → None

Clear the contents

count (*self:* rti.connextdds.Float128Seq, *x:* rti.connextdds.LongDouble) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.Float128Seq, L: rti.connextdds.Float128Seq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.Float128Seq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.Float128Seq, *i: int*, *x:* rti.connextdds.LongDouble) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.Float128Seq) -> rti.connextdds.LongDouble

Remove and return the last item

2. pop(self: rti.connextdds.Float128Seq, i: int) -> rti.connextdds.LongDouble

Remove and return the item at index `i`

remove (*self:* rti.connextdds.Float128Seq, *x:* rti.connextdds.LongDouble) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**Float128Type**

Bases: *DynamicType*

__eq__ (*self:* rti.connextdds.Float128Type, *arg0:* rti.connextdds.Float128Type) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.Float128Type) → None

Get the singleton for Float128Type

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Float128Type, *arg0:* rti.connextdds.Float128Type) → bool

Test for inequality.

class rti.connextdds.**Float32Seq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.Float32Seq, *arg0:* rti.connextdds.Float32Seq) → *rti.connextdds.Float32Seq*

__bool__ (*self:* rti.connextdds.Float32Seq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.Float32Seq, *x: float*) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.Float32Seq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.Float32Seq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.Float32Seq, *arg0:* rti.connextdds.Float32Seq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.Float32Seq, s: slice) -> rti.connextdds.Float32Seq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.Float32Seq, arg0: int) -> float

__getstate__ (*self:* rti.connextdds.Float32Seq) → bytes

__hash__ = None

__iadd__ (*self:* rti.connextdds.Float32Seq, *arg0:* rti.connextdds.Float32Seq) → *rti.connextdds.Float32Seq*

__imul__ (*self:* rti.connextdds.Float32Seq, *arg0: int*) → *rti.connextdds.Float32Seq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Float32Seq, arg0: buffer) -> None
2. __init__(self: rti.connextdds.Float32Seq) -> None
3. __init__(self: rti.connextdds.Float32Seq, arg0: rti.connextdds.Float32Seq) -> None

Copy constructor

4. __init__(self: rti.connextdds.Float32Seq, arg0: Iterable) -> None
5. __init__(self: rti.connextdds.Float32Seq, arg0: int) -> None

__iter__ (*self:* rti.connextdds.Float32Seq) → Iterator

__len__ (*self:* rti.connextdds.Float32Seq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.Float32Seq, *arg0: int*) → *rti.connextdds.Float32Seq*

__ne__ (*self:* rti.connextdds.Float32Seq, *arg0:* rti.connextdds.Float32Seq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__ (*self:* rti.connextdds.Float32Seq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.Float32Seq, *arg0: int*) → *rti.connextdds.Float32Seq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.Float32Seq, arg0: int, arg1: float) -> None
2. __setitem__(self: rti.connextdds.Float32Seq, arg0: slice, arg1: rti.connextdds.Float32Seq) -> None

Assign list elements using a slice object

__setstate__ (*self:* rti.connextdds.Float32Seq, *arg0: bytes*) → None

append (*self:* rti.connextdds.Float32Seq, *x: float*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.Float32Seq) → None

Clear the contents

count (*self:* rti.connextdds.Float32Seq, *x: float*) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.Float32Seq, L: rti.connextdds.Float32Seq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.Float32Seq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.Float32Seq, *i: int*, *x: float*) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.Float32Seq) -> float

Remove and return the last item

2. pop(self: rti.connextdds.Float32Seq, i: int) -> float

Remove and return the item at index `i`

remove (*self:* rti.connextdds.Float32Seq, *x: float*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

resize (*self:* rti.connextdds.Float32Seq, *arg0: int*) → None

resizes the vector to the given size

class rti.connextdds.**Float32Type**

Bases: *DynamicType*

__eq__ (*self:* rti.connextdds.Float32Type, *arg0:* rti.connextdds.Float32Type) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.Float32Type) → None

Get the singleton for Float32Type

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Float32Type, *arg0:* rti.connextdds.Float32Type) → bool

Test for inequality.

class rti.connextdds.**Float64Seq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.Float64Seq, *arg0:* rti.connextdds.Float64Seq) → *rti.connextdds.Float64Seq*

__bool__ (*self:* rti.connextdds.Float64Seq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.Float64Seq, *x: float*) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.Float64Seq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.Float64Seq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.Float64Seq, *arg0:* rti.connextdds.Float64Seq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.Float64Seq, s: slice) -> rti.connextdds.Float64Seq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.Float64Seq, arg0: int) -> float

__getstate__ (*self:* rti.connextdds.Float64Seq) → bytes

__hash__ = None

__iadd__ (*self:* rti.connextdds.Float64Seq, *arg0:* rti.connextdds.Float64Seq) → *rti.connextdds.Float64Seq*

__imul__ (*self:* rti.connextdds.Float64Seq, *arg0: int*) → *rti.connextdds.Float64Seq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Float64Seq, arg0: buffer) -> None
2. __init__(self: rti.connextdds.Float64Seq) -> None
3. __init__(self: rti.connextdds.Float64Seq, arg0: rti.connextdds.Float64Seq) -> None

Copy constructor

4. __init__(self: rti.connextdds.Float64Seq, arg0: Iterable) -> None
5. __init__(self: rti.connextdds.Float64Seq, arg0: int) -> None

__iter__ (*self:* rti.connextdds.Float64Seq) → Iterator

__len__ (*self:* rti.connextdds.Float64Seq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.Float64Seq, *arg0: int*) → *rti.connextdds.Float64Seq*

__ne__ (*self:* rti.connextdds.Float64Seq, *arg0:* rti.connextdds.Float64Seq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__ (*self:* rti.connextdds.Float64Seq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.Float64Seq, *arg0: int*) → *rti.connextdds.Float64Seq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.Float64Seq, arg0: int, arg1: float) -> None
2. __setitem__(self: rti.connextdds.Float64Seq, arg0: slice, arg1: rti.connextdds.Float64Seq) -> None

Assign list elements using a slice object

__setstate__ (*self:* rti.connextdds.Float64Seq, *arg0: bytes*) → None

append (*self:* rti.connextdds.Float64Seq, *x: float*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.Float64Seq) → None

Clear the contents

count (*self:* rti.connextdds.Float64Seq, *x: float*) → int

Return the number of times x appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.Float64Seq, L: rti.connextdds.Float64Seq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.Float64Seq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert(*self:* rti.connextdds.Float64Seq, *i: int*, *x: float*) → None

Insert an item at a given position.

pop(**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.Float64Seq) -> float

Remove and return the last item

2. pop(self: rti.connextdds.Float64Seq, i: int) -> float

Remove and return the item at index `i`

remove(*self:* rti.connextdds.Float64Seq, *x: float*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

resize(*self:* rti.connextdds.Float64Seq, *arg0: int*) → None

resizes the vector to the given size

class rti.connextdds.**Float64Type**

Bases: *DynamicType*

__eq__(*self:* rti.connextdds.Float64Type, *arg0:* rti.connextdds.Float64Type) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.Float64Type) → None

Get the singleton for Float64Type

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.Float64Type, *arg0:* rti.connextdds.Float64Type) → bool

Test for inequality.

rti.connextdds.**FloatSeq**

alias of *Float32Seq*

rti.connextdds.**FloatType**

alias of *Float32Type*

class rti.connextdds.**FlowController**

Bases: pybind11_object

DEFAULT_NAME = 'DDS_DEFAULT_FLOW_CONTROLLER_NAME'

FIXED_RATE_NAME = 'DDS_FIXED_RATE_FLOW_CONTROLLER_NAME'

ON_DEMAND_NAME = 'DDS_ON_DEMAND_FLOW_CONTROLLER_NAME'

__eq__ (*self:* rti.connextdds.FlowController, *arg0:* rti.connextdds.FlowController) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.FlowController, *participant:* rti.connextdds.DomainParticipant, *name: str*, *token_bucket:* rti.connextdds.FlowControllerProperty = *FlowControllerProperty()*) → None

Creates a FlowController with specific properties.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.FlowController, *arg0:* rti.connextdds.FlowController) → bool

Test for inequality.

close (*self:* rti.connextdds.FlowController) → None

Manually destroys this object.

property closed

Whether this FlowController has been closed.

property name

The name of this FlowController.

property participant

The participant of this FlowController.

property property

The configuration of this FlowController.

retain (*self:* rti.connextdds.FlowController) → None

Disables the automatic destruction of this object.

trigger_flow (*self:* rti.connextdds.FlowController) → None

Provides an external way to trigger a FlowController.

class rti.connextdds.**FlowControllerProperty**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.FlowControllerProperty, *arg0:* rti.connextdds.FlowControllerProperty) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.FlowControllerProperty) -> None

Creates a FlowControllerProperty with earliest-deadline-first scheduling policy and default token-bucket configuration.

2. __init__(self: rti.connextdds.FlowControllerProperty, scheduling_policy: rti.connextdds.FlowControllerSchedulingPolicy.FlowControllerSchedulingPolicy, token_bucket: rti.connextdds.FlowControllerTokenBucketProperty = FlowControllerTokenBucketProperty()) -> None

Creates a FlowControllerProperty with the specified scheduling policy and token-bucket configuration.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.FlowControllerProperty, *arg0:* rti.connextdds.FlowControllerProperty) → bool

Test for inequality.

property scheduling_policy

The scheduling policy.

property token_bucket

The token-bucket configuration

class rti.connextdds.**FlowControllerSchedulingPolicy**

Bases: pybind11_object

EARLIEST_DEADLINE_FIRST = <FlowControllerSchedulingPolicy.EARLIEST_DEADLINE_FIRST: 1>

class FlowControllerSchedulingPolicy

Bases: pybind11_object

Members:

ROUND_ROBIN : Indicates to flow control in a round-robin fashion.

Whenever tokens become available, the flow controller distributes the tokens uniformly across all of its (non-empty) destination queues. No destinations are prioritized. Instead, all destinations are treated equally and are serviced in a round-robin fashion.

EARLIEST_DEADLINE_FIRST : Indicates to flow control in an earliest-deadline-first fashion.

A sample's deadline is determined by the time it was written plus the latency budget of the DataWriter at the time of the write call (as specified in the LatencyBudget). The relative priority of a flow controller's destination queue is determined by the earliest deadline across all samples it contains.

When tokens become available, the FlowController distributes tokens to the destination queues in order of their deadline priority. In other words, the queue containing the sample with the earliest

deadline is serviced first. The number of tokens granted equals the number of tokens required to send the first sample in the queue. Note that the priority of a queue may change as samples are sent (i.e. removed from the queue). If a sample must be sent to multiple destinations or two samples have an equal deadline value, the corresponding destination queues are serviced in a round-robin fashion.

Hence, under the default LatencyBudget::duration setting, an EDF_FLOW_CONTROLLER_SCHED_POLICY FlowController preserves the order in which the user calls DataWriter.write() across the DataWriters associated with the flow controller.

Since the LatencyBudget is mutable, a sample written second may contain an earlier deadline than the sample written first if the LatencyBudget.duration value is sufficiently decreased in between writing the two samples. In that case, if the first sample is not yet written (still in queue waiting for its turn), it inherits the priority corresponding to the (earlier) deadline from the second sample.

In other words, the priority of a destination queue is always determined by the earliest deadline among all samples contained in the queue. This priority inheritance approach is required in order to both honor the updated LatencyBudget.duration and adhere to the DataWriter in-order data delivery guarantee.

[default] for DataWriter

> HIGHEST_PRIORITY_FIRST : Indicates to flow control in a highest-priority-first fashion.

Determines the next destination queue to service as determined by the publication priority of the DataWriter, channel of multi-channel DataWriter, or individual sample.

The relative priority of a flow controller's destination queue is determined by the highest publication priority of all samples it contains.

When tokens become available, the FlowController distributes tokens to the destination queues in order of their publication priority. In other words, the queue containing the sample with the highest publication priority is serviced first. The number of tokens granted equals the number of tokens required to send the first sample in the queue. Note that the priority of a queue may change as samples are sent (i.e. removed from the queue). If a sample must be sent to multiple destinations or two samples have an equal publication priority, the corresponding destination queues are serviced in a round-robin fashion.

This priority inheritance approach is required in order to both honor the designated publication priority and adhere to the DataWriter in-order data delivery guarantee.

EARLIEST_DEADLINE_FIRST =
<FlowControllerSchedulingPolicy.EARLIEST_DEADLINE_FIRST: 1>

HIGHEST_PRIORITY_FIRST =
<FlowControllerSchedulingPolicy.HIGHEST_PRIORITY_FIRST: 2>

ROUND_ROBIN = <FlowControllerSchedulingPolicy.ROUND_ROBIN: 0>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.FlowControllerSchedulingPolicy.FlowControllerSchedulingPolicy) → int

__init__(*self:* rti.connextdds.FlowControllerSchedulingPolicy.FlowControllerSchedulingPolicy, *value: int*) → None

__int__(*self:* rti.connextdds.FlowControllerSchedulingPolicy.FlowControllerSchedulingPolicy) → int

__members__ = {'EARLIEST_DEADLINE_FIRST': <FlowControllerSchedulingPolicy.EARLIEST_DEADLINE_FIRST: 1>, 'HIGHEST_PRIORITY_FIRST': <FlowControllerSchedulingPolicy.HIGHEST_PRIORITY_FIRST: 2>, 'ROUND_ROBIN': <FlowControllerSchedulingPolicy.ROUND_ROBIN: 0>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.FlowControllerSchedulingPolicy.FlowControllerSchedulingPolicy, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

HIGHEST_PRIORITY_FIRST = <FlowControllerSchedulingPolicy.HIGHEST_PRIORITY_FIRST: 2>

ROUND_ROBIN = <FlowControllerSchedulingPolicy.ROUND_ROBIN: 0>

__eq__(*self:* rti.connextdds.FlowControllerSchedulingPolicy, *arg0:* rti.connextdds.FlowControllerSchedulingPolicy) → bool

Apply operator to underlying enumerated values.

__ge__(*self:* rti.connextdds.FlowControllerSchedulingPolicy, *arg0:* rti.connextdds.FlowControllerSchedulingPolicy) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.FlowControllerSchedulingPolicy, *arg0:* rti.connextdds.FlowControllerSchedulingPolicy) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.FlowControllerSchedulingPolicy) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.FlowControllerSchedulingPolicy, arg0: rti.connextdds.FlowControllerSchedulingPolicy.FlowControllerSchedulingPolicy) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.FlowControllerSchedulingPolicy) → *rti.connextdds.FlowControllerSchedulingPolicy.FlowControllerSchedulingPolicy*

Int conversion.

__le__ (*self:* rti.connextdds.FlowControllerSchedulingPolicy, *arg0:* rti.connextdds.FlowControllerSchedulingPolicy) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.FlowControllerSchedulingPolicy, *arg0:* rti.connextdds.FlowControllerSchedulingPolicy) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.FlowControllerSchedulingPolicy, *arg0:* rti.connextdds.FlowControllerSchedulingPolicy) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.FlowControllerSchedulingPolicy) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**FlowControllerTokenBucketProperty**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.FlowControllerTokenBucketProperty, *arg0:* rti.connextdds.FlowControllerTokenBucketProperty) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.FlowControllerTokenBucketProperty, *max_tokens: int = LENGTH_UNLIMITED*, *tokens_added_per_period: int = LENGTH_UNLIMITED*, *tokens_leaked_per_period: int = 0*, *period:* rti.connextdds.Duration = *Duration(1, 0)*, *bytes_per_token: int = LENGTH_UNLIMITED*) → None

Initializes the properties.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.FlowControllerTokenBucketProperty, *arg0:* rti.connextdds.FlowControllerTokenBucketProperty) → bool

Test for inequality.

property bytes_per_token

The maximum number of bytes allowed to send for each token available.

property max_tokens

The maximum number of tokens that can accumulate in the token bucket.

property period

The period for adding tokens to and removing tokens from the bucket.

property tokens_added_per_period

The number of tokens added to the token bucket per specified period.

property tokens_leaked_per_period

The number of tokens removed from the token bucket per specified period.

class rti.connextdds.**GenerationCount**

Bases: pybind11_object

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.GenerationCount) -> None

Create a default GenerationCount object.

2. __init__(self: rti.connextdds.GenerationCount, disposed_count: int, no_writers_count: int) -> None

Create a GenerationCount object with the provided disposed_count and no_writers count.

__module__ = 'rti.connextdds'

property disposed

Get the disposed generation count.

property no_writers

Get the no_writers generation count.

class rti.connextdds.**GroupData**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.GroupData, *arg0:* rti.connextdds.GroupData) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.GroupData) -> None

Create a GroupData instance.

2. __init__(self: rti.connextdds.GroupData, bytes: rti.connextdds.Uint8Seq) -> None

Create a GroupData instance with a sequence of bytes.

__iter__ (*self:* rti.connextdds.GroupData) → Iterator

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.GroupData, *arg0:* rti.connextdds.GroupData) → bool

Test for inequality.

property value

The byte sequence of this GroupData.

class rti.connextdds.**GuardCondition**

Bases: *ICondition*

__eq__ (*self:* rti.connextdds.GuardCondition, *arg0:* rti.connextdds.GuardCondition) → bool

Compare GuardCondition objects for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.GuardCondition) -> None

Create a GuardCondition in an untriggered state.

2. __init__(self: rti.connextdds.GuardCondition, condition: rti.connextdds.ICondition) -> None

Create a GuardCondition from a Condition.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.GuardCondition, *arg0:* rti.connextdds.GuardCondition) → bool

Compare GuardCondition objects for inequality.

reset_handler (*self:* rti.connextdds.GuardCondition) → None

Resets the handler for this GuardCondition.

set_handler (*self:* rti.connextdds.GuardCondition, *func:* *Callable[[*rti.connextdds.GuardCondition*], None]*) → None

Set a handler function receiving the Condition as its only argument.

set_handler_no_args (*self:* rti.connextdds.GuardCondition, *func: Callable[[], None]*) → None

Set a handler function with no arguments.

property trigger_value

Get/set the trigger value for this GuardCondition

class rti.connextdds.**Guid**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.Guid, *arg0:* rti.connextdds.Guid) → bool

Compare Guids for equality.

__ge__ (*self:* rti.connextdds.Guid, *arg0:* rti.connextdds.Guid) → bool

Compare Guids.

__getitem__ (*self:* rti.connextdds.Guid, *arg0: int*) → int

__gt__ (*self:* rti.connextdds.Guid, *arg0:* rti.connextdds.Guid) → bool

Compare Guids.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Guid) -> None

Create a default Guid (equivalent to unknown Guid).

2. __init__(self: rti.connextdds.Guid, builtin_topic_key: rti.connextdds.BuiltinTopicKey) -> None

Create a Guid from a BuiltinTopicKey.

__le__ (*self:* rti.connextdds.Guid, *arg0:* rti.connextdds.Guid) → bool

Compare Guids.

__len__ (*self:* rti.connextdds.Guid) → int

__lt__ (*self:* rti.connextdds.Guid, *arg0:* rti.connextdds.Guid) → bool

Compare Guids.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Guid, *arg0:* rti.connextdds.Guid) → bool

Compare Guids for inequality.

__setitem__ (*self:* rti.connextdds.Guid, *arg0: int*, *arg1: int*) → int

__str__ (*self:* rti.connextdds.Guid) → str

automatic = <rti.connextdds.Guid object>

unknown = <rti.connextdds.Guid object>

class rti.connextdds.**History**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.History, *arg0:* rti.connextdds.History) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.History) -> None

Creates a policy that keeps the last sample only.

2. __init__(self: rti.connextdds.History, kind: rti.connextdds.HistoryKind, depth: int = 1) -> None

Creates a policy with a specific history kind and optionally a history depth.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.History, *arg0:* rti.connextdds.History) → bool

Test for inequality.

property depth

The history kind.

keep_all = <rti.connextdds.History object>

static keep_last (*depth: int*) → *rti.connextdds.History*

Creates a History with HistoryKind.KEEP_LAST and the specified depth.

property kind

The history kind.

class rti.connextdds.**HistoryKind**

Bases: pybind11_object

class HistoryKind

Bases: pybind11_object

Members:

KEEP_LAST : [default] Keep the last depth samples.

On the publishing side, RTI Connext will only attempt to keep the most recent depth samples of each instance of data (identified by its key) managed by the DataWriter. Invalid samples represeting a disposal or unregistration of an instance do not count towards the history depth.

On the subscribing side, the DataReader will only attempt to keep the most recent depth samples received for each instance (identified by its key) until the application takes them via the DataReader 's take() operation.

Invalid samples represeting a dispoasal or unregistration of an instance do count towards the history depth and may therefore replace a value of the instance currently being stored in the reader queue.

KEEP_ALL : Keep all the samples.

On the publishing side, RTI Connext will attempt to keep all samples (representing each value written) of each instance of data (identified by its key) managed by the DataWriter until they can be delivered to all subscribers.

On the subscribing side, RTI Connext will attempt to keep all samples of each instance of data (identified by its key) managed by the DataReader. These samples are kept until the application takes them from RTI Connext via the take() operation.

KEEP_ALL = <HistoryKind.KEEP_ALL: 1>

KEEP_LAST = <HistoryKind.KEEP_LAST: 0>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.HistoryKind.HistoryKind) → int

__init__(*self:* rti.connextdds.HistoryKind.HistoryKind, *value: int*) → None

__int__(*self:* rti.connextdds.HistoryKind.HistoryKind) → int

__members__ = {'KEEP_ALL': <HistoryKind.KEEP_ALL: 1>, 'KEEP_LAST': <HistoryKind.KEEP_LAST: 0>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.HistoryKind.HistoryKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

KEEP_ALL = <HistoryKind.KEEP_ALL: 1>

KEEP_LAST = <HistoryKind.KEEP_LAST: 0>

__eq__ (*self:* rti.connextdds.HistoryKind, *arg0:* rti.connextdds.HistoryKind) → bool

Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.HistoryKind, *arg0:* rti.connextdds.HistoryKind) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.HistoryKind, *arg0:* rti.connextdds.HistoryKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.HistoryKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.HistoryKind, arg0: rti.connextdds.HistoryKind.HistoryKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.HistoryKind) → *rti.connextdds.HistoryKind.HistoryKind*

Int conversion.

__le__ (*self:* rti.connextdds.HistoryKind, *arg0:* rti.connextdds.HistoryKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.HistoryKind, *arg0:* rti.connextdds.HistoryKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.HistoryKind, *arg0:* rti.connextdds.HistoryKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.HistoryKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**IAnyDataReader**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.IAnyDataReader, *arg0:* rti.connextdds.IAnyDataReader) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.IAnyDataReader, *arg0:* rti.connextdds.IAnyDataReader) → bool

Test for inequality.

close (*self:* rti.connextdds.IAnyDataReader) → None

Close this DataReader.

property qos

The QoS for this AnyDataReader.

This property's getter returns a deep copy.

retain (*self:* rti.connextdds.IAnyDataReader) → None

Retain this DataReader.

property subscriber

The Publisher for this AnyDataReader.

property topic_name

The Topic name for this AnyDataReader.

property type_name

The type name for this AnyDataReader.

class rti.connextdds.**IAnyDataReaderSeq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.IAnyDataReaderSeq, *arg0:* rti.connextdds.IAnyDataReaderSeq) → *rti.connextdds.IAnyDataReaderSeq*

__bool__ (*self:* rti.connextdds.IAnyDataReaderSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.IAnyDataReaderSeq, *x:* rti.connextdds.IAnyDataReader) → bool

Return true the container contains x

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.IAnyDataReaderSeq, arg0: int) -> None

Delete the list elements at index i

2. __delitem__(self: rti.connextdds.IAnyDataReaderSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.IAnyDataReaderSeq, *arg0:* rti.connextdds.IAnyDataReaderSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.IAnyDataReaderSeq, s: slice) -> rti.connextdds.IAnyDataReaderSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.IAnyDataReaderSeq, arg0: int) -> rti.connextdds.IAnyDataReader

__hash__ = None

__iadd__(*self:* rti.connextdds.IAnyDataReaderSeq, *arg0:* rti.connextdds.IAnyDataReaderSeq) → *rti.connextdds.IAnyDataReaderSeq*

__imul__(*self:* rti.connextdds.IAnyDataReaderSeq, *arg0: int*) → *rti.connextdds.IAnyDataReaderSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.IAnyDataReaderSeq) -> None

2. __init__(self: rti.connextdds.IAnyDataReaderSeq, arg0: rti.connextdds.IAnyDataReaderSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.IAnyDataReaderSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.IAnyDataReaderSeq) → Iterator

__len__(*self:* rti.connextdds.IAnyDataReaderSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.IAnyDataReaderSeq, *arg0: int*) → *rti.connextdds.IAnyDataReaderSeq*

__ne__(*self:* rti.connextdds.IAnyDataReaderSeq, *arg0:* rti.connextdds.IAnyDataReaderSeq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__(*self:* rti.connextdds.IAnyDataReaderSeq) → str

Return the canonical string representation of this list.

__rmul__(*self:* rti.connextdds.IAnyDataReaderSeq, *arg0: int*) → *rti.connextdds.IAnyDataReaderSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.IAnyDataReaderSeq, arg0: int, arg1: rti.connextdds.IAnyDataReader) -> None

2. __setitem__(self: rti.connextdds.IAnyDataReaderSeq, arg0: slice, arg1: rti.connextdds.IAnyDataReaderSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.IAnyDataReaderSeq, *x:* rti.connextdds.IAnyDataReader) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.IAnyDataReaderSeq) → None

Clear the contents

count (*self:* rti.connextdds.IAnyDataReaderSeq, *x:* rti.connextdds.IAnyDataReader) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.IAnyDataReaderSeq, L: rti.connextdds.IAnyDataReaderSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.IAnyDataReaderSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.IAnyDataReaderSeq, *i: int*, *x:* rti.connextdds.IAnyDataReader) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.IAnyDataReaderSeq) -> rti.connextdds.IAnyDataReader

Remove and return the last item

2. pop(self: rti.connextdds.IAnyDataReaderSeq, i: int) -> rti.connextdds.IAnyDataReader

Remove and return the item at index `i`

remove (*self:* rti.connextdds.IAnyDataReaderSeq, *x:* rti.connextdds.IAnyDataReader) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**IAnyDataWriter**

Bases: `pybind11_object`

__eq__ (*self:* rti.connextdds.IAnyDataWriter, *arg0:* rti.connextdds.IAnyDataWriter) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.IAnyDataWriter, *arg0:* rti.connextdds.IAnyDataWriter) → bool

Test for inequality.

close (*self:* rti.connextdds.IAnyDataWriter) → None

Close this DataWriter.

property publisher

The Publisher for this AnyDataWriter.

property qos

The QoS for this AnyDataWriter.

This property's getter returns a deep copy.

retain (*self:* rti.connextdds.IAnyDataWriter) → None

Retain this DataWriter.

property topic_name

The Topic name for this AnyDataWriter.

property type_name

The type name for this AnyDataWriter.

wait_for_acknowledgments (*self:* rti.connextdds.IAnyDataWriter, *timeout:* rti.connextdds.Duration) → None

Wait for acknowledgments from subscribers.

class rti.connextdds.**IAnyDataWriterSeq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.IAnyDataWriterSeq, *arg0:* rti.connextdds.IAnyDataWriterSeq) → *rti.connextdds.IAnyDataWriterSeq*

__bool__ (*self:* rti.connextdds.IAnyDataWriterSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.IAnyDataWriterSeq, *x:* rti.connextdds.IAnyDataWriter) → bool

Return true the container contains x

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.IAnyDataWriterSeq, arg0: int) -> None

Delete the list elements at index i

2. __delitem__(self: rti.connextdds.IAnyDataWriterSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.IAnyDataWriterSeq, *arg0:* rti.connextdds.IAnyDataWriterSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.IAnyDataWriterSeq, s: slice) -> rti.connextdds.IAnyDataWriterSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.IAnyDataWriterSeq, arg0: int) -> rti.connextdds.IAnyDataWriter

__hash__ = None

__iadd__ (*self:* rti.connextdds.IAnyDataWriterSeq, *arg0:* rti.connextdds.IAnyDataWriterSeq) → *rti.connextdds.IAnyDataWriterSeq*

__imul__ (*self:* rti.connextdds.IAnyDataWriterSeq, *arg0: int*) → *rti.connextdds.IAnyDataWriterSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.IAnyDataWriterSeq) -> None

2. __init__(self: rti.connextdds.IAnyDataWriterSeq, arg0: rti.connextdds.IAnyDataWriterSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.IAnyDataWriterSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.IAnyDataWriterSeq) → Iterator

__len__ (*self:* rti.connextdds.IAnyDataWriterSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.IAnyDataWriterSeq, *arg0: int*) → *rti.connextdds.IAnyDataWriterSeq*

__ne__ (*self:* rti.connextdds.IAnyDataWriterSeq, *arg0:* rti.connextdds.IAnyDataWriterSeq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__ (*self:* rti.connextdds.IAnyDataWriterSeq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.IAnyDataWriterSeq, *arg0: int*) → *rti.connextdds.IAnyDataWriterSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.IAnyDataWriterSeq, arg0: int, arg1: rti.connextdds.IAnyDataWriter) -> None

2. __setitem__(self: rti.connextdds.IAnyDataWriterSeq, arg0: slice, arg1: rti.connextdds.IAnyDataWriterSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.IAnyDataWriterSeq, *x:* rti.connextdds.IAnyDataWriter) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.IAnyDataWriterSeq) → None

Clear the contents

count (*self:* rti.connextdds.IAnyDataWriterSeq, *x:* rti.connextdds.IAnyDataWriter) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.IAnyDataWriterSeq, L: rti.connextdds.IAnyDataWriterSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.IAnyDataWriterSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.IAnyDataWriterSeq, *i: int*, *x:* rti.connextdds.IAnyDataWriter) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.IAnyDataWriterSeq) -> rti.connextdds.IAnyDataWriter

Remove and return the last item

2. pop(self: rti.connextdds.IAnyDataWriterSeq, i: int) -> rti.connextdds.IAnyDataWriter

Remove and return the item at index `i`

remove (*self:* rti.connextdds.IAnyDataWriterSeq, *x:* rti.connextdds.IAnyDataWriter) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class `rti.connextdds.`**IAnyTopic**

Bases: `pybind11_object`

__eq__ (*self:* rti.connextdds.IAnyTopic, *arg0:* rti.connextdds.IAnyTopic) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.IAnyTopic, *arg0:* rti.connextdds.IAnyTopic) → bool

Test for inequality.

close (*self:* rti.connextdds.IAnyTopic) → None

Close this Topic.

property name

The Topic name for this AnyTopic.

property qos

The QoS for this AnyTopic.

This property's getter returns a deep copy.

property type_name

The type name for this AnyTopic.

class rti.connextdds.**IAnyTopicSeq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.IAnyTopicSeq, *arg0:* rti.connextdds.IAnyTopicSeq) → *rti.connextdds.IAnyTopicSeq*

__bool__ (*self:* rti.connextdds.IAnyTopicSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.IAnyTopicSeq, *x:* rti.connextdds.IAnyTopic) → bool

Return true the container contains x

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.IAnyTopicSeq, arg0: int) -> None

Delete the list elements at index i

2. __delitem__(self: rti.connextdds.IAnyTopicSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.IAnyTopicSeq, *arg0:* rti.connextdds.IAnyTopicSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.IAnyTopicSeq, s: slice) -> rti.connextdds.IAnyTopicSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.IAnyTopicSeq, arg0: int) -> rti.connextdds.IAnyTopic

__hash__ = None

__iadd__ (*self:* rti.connextdds.IAnyTopicSeq, *arg0:* rti.connextdds.IAnyTopicSeq) → *rti.connextdds.IAnyTopicSeq*

__imul__ (*self:* rti.connextdds.IAnyTopicSeq, *arg0: int*) → *rti.connextdds.IAnyTopicSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.IAnyTopicSeq) -> None

2. __init__(self: rti.connextdds.IAnyTopicSeq, arg0: rti.connextdds.IAnyTopicSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.IAnyTopicSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.IAnyTopicSeq) → Iterator

__len__ (*self:* rti.connextdds.IAnyTopicSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.IAnyTopicSeq, *arg0: int*) → *rti.connextdds.IAnyTopicSeq*

__ne__ (*self:* rti.connextdds.IAnyTopicSeq, *arg0:* rti.connextdds.IAnyTopicSeq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__ (*self:* rti.connextdds.IAnyTopicSeq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.IAnyTopicSeq, *arg0: int*) → *rti.connextdds.IAnyTopicSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.IAnyTopicSeq, arg0: int, arg1: rti.connextdds.IAnyTopic) -> None

2. __setitem__(self: rti.connextdds.IAnyTopicSeq, arg0: slice, arg1: rti.connextdds.IAnyTopicSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.IAnyTopicSeq, *x:* rti.connextdds.IAnyTopic) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.IAnyTopicSeq) → None

Clear the contents

count (*self:* rti.connextdds.IAnyTopicSeq, *x:* rti.connextdds.IAnyTopic) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.IAnyTopicSeq, L: rti.connextdds.IAnyTopicSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.IAnyTopicSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.IAnyTopicSeq, *i: int*, *x:* rti.connextdds.IAnyTopic) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.IAnyTopicSeq) -> rti.connextdds.IAnyTopic

Remove and return the last item

2. pop(self: rti.connextdds.IAnyTopicSeq, i: int) -> rti.connextdds.IAnyTopic

Remove and return the item at index `i`

remove (*self:* rti.connextdds.IAnyTopicSeq, *x:* rti.connextdds.IAnyTopic) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**ICondition**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.ICondition, *arg0:* rti.connextdds.ICondition) → bool

Compare Condition objects for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ICondition, *arg0:* rti.connextdds.ICondition) → bool

Compare Condition objects for inequality.

dispatch (*self:* rti.connextdds.ICondition) → None

Dispatches the functions registered with the condition.

property trigger_value

The trigger value of the condition.

class rti.connextdds.**IConditionSeq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.IConditionSeq, *arg0:* rti.connextdds.IConditionSeq) → *rti.connextdds.IConditionSeq*

__bool__ (*self:* rti.connextdds.IConditionSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.IConditionSeq, *x:* rti.connextdds.ICondition) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.IConditionSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.IConditionSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.IConditionSeq, *arg0:* rti.connextdds.IConditionSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.IConditionSeq, s: slice) -> rti.connextdds.IConditionSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.IConditionSeq, arg0: int) -> rti.connextdds.ICondition

__hash__ = None

__iadd__ (*self:* rti.connextdds.IConditionSeq, *arg0:* rti.connextdds.IConditionSeq) → *rti.connextdds.IConditionSeq*

__imul__ (*self:* rti.connextdds.IConditionSeq, *arg0: int*) → *rti.connextdds.IConditionSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.IConditionSeq) -> None

2. __init__(self: rti.connextdds.IConditionSeq, arg0: rti.connextdds.IConditionSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.IConditionSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.IConditionSeq) → Iterator

__len__ (*self:* rti.connextdds.IConditionSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.IConditionSeq, *arg0: int*) → *rti.connextdds.IConditionSeq*

__ne__ (*self:* rti.connextdds.IConditionSeq, *arg0:* rti.connextdds.IConditionSeq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__ (*self:* rti.connextdds.IConditionSeq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.IConditionSeq, *arg0: int*) → *rti.connextdds.IConditionSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.IConditionSeq, arg0: int, arg1: rti.connextdds.ICondition) -> None

2. __setitem__(self: rti.connextdds.IConditionSeq, arg0: slice, arg1: rti.connextdds.IConditionSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.IConditionSeq, *x:* rti.connextdds.ICondition) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.IConditionSeq) → None

Clear the contents

count (*self:* rti.connextdds.IConditionSeq, *x:* rti.connextdds.ICondition) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.IConditionSeq, L: rti.connextdds.IConditionSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.IConditionSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.IConditionSeq, *i: int*, *x:* rti.connextdds.ICondition) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.IConditionSeq) -> rti.connextdds.ICondition

Remove and return the last item

2. pop(self: rti.connextdds.IConditionSeq, i: int) -> rti.connextdds.ICondition

Remove and return the item at index `i`

remove (*self:* rti.connextdds.IConditionSeq, *x:* rti.connextdds.ICondition) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**IDataReader**

Bases: *IEntity*, *IAnyDataReader*

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

class rti.connextdds.**IEntity**

Bases: pybind11_object

This is the abstract base class for all the DDS objects that support QoS policies, a listener and a status condition.

__eq__(*self:* rti.connextdds.IEntity, *arg0:* rti.connextdds.IEntity) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.IEntity, *arg0:* rti.connextdds.IEntity) → bool

Test for inequality.

close(*self:* rti.connextdds.IEntity) → None

Forces the destruction of this entity.

property closed

Returns a boolean indicating if this Entity is closed.

enable(*self:* rti.connextdds.IEntity) → None

Enables this entity (if it was created disabled).

property enabled

Returns a boolean indicating if this Entity is enabled.

property instance_handle

The instance handle that represents this entity.

retain(*self:* rti.connextdds.IEntity) → None

Disables the automatic destruction of this entity.

property status_changes

The list of communication statuses that are triggered.

unretain(*self:* rti.connextdds.IEntity) → None

Decrement the retention count.

property use_count

Returns the internal use count value for this Entity.

class rti.connextdds.**IEntitySeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.IEntitySeq, *arg0:* rti.connextdds.IEntitySeq) → *rti.connextdds.IEntitySeq*

__bool__ (*self:* rti.connextdds.IEntitySeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.IEntitySeq, *x:* rti.connextdds.IEntity) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.IEntitySeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.IEntitySeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.IEntitySeq, *arg0:* rti.connextdds.IEntitySeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.IEntitySeq, s: slice) -> rti.connextdds.IEntitySeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.IEntitySeq, arg0: int) -> rti.connextdds.IEntity

__hash__ = None

__iadd__ (*self:* rti.connextdds.IEntitySeq, *arg0:* rti.connextdds.IEntitySeq) → *rti.connextdds.IEntitySeq*

__imul__ (*self:* rti.connextdds.IEntitySeq, *arg0: int*) → *rti.connextdds.IEntitySeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.IEntitySeq) -> None

2. __init__(self: rti.connextdds.IEntitySeq, arg0: rti.connextdds.IEntitySeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.IEntitySeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.IEntitySeq) → Iterator

__len__ (*self:* rti.connextdds.IEntitySeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.IEntitySeq, *arg0: int*) → *rti.connextdds.IEntitySeq*

__ne__ (*self:* rti.connextdds.IEntitySeq, *arg0:* rti.connextdds.IEntitySeq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__(*self:* rti.connextdds.IEntitySeq) → str

Return the canonical string representation of this list.

__rmul__(*self:* rti.connextdds.IEntitySeq, *arg0: int*) → *rti.connextdds.IEntitySeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.IEntitySeq, arg0: int, arg1: rti.connextdds.IEntity) -> None
2. __setitem__(self: rti.connextdds.IEntitySeq, arg0: slice, arg1: rti.connextdds.IEntitySeq) -> None

Assign list elements using a slice object

append(*self:* rti.connextdds.IEntitySeq, *x:* rti.connextdds.IEntity) → None

Add an item to the end of the list

clear(*self:* rti.connextdds.IEntitySeq) → None

Clear the contents

count(*self:* rti.connextdds.IEntitySeq, *x:* rti.connextdds.IEntity) → int

Return the number of times `x` appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.IEntitySeq, L: rti.connextdds.IEntitySeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.IEntitySeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert(*self:* rti.connextdds.IEntitySeq, *i: int*, *x:* rti.connextdds.IEntity) → None

Insert an item at a given position.

pop(**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.IEntitySeq) -> rti.connextdds.IEntity

Remove and return the last item

2. pop(self: rti.connextdds.IEntitySeq, i: int) -> rti.connextdds.IEntity

Remove and return the item at index `i`

remove(*self:* rti.connextdds.IEntitySeq, *x:* rti.connextdds.IEntity) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**IReadCondition**

Bases: *ICondition*

__enter__(*self:* rti.connextdds.IReadCondition) → *rti.connextdds.IReadCondition*

__eq__(*self:* rti.connextdds.IReadCondition, *arg0:* rti.connextdds.IReadCondition) → bool

Test for equality.

__exit__(*self:* rti.connextdds.IReadCondition, *arg0: object*, *arg1: object*, *arg2: object*) → None

__hash__ = None

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.IReadCondition, *arg0:* rti.connextdds.IReadCondition) → bool

Test for inequality.

close(*self:* rti.connextdds.IReadCondition) → None

Returns the DataReader associated to this condition.

property closed

Returns the DataReader associated to this condition.

property data_reader

Returns the DataReader associated to this condition.

property state_filter

Returns the DataState of this condition.

class rti.connextdds.**ITopicDescription**

Bases: *IEntity*

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property name

The name of the entity conforming to the ITopicDescription interface.

property participant

The parent DomainParticipant.

property type_name

The name of the associated type.

class rti.connextdds.**IgnoredEntityReplacementKind**

Bases: pybind11_object

class IgnoredEntityReplacementKind

Bases: pybind11_object

Members:

NO_REPLACEMENT

NOT_ALIVE_FIRST

NOT_ALIVE_FIRST = <IgnoredEntityReplacementKind.NOT_ALIVE_FIRST: 1>

NO_REPLACEMENT = <IgnoredEntityReplacementKind.NO_REPLACEMENT: 0>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.IgnoredEntityReplacementKind.IgnoredEntityReplacementKind) → int

__init__(*self:* rti.connextdds.IgnoredEntityReplacementKind.IgnoredEntityReplacementKind, *value: int*) → None

__int__(*self:* rti.connextdds.IgnoredEntityReplacementKind.IgnoredEntityReplacementKind) → int

__members__ = {'NOT_ALIVE_FIRST': <IgnoredEntityReplacementKind.NOT_ALIVE_FIRST: 1>, 'NO_REPLACEMENT': <IgnoredEntityReplacementKind.NO_REPLACEMENT: 0>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.IgnoredEntityReplacementKind.IgnoredEntityReplacementKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

NOT_ALIVE_FIRST = <IgnoredEntityReplacementKind.NOT_ALIVE_FIRST: 1>

NO_REPLACEMENT = <IgnoredEntityReplacementKind.NO_REPLACEMENT: 0>

__eq__ (*self:* rti.connextdds.IgnoredEntityReplacementKind, *arg0:* rti.connextdds.IgnoredEntityReplacementKind) → bool

Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.IgnoredEntityReplacementKind, *arg0:* rti.connextdds.IgnoredEntityReplacementKind) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.IgnoredEntityReplacementKind, *arg0:* rti.connextdds.IgnoredEntityReplacementKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.IgnoredEntityReplacementKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.IgnoredEntityReplacementKind, arg0: rti.connextdds.IgnoredEntityReplacementKind.IgnoredEntityReplacementKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.IgnoredEntityReplacementKind) → *rti.connextdds.IgnoredEntityReplacementKind.IgnoredEntityReplacementKind*

Int conversion.

__le__ (*self:* rti.connextdds.IgnoredEntityReplacementKind, *arg0:* rti.connextdds.IgnoredEntityReplacementKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.IgnoredEntityReplacementKind, *arg0:* rti.connextdds.IgnoredEntityReplacementKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.IgnoredEntityReplacementKind, *arg0:* rti.connextdds.IgnoredEntityReplacementKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.IgnoredEntityReplacementKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

exception rti.connextdds.**IllegalOperationError**

Bases: *Exception*

__module__ = 'rti.connextdds'

exception rti.connextdds.**ImmutablePolicyError**

Bases: *Exception*

__module__ = 'rti.connextdds'

exception rti.connextdds.**InconsistentPolicyError**

Bases: *Exception*

__module__ = 'rti.connextdds'

class rti.connextdds.**InconsistentTopicStatus**

Bases: pybind11_object

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property total_count

Get the total count of pairs of DataReaders/DataWriters whose Topic names match but data types are inconsistent according to the current type consistency enforcement policy.

property total_count_change

The delta number of inconsistent pairs of DataReaders/DataWriters for the Topic that have been discovered since the last time this status was read.

class rti.connextdds.**InstanceHandle**

Bases: pybind11_object

__bool__(*self:* rti.connextdds.InstanceHandle) → bool

__eq__(*self:* rti.connextdds.InstanceHandle, *arg0:* rti.connextdds.InstanceHandle) → bool

Test for equality.

__hash__(*self:* rti.connextdds.InstanceHandle) → int

__init__(*self:* rti.connextdds.InstanceHandle) → None

Create a nil InstanceHandle

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.InstanceHandle, *arg0:* rti.connextdds.InstanceHandle) → bool

Test for inequality.

__nonzero__(*self:* rti.connextdds.InstanceHandle) → bool

__str__(*self:* rti.connextdds.InstanceHandle) → str

property is_nil

Nil status of InstanceStatus

static nil() → *rti.connextdds.InstanceHandle*

Create a nil InstanceHandle.

class rti.connextdds.**InstanceHandleSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.InstanceHandleSeq, *arg0:* rti.connextdds.InstanceHandleSeq) → *rti.connextdds.InstanceHandleSeq*

__bool__(*self:* rti.connextdds.InstanceHandleSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.InstanceHandleSeq, *x:* rti.connextdds.InstanceHandle) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.InstanceHandleSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.InstanceHandleSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.InstanceHandleSeq, *arg0:* rti.connextdds.InstanceHandleSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.InstanceHandleSeq, s: slice) -> rti.connextdds.InstanceHandleSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.InstanceHandleSeq, arg0: int) -> rti.connextdds.InstanceHandle

__hash__ = None

__iadd__(*self:* rti.connextdds.InstanceHandleSeq, *arg0:* rti.connextdds.InstanceHandleSeq) → *rti.connextdds.InstanceHandleSeq*

__imul__(*self:* rti.connextdds.InstanceHandleSeq, *arg0: int*) → *rti.connextdds.InstanceHandleSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.InstanceHandleSeq) -> None

2. __init__(self: rti.connextdds.InstanceHandleSeq, arg0: rti.connextdds.InstanceHandleSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.InstanceHandleSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.InstanceHandleSeq) → Iterator

__len__ (*self:* rti.connextdds.InstanceHandleSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.InstanceHandleSeq, *arg0: int*) → *rti.connextdds.InstanceHandleSeq*

__ne__ (*self:* rti.connextdds.InstanceHandleSeq, *arg0:* rti.connextdds.InstanceHandleSeq) → bool

__repr__ (*self:* rti.connextdds.InstanceHandleSeq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.InstanceHandleSeq, *arg0: int*) → *rti.connextdds.InstanceHandleSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.InstanceHandleSeq, arg0: int, arg1: rti.connextdds.InstanceHandle) -> None

2. __setitem__(self: rti.connextdds.InstanceHandleSeq, arg0: slice, arg1: rti.connextdds.InstanceHandleSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.InstanceHandleSeq, *x:* rti.connextdds.InstanceHandle) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.InstanceHandleSeq) → None

Clear the contents

count (*self:* rti.connextdds.InstanceHandleSeq, *x:* rti.connextdds.InstanceHandle) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.InstanceHandleSeq, L: rti.connextdds.InstanceHandleSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.InstanceHandleSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.InstanceHandleSeq, *i: int*, *x:* rti.connextdds.InstanceHandle) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.InstanceHandleSeq) -> rti.connextdds.InstanceHandle

Remove and return the last item

2. pop(self: rti.connextdds.InstanceHandleSeq, i: int) -> rti.connextdds.InstanceHandle

Remove and return the item at index `i`

remove (*self:* rti.connextdds.InstanceHandleSeq, *x:* rti.connextdds.InstanceHandle) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**InstanceState**

Bases: `pybind11_object`

ALIVE = <rti.connextdds.InstanceState object>

ANY = <rti.connextdds.InstanceState object>

NOT_ALIVE_DISPOSED = <rti.connextdds.InstanceState object>

NOT_ALIVE_MASK = <rti.connextdds.InstanceState object>

NOT_ALIVE_NO_WRITERS = <rti.connextdds.InstanceState object>

__and__ (*self:* rti.connextdds.InstanceState, *arg0:* rti.connextdds.InstanceState) → *rti.connextdds.InstanceState*

Bitwise logical AND of masks.

__bool__ (*self:* rti.connextdds.InstanceState) → *rti.connextdds.InstanceState*

Test if any bits are set.

__contains__ (*self:* rti.connextdds.InstanceState, *arg0:* rti.connextdds.InstanceState) → bool

__eq__ (*self:* rti.connextdds.InstanceState, *arg0:* rti.connextdds.InstanceState) → bool

Compare masks for equality.

__getitem__ (*self:* rti.connextdds.InstanceState, *arg0: int*) → bool

Get individual mask bit.

__hash__ = None

__iand__ (*self:* rti.connextdds.InstanceState, *arg0:* rti.connextdds.InstanceState) → *rti.connextdds.InstanceState*

Set mask to logical AND with another mask.

__ilshift__ (*self:* rti.connextdds.InstanceState, *arg0: int*) → *rti.connextdds.InstanceState*

Left shift bits in mask.

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.InstanceState) -> None

Create an InstanceState with no bits set.

2. __init__(self: rti.connextdds.InstanceState, value: int) -> None

Creates a mask from the bits in an integer.

__int__ (*self:* rti.connextdds.InstanceState) → int

Convert mask to int.

__ior__ (*self:* rti.connextdds.InstanceState, *arg0:* rti.connextdds.InstanceState) → *rti.connextdds.InstanceState*

Set mask to logical OR with another mask.

__irshift__ (*self:* rti.connextdds.InstanceState, *arg0: int*) → *rti.connextdds.InstanceState*

Right shift bits in mask.

__ixor__ (*self:* rti.connextdds.InstanceState, *arg0:* rti.connextdds.InstanceState) → *rti.connextdds.InstanceState*

Set mask to logical XOR with another mask.

__lshift__ (*self:* rti.connextdds.InstanceState, *arg0: int*) → *rti.connextdds.InstanceState*

Left shift bits in mask.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.InstanceState, *arg0:* rti.connextdds.InstanceState) → bool

Compare masks for inequality.

__or__ (*self:* rti.connextdds.InstanceState, *arg0:* rti.connextdds.InstanceState) → *rti.connextdds.InstanceState*

Bitwise logical OR of masks.

__rshift__ (*self:* rti.connextdds.InstanceState, *arg0: int*) → *rti.connextdds.InstanceState*

Right shift bits in mask.

__setitem__ (*self:* rti.connextdds.InstanceState, *arg0: int*, *arg1: bool*) → None

Set individual mask bit

__str__ (*self:* rti.connextdds.InstanceState) → str

__xor__ (*self:* rti.connextdds.InstanceState, *arg0:* rti.connextdds.InstanceState) → *rti.connextdds.InstanceState*

Bitwise logical XOR of masks.

property count

Returns the number of bits set in the mask.

flip (**args*, ***kwargs*)

Overloaded function.

1. flip(self: rti.connextdds.InstanceState) -> rti.connextdds.InstanceState

Flip all bits in the mask.

2. flip(self: rti.connextdds.InstanceState, pos: int) -> rti.connextdds.InstanceState

Flip the mask bit at the specified position.

reset (**args*, ***kwargs*)

Overloaded function.

1. reset(self: rti.connextdds.InstanceState) -> rti.connextdds.InstanceState

Clear all bits in the mask.

2. reset(self: rti.connextdds.InstanceState, pos: int) -> rti.connextdds.InstanceState

Clear the mask bit at the specified position.

set (**args*, ***kwargs*)

Overloaded function.

1. set(self: rti.connextdds.InstanceState) -> rti.connextdds.InstanceState

Set all bits in the mask.

2. set(self: rti.connextdds.InstanceState, pos: int, value: bool = True) -> rti.connextdds.InstanceState

Set the mask bit at the specified position to the provided value (default: true).

property size

Returns the number of bits in the mask type.

test (*self:* rti.connextdds.InstanceState, *pos: int*) → bool

Test whether the mask bit at position "pos" is set.

test_all (*self:* rti.connextdds.InstanceState) → bool

Test if all bits are set.

test_any (*self:* rti.connextdds.InstanceState) → bool

Test if any bits are set.

test_none (*self:* rti.connextdds.InstanceState) → bool

Test if none of the bits are set.

class rti.connextdds.**InstanceStateConsistencyKind**

Bases: pybind11_object

Members:

NONE : Instance states on the DataReader are not guaranteed to be correct after liveliness is regained after a disconnect

RECOVER_STATE : Instance states on the DataReader are guaranteed to be correct after liveliness is regained after a disconnect.

NONE = <InstanceStateConsistencyKind.NONE: 0>

RECOVER_STATE = <InstanceStateConsistencyKind.RECOVER_STATE: 1>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.InstanceStateConsistencyKind) → int

__init__(*self:* rti.connextdds.InstanceStateConsistencyKind, *value: int*) → None

__int__(*self:* rti.connextdds.InstanceStateConsistencyKind) → int

__members__ = {'NONE': <InstanceStateConsistencyKind.NONE: 0>, 'RECOVER_STATE': <InstanceStateConsistencyKind.RECOVER_STATE: 1>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.InstanceStateConsistencyKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

class rti.connextdds.**Int16Seq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.Int16Seq, *arg0:* rti.connextdds.Int16Seq) → *rti.connextdds.Int16Seq*

__bool__(*self:* rti.connextdds.Int16Seq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.Int16Seq, *x: int*) → bool

Return true the container contains x

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.Int16Seq, arg0: int) -> None

Delete the list elements at index i

2. __delitem__(self: rti.connextdds.Int16Seq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.Int16Seq, *arg0:* rti.connextdds.Int16Seq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.Int16Seq, s: slice) -> rti.connextdds.Int16Seq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.Int16Seq, arg0: int) -> int

__getstate__ (*self:* rti.connextdds.Int16Seq) → bytes

__hash__ = None

__iadd__ (*self:* rti.connextdds.Int16Seq, *arg0:* rti.connextdds.Int16Seq) → *rti.connextdds.Int16Seq*

__imul__ (*self:* rti.connextdds.Int16Seq, *arg0: int*) → *rti.connextdds.Int16Seq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Int16Seq, arg0: buffer) -> None

2. __init__(self: rti.connextdds.Int16Seq) -> None

3. __init__(self: rti.connextdds.Int16Seq, arg0: rti.connextdds.Int16Seq) -> None

Copy constructor

4. __init__(self: rti.connextdds.Int16Seq, arg0: Iterable) -> None

5. __init__(self: rti.connextdds.Int16Seq, arg0: int) -> None

__iter__ (*self:* rti.connextdds.Int16Seq) → Iterator

__len__ (*self:* rti.connextdds.Int16Seq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.Int16Seq, *arg0: int*) → *rti.connextdds.Int16Seq*

__ne__ (*self:* rti.connextdds.Int16Seq, *arg0:* rti.connextdds.Int16Seq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__ (*self:* rti.connextdds.Int16Seq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.Int16Seq, *arg0: int*) → *rti.connextdds.Int16Seq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.Int16Seq, arg0: int, arg1: int) -> None
2. __setitem__(self: rti.connextdds.Int16Seq, arg0: slice, arg1: rti.connextdds.Int16Seq) -> None

Assign list elements using a slice object

__setstate__ (*self:* rti.connextdds.Int16Seq, *arg0: bytes*) → None

append (*self:* rti.connextdds.Int16Seq, *x: int*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.Int16Seq) → None

Clear the contents

count (*self:* rti.connextdds.Int16Seq, *x: int*) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.Int16Seq, L: rti.connextdds.Int16Seq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.Int16Seq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.Int16Seq, *i: int*, *x: int*) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.Int16Seq) -> int

Remove and return the last item

2. pop(self: rti.connextdds.Int16Seq, i: int) -> int

Remove and return the item at index `i`

remove (*self:* rti.connextdds.Int16Seq, *x: int*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

resize (*self:* rti.connextdds.Int16Seq, *arg0: int*) → None

resizes the vector to the given size

class rti.connextdds.**Int16Type**

Bases: *DynamicType*

__eq__ (*self:* rti.connextdds.Int16Type, *arg0:* rti.connextdds.Int16Type) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.Int16Type) → None

Get the singleton for Int16Type

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Int16Type, *arg0:* rti.connextdds.Int16Type) → bool

Test for inequality.

class rti.connextdds.**Int32Seq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.Int32Seq, *arg0:* rti.connextdds.Int32Seq) → *rti.connextdds.Int32Seq*

__bool__ (*self:* rti.connextdds.Int32Seq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.Int32Seq, *x: int*) → bool

Return true the container contains x

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.Int32Seq, arg0: int) -> None

Delete the list elements at index i

2. __delitem__(self: rti.connextdds.Int32Seq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.Int32Seq, *arg0:* rti.connextdds.Int32Seq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.Int32Seq, s: slice) -> rti.connextdds.Int32Seq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.Int32Seq, arg0: int) -> int

__getstate__ (*self:* rti.connextdds.Int32Seq) → bytes

__hash__ = None

__iadd__ (*self:* rti.connextdds.Int32Seq, *arg0:* rti.connextdds.Int32Seq) → *rti.connextdds.Int32Seq*

__imul__ (*self:* rti.connextdds.Int32Seq, *arg0: int*) → *rti.connextdds.Int32Seq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Int32Seq, arg0: buffer) -> None
2. __init__(self: rti.connextdds.Int32Seq) -> None
3. __init__(self: rti.connextdds.Int32Seq, arg0: rti.connextdds.Int32Seq) -> None

Copy constructor

4. __init__(self: rti.connextdds.Int32Seq, arg0: Iterable) -> None
5. __init__(self: rti.connextdds.Int32Seq, arg0: int) -> None

__iter__ (*self:* rti.connextdds.Int32Seq) → Iterator

__len__ (*self:* rti.connextdds.Int32Seq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.Int32Seq, *arg0: int*) → *rti.connextdds.Int32Seq*

__ne__ (*self:* rti.connextdds.Int32Seq, *arg0:* rti.connextdds.Int32Seq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__ (*self:* rti.connextdds.Int32Seq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.Int32Seq, *arg0: int*) → *rti.connextdds.Int32Seq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.Int32Seq, arg0: int, arg1: int) -> None
2. __setitem__(self: rti.connextdds.Int32Seq, arg0: slice, arg1: rti.connextdds.Int32Seq) -> None

Assign list elements using a slice object

__setstate__ (*self:* rti.connextdds.Int32Seq, *arg0: bytes*) → None

append (*self:* rti.connextdds.Int32Seq, *x: int*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.Int32Seq) → None

Clear the contents

count (*self:* rti.connextdds.Int32Seq, *x: int*) → int

Return the number of times x appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.Int32Seq, L: rti.connextdds.Int32Seq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.Int32Seq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert(*self:* rti.connextdds.Int32Seq, *i: int*, *x: int*) → None

Insert an item at a given position.

pop(**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.Int32Seq) -> int

Remove and return the last item

2. pop(self: rti.connextdds.Int32Seq, i: int) -> int

Remove and return the item at index `i`

remove(*self:* rti.connextdds.Int32Seq, *x: int*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

resize(*self:* rti.connextdds.Int32Seq, *arg0: int*) → None

resizes the vector to the given size

class rti.connextdds.**Int32Type**

Bases: *DynamicType*

__eq__(*self:* rti.connextdds.Int32Type, *arg0:* rti.connextdds.Int32Type) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.Int32Type) → None

Get the singleton for Int32Type

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.Int32Type, *arg0:* rti.connextdds.Int32Type) → bool

Test for inequality.

class rti.connextdds.**Int32Vector**

Bases: pybind11_object

A DDS standard container with functionality similar to a C++ vector.

__eq__(*self:* rti.connextdds.Int32Vector, *arg0:* rti.connextdds.Int32Vector) → bool

Compare Int32Vectors for equality.

__getitem__ (*self:* rti.connextdds.Int32Vector, *arg0: int*) → int

Get the value at the specified index.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Int32Vector, buffer: buffer) -> None

Create a Int32Vector from another Python buffer.

2. __init__(self: rti.connextdds.Int32Vector) -> None

Create an empty Int32Vector

3. __init__(self: rti.connextdds.Int32Vector, size: int) -> None

Create a Int32Vector with a preallocated size.

4. __init__(self: rti.connextdds.Int32Vector, vector: rti.connextdds.Int32Vector) -> None

Create a copy from another Int32Vector.

5. __init__(self: rti.connextdds.Int32Vector, arg0: Iterable) -> None

6. __init__(self: rti.connextdds.Int32Vector, list: rti.connextdds.Int32Seq) -> None

Create a Int32Vector from a list of values.

__iter__ (*self:* rti.connextdds.Int32Vector) → Iterator

Iterate over the contents of the vector.

__len__ (*self:* rti.connextdds.Int32Vector) → int

Get the length of the Int32Vector.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Int32Vector, *arg0:* rti.connextdds.Int32Vector) → bool

Compare Int32Vectors for inequality.

__setitem__ (*self:* rti.connextdds.Int32Vector, *arg0: int*, *arg1: int*) → None

Set the value at the specified index.

clear (*self:* rti.connextdds.Int32Vector) → None

Resize Int32Vector to 0.

resize (*self:* rti.connextdds.Int32Vector, *size: int*) → None

Resize Int32Vector.

class rti.connextdds.**Int64Seq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.Int64Seq, *arg0:* rti.connextdds.Int64Seq) → *rti.connextdds.Int64Seq*

__bool__ (*self:* rti.connextdds.Int64Seq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.Int64Seq, *x: int*) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.Int64Seq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.Int64Seq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.Int64Seq, *arg0:* rti.connextdds.Int64Seq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.Int64Seq, s: slice) -> rti.connextdds.Int64Seq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.Int64Seq, arg0: int) -> int

__getstate__ (*self:* rti.connextdds.Int64Seq) → bytes

__hash__ = None

__iadd__ (*self:* rti.connextdds.Int64Seq, *arg0:* rti.connextdds.Int64Seq) → *rti.connextdds.Int64Seq*

__imul__ (*self:* rti.connextdds.Int64Seq, *arg0: int*) → *rti.connextdds.Int64Seq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Int64Seq, arg0: buffer) -> None
2. __init__(self: rti.connextdds.Int64Seq) -> None
3. __init__(self: rti.connextdds.Int64Seq, arg0: rti.connextdds.Int64Seq) -> None

Copy constructor

4. __init__(self: rti.connextdds.Int64Seq, arg0: Iterable) -> None
5. __init__(self: rti.connextdds.Int64Seq, arg0: int) -> None

__iter__ (*self:* rti.connextdds.Int64Seq) → Iterator

__len__ (*self:* rti.connextdds.Int64Seq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.Int64Seq, *arg0: int*) → *rti.connextdds.Int64Seq*

__ne__ (*self:* rti.connextdds.Int64Seq, *arg0:* rti.connextdds.Int64Seq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__ (*self:* rti.connextdds.Int64Seq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.Int64Seq, *arg0: int*) → *rti.connextdds.Int64Seq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.Int64Seq, arg0: int, arg1: int) -> None
2. __setitem__(self: rti.connextdds.Int64Seq, arg0: slice, arg1: rti.connextdds.Int64Seq) -> None

Assign list elements using a slice object

__setstate__ (*self:* rti.connextdds.Int64Seq, *arg0: bytes*) → None

append (*self:* rti.connextdds.Int64Seq, *x: int*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.Int64Seq) → None

Clear the contents

count (*self:* rti.connextdds.Int64Seq, *x: int*) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.Int64Seq, L: rti.connextdds.Int64Seq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.Int64Seq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.Int64Seq, *i: int*, *x: int*) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.Int64Seq) -> int

Remove and return the last item

2. pop(self: rti.connextdds.Int64Seq, i: int) -> int

Remove and return the item at index `i`

remove (*self:* rti.connextdds.Int64Seq, *x: int*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

resize (*self:* rti.connextdds.Int64Seq, *arg0: int*) → None

resizes the vector to the given size

class rti.connextdds.**Int64Type**

Bases: *DynamicType*

__eq__ (*self:* rti.connextdds.Int64Type, *arg0:* rti.connextdds.Int64Type) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.Int64Type) → None

Get the singleton for Int64Type

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Int64Type, *arg0:* rti.connextdds.Int64Type) → bool

Test for inequality.

class rti.connextdds.**Int8Seq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.Int8Seq, *arg0:* rti.connextdds.Int8Seq) → *rti.connextdds.Int8Seq*

__bool__ (*self:* rti.connextdds.Int8Seq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.Int8Seq, *x: int*) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.Int8Seq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.Int8Seq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.Int8Seq, *arg0:* rti.connextdds.Int8Seq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.Int8Seq, s: slice) -> rti.connextdds.Int8Seq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.Int8Seq, arg0: int) -> int

__getstate__ (*self:* rti.connextdds.Int8Seq) → bytes

__hash__ = None

__iadd__ (*self:* rti.connextdds.Int8Seq, *arg0:* rti.connextdds.Int8Seq) → *rti.connextdds.Int8Seq*

__imul__ (*self:* rti.connextdds.Int8Seq, *arg0: int*) → *rti.connextdds.Int8Seq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Int8Seq, arg0: buffer) -> None
2. __init__(self: rti.connextdds.Int8Seq) -> None
3. __init__(self: rti.connextdds.Int8Seq, arg0: rti.connextdds.Int8Seq) -> None

Copy constructor

4. __init__(self: rti.connextdds.Int8Seq, arg0: Iterable) -> None
5. __init__(self: rti.connextdds.Int8Seq, arg0: int) -> None

__iter__ (*self:* rti.connextdds.Int8Seq) → Iterator

__len__ (*self:* rti.connextdds.Int8Seq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.Int8Seq, *arg0: int*) → *rti.connextdds.Int8Seq*

__ne__ (*self:* rti.connextdds.Int8Seq, *arg0:* rti.connextdds.Int8Seq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__ (*self:* rti.connextdds.Int8Seq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.Int8Seq, *arg0: int*) → *rti.connextdds.Int8Seq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.Int8Seq, arg0: int, arg1: int) -> None
2. __setitem__(self: rti.connextdds.Int8Seq, arg0: slice, arg1: rti.connextdds.Int8Seq) -> None

Assign list elements using a slice object

__setstate__ (*self:* rti.connextdds.Int8Seq, *arg0: bytes*) → None

append (*self:* rti.connextdds.Int8Seq, *x: int*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.Int8Seq) → None

Clear the contents

count (*self:* rti.connextdds.Int8Seq, *x: int*) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.Int8Seq, L: rti.connextdds.Int8Seq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.Int8Seq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.Int8Seq, *i: int*, *x: int*) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.Int8Seq) -> int

Remove and return the last item

2. pop(self: rti.connextdds.Int8Seq, i: int) -> int

Remove and return the item at index `i`

remove (*self:* rti.connextdds.Int8Seq, *x: int*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

resize (*self:* rti.connextdds.Int8Seq, *arg0: int*) → None

resizes the vector to the given size

class rti.connextdds.**Int8Type**

Bases: *DynamicType*

__eq__ (*self:* rti.connextdds.Int8Type, *arg0:* rti.connextdds.Int8Type) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.Int8Type) → None

Get the singleton for Int8Type

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Int8Type, *arg0:* rti.connextdds.Int8Type) → bool

Test for inequality.

exception rti.connextdds.**InvalidArgumentError**

Bases: *Exception*

__module__ = 'rti.connextdds'

exception rti.connextdds.**InvalidDowncastError**

Bases: *Exception*

__module__ = 'rti.connextdds'

class rti.connextdds.**InvalidLocalIdentityAdvanceNoticeStatus**

Bases: pybind11_object

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property expiration_time

Get the expiration time of the status

class rti.connextdds.**LatencyBudget**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.LatencyBudget, *arg0:* rti.connextdds.LatencyBudget) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.LatencyBudget) -> None

Creates a latency budget with zero duration.

2. __init__(self: rti.connextdds.LatencyBudget, duration: rti.connextdds.Duration) -> None

Creates an instance with the specified duration.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.LatencyBudget, *arg0:* rti.connextdds.LatencyBudget) → bool

Test for inequality.

property duration

The duration of the maximum acceptable delay.

This property's getter returns a deep copy.

class rti.connextdds.**Lifespan**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.Lifespan, *arg0:* rti.connextdds.Lifespan) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Lifespan) -> None

Creates the default policy with an infinite lifespan.

2. __init__(self: rti.connextdds.Lifespan, duration: rti.connextdds.Duration) -> None

Creates a policy with the specified lifespan duration.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Lifespan, *arg0:* rti.connextdds.Lifespan) → bool

Test for inequality.

property duration

Sets the maximum duration for which the data is valid.

This property's getter returns a deep copy.

class rti.connextdds.**Liveliness**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.Liveliness, *arg0:* rti.connextdds.Liveliness) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Liveliness) -> None

Creates an automatic liveliness policy with infinite lease duration.

2. __init__(self: rti.connextdds.Liveliness, kind: rti.connextdds.LivelinessKind, duration: rti.connextdds.Duration) -> None

Creates an instance with the specified liveliness kind and lease duration.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Liveliness, *arg0:* rti.connextdds.Liveliness) → bool

Test for inequality.

property assertions_per_lease_duration

The number of assertions to send during the lease duration.

automatic = <rti.connextdds.Liveliness object>

property kind

The liveliness kind.

property lease_duration

The duration within which a Entity must be asserted or else it is considered not alive.

This property's getter returns a deep copy.

static manual_by_participant(*lease:* rti.connextdds.Duration = *Duration.infinite*) → *rti.connextdds.Liveliness*

Creates a Liveliness instance with LivelinessKind.MANUAL_BY_PARTICIPANT.

static manual_by_topic(*lease:* rti.connextdds.Duration = *Duration.infinite*) → *rti.connextdds.Liveliness*

Creates a Liveliness instance with LivelinessKind.MANUAL_BY_TOPIC.

class rti.connextdds.**LivelinessChangedStatus**

Bases: pybind11_object

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property alive_count

The number of currently alive DataWriters that write to the Topic of the DataReader.

property alive_count_change

The delta in the alive count since the last time the listener fired or the status was read.

property last_publication_handle

The instance handle of the DataWriter with the most recent change in liveliness.

property not_alive_count

The number of currently NOT_ALIVE DataWriters that write to the Topic of the DataReader.

property not_alive_count_change

The delta in the NOT_ALIVE count since the last time the listener fired or the status was read.

class rti.connextdds.**LivelinessKind**

Bases: pybind11_object

AUTOMATIC = <LivelinessKind.AUTOMATIC: 0>

class LivelinessKind

Bases: pybind11_object

Members:

AUTOMATIC : [default] The infrastructure will automatically signal liveliness for the DataWriter (s) at least as often as required by the DataWriter (S) lease_duration.

A DataWriter with this setting does not need to take any specific action in order to be considered 'alive.' The DataWriter is only 'not alive' when the participant to which it belongs terminates (gracefully or not), or when there is a network problem that prevents the current participant from contacting that remote participant.

MANUAL_BY_PARTICIPANT : RTI Connext will assume that as long as at least one DataWriter belonging to the DomainParticipant (or the DomainParticipant itself) has asserted its liveliness, then the other DataWriters belonging to that same DomainParticipant are also alive.

The user application takes responsibility to signal liveliness to RTI Connext either by calling DomainParticipant.assert_liveliness, or by calling DataWriter.assert_liveliness, or DataWriter.write() on any DataWriter belonging to the DomainParticipant.

MANUAL_BY_TOPIC : RTI Connext will only assume liveliness of the DataWriter if the application has asserted liveliness of that DataWriter itself.

The user application takes responsibility to signal liveliness to RTI Connext using the DataWriter.assert_liveliness method, or by writing some data.

AUTOMATIC = <LivelinessKind.AUTOMATIC: 0>

MANUAL_BY_PARTICIPANT = <LivelinessKind.MANUAL_BY_PARTICIPANT: 1>

MANUAL_BY_TOPIC = <LivelinessKind.MANUAL_BY_TOPIC: 2>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.LivelinessKind.LivelinessKind) → int

__init__(*self:* rti.connextdds.LivelinessKind.LivelinessKind, *value: int*) → None

__int__(*self:* rti.connextdds.LivelinessKind.LivelinessKind) → int

__members__ = {'AUTOMATIC': <LivelinessKind.AUTOMATIC: 0>, 'MANUAL_BY_PARTICIPANT': <LivelinessKind.MANUAL_BY_PARTICIPANT: 1>, 'MANUAL_BY_TOPIC': <LivelinessKind.MANUAL_BY_TOPIC: 2>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.LivelinessKind.LivelinessKind, *state: int*) → None

__str__()
name(self: handle) -> str

property name

property value

MANUAL_BY_PARTICIPANT = <LivelinessKind.MANUAL_BY_PARTICIPANT: 1>

MANUAL_BY_TOPIC = <LivelinessKind.MANUAL_BY_TOPIC: 2>

__eq__ (*self:* rti.connextdds.LivelinessKind, *arg0:* rti.connextdds.LivelinessKind) → bool

Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.LivelinessKind, *arg0:* rti.connextdds.LivelinessKind) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.LivelinessKind, *arg0:* rti.connextdds.LivelinessKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.LivelinessKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.LivelinessKind, arg0: rti.connextdds.LivelinessKind.LivelinessKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.LivelinessKind) → *rti.connextdds.LivelinessKind.LivelinessKind*

Int conversion.

__le__ (*self:* rti.connextdds.LivelinessKind, *arg0:* rti.connextdds.LivelinessKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.LivelinessKind, *arg0:* rti.connextdds.LivelinessKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.LivelinessKind, *arg0:* rti.connextdds.LivelinessKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.LivelinessKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**LivelinessLostStatus**

Bases: pybind11_object

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property total_count

Total count of times that a previously alive DataWriter became not alive due to a failure to assert its liveliness signal within the agreed lease duration.

property total_count_change

The delta number of times liveliness has been lost since the last time the listener callback fired or this status was checked.

class rti.connextdds.**LoanedDynamicData**

Bases: pybind11_object

__enter__(*self:* rti.connextdds.LoanedDynamicData) → *rti.connextdds.LoanedDynamicData*

Enter a context for the loaned field, loan returned on context exit.

__exit__(*self:* rti.connextdds.LoanedDynamicData, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for the loaned field, returning the resources.

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property data

Obtains the DynamicData object representing a member of a DynamicData object.

return_loan(*self:* rti.connextdds.LoanedDynamicData) → None

Explicitly return a dynamic data loan.

class rti.connextdds.**Locator**

Bases: pybind11_object

Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.

__eq__(*self:* rti.connextdds.Locator, *arg0:* rti.connextdds.Locator) → bool

Compare Locators for equality.

__hash__ = None

__init__(*self:* rti.connextdds.Locator, *kind:* rti.connextdds.LocatorKind.LocatorKind, *port: int*, *address:* rti.connextdds.Uint8Seq) → None

Construct a Locator with the provided kind, port, and address.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.Locator, *arg0:* rti.connextdds.Locator) → bool

Compare Locators for inequality.

property address

Get/set the address for this Locator.

invalid = <rti.connextdds.Locator object>

property kind

Get/set the kind for this Locator.

property port

Get/set the port for this Locator.

class rti.connextdds.**LocatorFilter**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.LocatorFilter, *arg0:* rti.connextdds.LocatorFilter) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.LocatorFilter) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.LocatorFilter, locator_filters: rti.connextdds.LocatorFilterElementSeq, filter_name: str = Filter.stringmatch_filter_name) -> None

Creates an instance with a sequence of filters and a filter name.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.LocatorFilter, *arg0:* rti.connextdds.LocatorFilter) → bool

Test for inequality.

property filter_name

The filter name.

property locator_filters

The locator filters.

This property's getter returns a deep copy.

class rti.connextdds.**LocatorFilterElement**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.LocatorFilterElement, *arg0:* rti.connextdds.LocatorFilterElement) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.LocatorFilterElement, *arg0: str*, *arg1:* rti.connextdds.LocatorSeq) → None

Creates an instance with the provided filter_expression and locators.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.LocatorFilterElement, *arg0:* rti.connextdds.LocatorFilterElement) → bool

Test for inequality.

property filter_expression

The filter expression.

property locators

The locators associated with this filter.

This property's getter returns a deep copy.

class rti.connextdds.**LocatorFilterElementSeq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.LocatorFilterElementSeq, *arg0:* rti.connextdds.LocatorFilterElementSeq) → *rti.connextdds.LocatorFilterElementSeq*

__bool__ (*self:* rti.connextdds.LocatorFilterElementSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.LocatorFilterElementSeq, *x:* rti.connextdds.LocatorFilterElement) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.LocatorFilterElementSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.LocatorFilterElementSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.LocatorFilterElementSeq, *arg0:* rti.connextdds.LocatorFilterElementSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.LocatorFilterElementSeq, s: slice) -> rti.connextdds.LocatorFilterElementSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.LocatorFilterElementSeq, arg0: int) -> rti.connextdds.LocatorFilterElement

__hash__ = None

__iadd__ (*self:* rti.connextdds.LocatorFilterElementSeq, *arg0:* rti.connextdds.LocatorFilterElementSeq) → *rti.connextdds.LocatorFilterElementSeq*

__imul__ (*self:* rti.connextdds.LocatorFilterElementSeq, *arg0: int*) → *rti.connextdds.LocatorFilterElementSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.LocatorFilterElementSeq) -> None
2. __init__(self: rti.connextdds.LocatorFilterElementSeq, arg0: rti.connextdds.LocatorFilterElementSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.LocatorFilterElementSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.LocatorFilterElementSeq) → Iterator

__len__ (*self:* rti.connextdds.LocatorFilterElementSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.LocatorFilterElementSeq, *arg0: int*) → *rti.connextdds.LocatorFilterElementSeq*

__ne__ (*self:* rti.connextdds.LocatorFilterElementSeq, *arg0:* rti.connextdds.LocatorFilterElementSeq) → bool

__rmul__ (*self:* rti.connextdds.LocatorFilterElementSeq, *arg0: int*) → *rti.connextdds.LocatorFilterElementSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.LocatorFilterElementSeq, arg0: int, arg1: rti.connextdds.LocatorFilterElement) -> None
2. __setitem__(self: rti.connextdds.LocatorFilterElementSeq, arg0: slice, arg1: rti.connextdds.LocatorFilterElementSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.LocatorFilterElementSeq, *x:* rti.connextdds.LocatorFilterElement) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.LocatorFilterElementSeq) → None

Clear the contents

count (*self:* rti.connextdds.LocatorFilterElementSeq, *x:* rti.connextdds.LocatorFilterElement) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.LocatorFilterElementSeq, L: rti.connextdds.LocatorFilterElementSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.LocatorFilterElementSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.LocatorFilterElementSeq, *i: int*, *x:* rti.connextdds.LocatorFilterElement) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.LocatorFilterElementSeq) -> rti.connextdds.LocatorFilterElement

Remove and return the last item

2. pop(self: rti.connextdds.LocatorFilterElementSeq, i: int) -> rti.connextdds.LocatorFilterElement

Remove and return the item at index `i`

remove (*self:* rti.connextdds.LocatorFilterElementSeq, *x:* rti.connextdds.LocatorFilterElement) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**LocatorKind**

Bases: `pybind11_object`

ANY = <LocatorKind.ANY: 0>

INTRA = <LocatorKind.INTRA: 3>

INVALID = <LocatorKind.INVALID: -1>

class LocatorKind

Bases: `pybind11_object`

Members:

INVALID : An invalid locator

ANY : A special value for any kind of locator.

UDPv4 : A locator for a UDPv4 address.

SHMEM : A locator for an address accessed via shared memory.

SHMEM_510 : A locator for an address accessed via shared memory with backwards compatibility for Connext 5.1.0 or earlier.

INTRA : A locator for the Connext INTRA transport.

UDPv6 : A locator for a UDPv6 address.

UDPv6_510 : A locator for a UDPv6 address with backwards compatibility for Connext 5.1.0 or earlier.

TCPV4_LAN : A locator for an address that communicates using TCP on a LAN.

TCPV4_WAN : A locator for an address that communicates using TCP over a WAN.

TLSV4_LAN : A locator for an address that communicates using TLS over a LAN.

TLSV4_WAN : A locator for an address that communicates using TLS over a WAN.

RESERVED : Reserved locator kind.

ANY = <LocatorKind.ANY: 0>

INTRA = <LocatorKind.INTRA: 3>

INVALID = <LocatorKind.INVALID: -1>

RESERVED = <LocatorKind.RESERVED: 1000>

SHMEM = <LocatorKind.SHMEM: 16777216>

SHMEM_510 = <LocatorKind.SHMEM_510: 2>

TCPV4_LAN = <LocatorKind.TCPV4_LAN: 8>

TCPV4_WAN = <LocatorKind.TCPV4_WAN: 9>

TLSV4_LAN = <LocatorKind.TLSV4_LAN: 10>

TLSV4_WAN = <LocatorKind.TLSV4_WAN: 11>

UDPv4 = <LocatorKind.UDPv4: 1>

UDPv6 = <LocatorKind.SHMEM_510: 2>

UDPv6_510 = <LocatorKind.UDPv6_510: 5>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.LocatorKind.LocatorKind) → int

__init__(*self:* rti.connextdds.LocatorKind.LocatorKind, *value: int*) → None

__int__(*self:* rti.connextdds.LocatorKind.LocatorKind) → int

__members__ = {'ANY': <LocatorKind.ANY: 0>, 'INTRA': <LocatorKind.INTRA: 3>, 'INVALID': <LocatorKind.INVALID: -1>, 'RESERVED': <LocatorKind.RESERVED: 1000>, 'SHMEM': <LocatorKind.SHMEM: 16777216>, 'SHMEM_510': <LocatorKind.SHMEM_510: 2>, 'TCPV4_LAN': <LocatorKind.TCPV4_LAN: 8>, 'TCPV4_WAN': <LocatorKind.TCPV4_WAN: 9>, 'TLSV4_LAN': <LocatorKind.TLSV4_LAN: 10>, 'TLSV4_WAN': <LocatorKind.TLSV4_WAN: 11>, 'UDPv4': <LocatorKind.UDPv4: 1>, 'UDPv6': <LocatorKind.SHMEM_510: 2>, 'UDPv6_510': <LocatorKind.UDPv6_510: 5>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.LocatorKind.LocatorKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

RESERVED = <LocatorKind.RESERVED: 1000>

SHMEM = <LocatorKind.SHMEM: 16777216>

SHMEM_510 = <LocatorKind.SHMEM_510: 2>

TCPV4_LAN = <LocatorKind.TCPV4_LAN: 8>

TCPV4_WAN = <LocatorKind.TCPV4_WAN: 9>

TLSV4_LAN = <LocatorKind.TLSV4_LAN: 10>

TLSV4_WAN = <LocatorKind.TLSV4_WAN: 11>

UDPv4 = <LocatorKind.UDPv4: 1>

UDPv6 = <LocatorKind.SHMEM_510: 2>

UDPv6_510 = <LocatorKind.UDPv6_510: 5>

__eq__(*self:* rti.connextdds.LocatorKind, *arg0:* rti.connextdds.LocatorKind) → bool

Apply operator to underlying enumerated values.

__ge__(*self:* rti.connextdds.LocatorKind, *arg0:* rti.connextdds.LocatorKind) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.LocatorKind, *arg0:* rti.connextdds.LocatorKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.LocatorKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.LocatorKind, arg0: rti.connextdds.LocatorKind.LocatorKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.LocatorKind) → *rti.connextdds.LocatorKind.LocatorKind*

Int conversion.

__le__ (*self:* rti.connextdds.LocatorKind, *arg0:* rti.connextdds.LocatorKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.LocatorKind, *arg0:* rti.connextdds.LocatorKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.LocatorKind, *arg0:* rti.connextdds.LocatorKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.LocatorKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**LocatorSeq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.LocatorSeq, *arg0:* rti.connextdds.LocatorSeq) → *rti.connextdds.LocatorSeq*

__bool__ (*self:* rti.connextdds.LocatorSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.LocatorSeq, *x:* rti.connextdds.Locator) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.LocatorSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.LocatorSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.LocatorSeq, *arg0:* rti.connextdds.LocatorSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.LocatorSeq, s: slice) -> rti.connextdds.LocatorSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.LocatorSeq, arg0: int) -> rti.connextdds.Locator

__hash__ = None

__iadd__(*self:* rti.connextdds.LocatorSeq, *arg0:* rti.connextdds.LocatorSeq) → *rti.connextdds.LocatorSeq*

__imul__(*self:* rti.connextdds.LocatorSeq, *arg0: int*) → *rti.connextdds.LocatorSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.LocatorSeq) -> None

2. __init__(self: rti.connextdds.LocatorSeq, arg0: rti.connextdds.LocatorSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.LocatorSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.LocatorSeq) → Iterator

__len__(*self:* rti.connextdds.LocatorSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.LocatorSeq, *arg0: int*) → *rti.connextdds.LocatorSeq*

__ne__(*self:* rti.connextdds.LocatorSeq, *arg0:* rti.connextdds.LocatorSeq) → bool

__rmul__(*self:* rti.connextdds.LocatorSeq, *arg0: int*) → *rti.connextdds.LocatorSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.LocatorSeq, arg0: int, arg1: rti.connextdds.Locator) -> None

2. __setitem__(self: rti.connextdds.LocatorSeq, arg0: slice, arg1: rti.connextdds.LocatorSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.LocatorSeq, *x:* rti.connextdds.Locator) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.LocatorSeq) → None

Clear the contents

count (*self:* rti.connextdds.LocatorSeq, *x:* rti.connextdds.Locator) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.LocatorSeq, L: rti.connextdds.LocatorSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.LocatorSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.LocatorSeq, *i: int*, *x:* rti.connextdds.Locator) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.LocatorSeq) -> rti.connextdds.Locator

Remove and return the last item

2. pop(self: rti.connextdds.LocatorSeq, i: int) -> rti.connextdds.Locator

Remove and return the item at index `i`

remove (*self:* rti.connextdds.LocatorSeq, *x:* rti.connextdds.Locator) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class `rti.connextdds.`**LocatorVector**

Bases: `pybind11_object`

A DDS standard container with functionality similar to a C++ vector.

__eq__ (*self:* rti.connextdds.LocatorVector, *arg0:* rti.connextdds.LocatorVector) → bool

Compare LocatorVectors for equality.

__getitem__ (*self:* rti.connextdds.LocatorVector, *arg0: int*) → *rti.connextdds.Locator*

Get the value at the specified index.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.LocatorVector) -> None

Create an empty LocatorVector

2. __init__(self: rti.connextdds.LocatorVector, size: int) -> None

Create a LocatorVector with a preallocated size.

3. __init__(self: rti.connextdds.LocatorVector, vector: rti.connextdds.LocatorVector) -> None

Create a copy from another LocatorVector.

4. __init__(self: rti.connextdds.LocatorVector, arg0: Iterable) -> None

5. __init__(self: rti.connextdds.LocatorVector, list: rti.connextdds.LocatorSeq) -> None

Create a LocatorVector from a list of values.

__iter__ (*self:* rti.connextdds.LocatorVector) → Iterator

Iterate over the contents of the vector.

__len__ (*self:* rti.connextdds.LocatorVector) → int

Get the length of the LocatorVector.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.LocatorVector, *arg0:* rti.connextdds.LocatorVector) → bool

Compare LocatorVectors for inequality.

__setitem__ (*self:* rti.connextdds.LocatorVector, *arg0: int*, *arg1:* rti.connextdds.Locator) → None

Set the value at the specified index.

clear (*self:* rti.connextdds.LocatorVector) → None

Resize LocatorVector to 0.

resize (*self:* rti.connextdds.LocatorVector, *size: int*) → None

Resize LocatorVector.

class rti.connextdds.**LogCategory**

Bases: pybind11_object

class LogCategory

Bases: pybind11_object

Members:

platform : Log messages pertaining to the underlying platform (hardware and OS) on which RTI Connext is running are in this category.

communication : Log messages pertaining to data serialization and deserialization and network traffic are in this category.

database : Log messages pertaining to the internal database in which RTI Connext objects are stored are in this category.

entities : Log messages pertaining to local and remote entities and to the discovery process are in this category.

api : Log messages pertaining to the API layer of RTI Connext (such as method argument validation) are in this category.

discovery : Log messages pertaining to discovery are in this category.

security : Log messages pertaining to Security Plugins are in this category.

user : Log messages that are generated by the user via the public log APIs are in this category.

all_categories : Log messages pertaining to all category of RTI Connext.

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.LogCategory.LogCategory) → int

__init__(*self:* rti.connextdds.LogCategory.LogCategory, *value: int*) → None

__int__(*self:* rti.connextdds.LogCategory.LogCategory) → int

__members__ = {'all_categories': <LogCategory.all_categories: 8>, 'api': <LogCategory.api: 4>, 'communication': <LogCategory.communication: 1>, 'database': <LogCategory.database: 2>, 'discovery': <LogCategory.discovery: 5>, 'entities': <LogCategory.entities: 3>, 'platform': <LogCategory.platform: 0>, 'security': <LogCategory.security: 6>, 'user': <LogCategory.user: 7>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.LogCategory.LogCategory, *state: int*) → None

__str__()

name(self: handle) -> str

all_categories = <LogCategory.all_categories: 8>

api = <LogCategory.api: 4>

communication = <LogCategory.communication: 1>

database = <LogCategory.database: 2>

discovery = <LogCategory.discovery: 5>

entities = <LogCategory.entities: 3>

property name

platform = <LogCategory.platform: 0>

security = <LogCategory.security: 6>

user = <LogCategory.user: 7>

property value

__eq__ (*self:* rti.connextdds.LogCategory, *arg0:* rti.connextdds.LogCategory) → bool
Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.LogCategory, *arg0:* rti.connextdds.LogCategory) → bool
Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.LogCategory, *arg0:* rti.connextdds.LogCategory) → bool
Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)
Overloaded function.

1. __init__(self: rti.connextdds.LogCategory) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.LogCategory, arg0: rti.connextdds.LogCategory.LogCategory) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.LogCategory) → *rti.connextdds.LogCategory.LogCategory*
Int conversion.

__le__ (*self:* rti.connextdds.LogCategory, *arg0:* rti.connextdds.LogCategory) → bool
Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.LogCategory, *arg0:* rti.connextdds.LogCategory) → bool
Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.LogCategory, *arg0:* rti.connextdds.LogCategory) → bool
Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.LogCategory) → str
String conversion.

all_categories = <LogCategory.all_categories: 8>

api = <LogCategory.api: 4>

communication = <LogCategory.communication: 1>

database = <LogCategory.database: 2>

discovery = <LogCategory.discovery: 5>

entities = <LogCategory.entities: 3>

platform = <LogCategory.platform: 0>

security = <LogCategory.security: 6>

property underlying

Retrieves the actual enumerated value.

user = <LogCategory.user: 7>

class rti.connextdds.**Logger**

Bases: pybind11_object

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

alert(*self:* rti.connextdds.Logger, *msg: str*) → None

Logs an alert message in the user category.

critical(*self:* rti.connextdds.Logger, *msg: str*) → None

Logs a critical message in the user category.

debug(*self:* rti.connextdds.Logger, *msg: str*) → None

Logs a debug message in the user category.

emergency(*self:* rti.connextdds.Logger, *msg: str*) → None

Logs an emergency message in the user category.

error(*self:* rti.connextdds.Logger, *msg: str*) → None

Logs an error message in the user category.

informational(*self:* rti.connextdds.Logger, *msg: str*) → None

Logs an informational message in the user category.

instance = <rti.connextdds.Logger object>

notice(*self:* rti.connextdds.Logger, *msg: str*) → None

Logs a notice message in the user category.

output_file(*self:* rti.connextdds.Logger, *file_name: str*) → None

Set the name of the file to which the logged output is redirected.

output_file_set(*self:* rti.connextdds.Logger, *file_prefix: str*, *file_suffix: str*, *max_bytes: int*, *max_files: int = LENGTH_UNLIMITED*) → None

Set the name of the file to which the logged output is redirected.

output_handler(*self:* rti.connextdds.Logger, *callable_handler: object*) → None

Assigns a callable to which log messages (strings) are directed

property print_format

The message format that RTI Connext will use to log diagnostic information.

reset_output_handler(*self:* rti.connextdds.Logger) → None

Removes the current log handler and sends logging back to the standard output.

property verbosity

The verbosity at which RTI Connext is currently logging diagnostic information.

verbosity_by_category(**args*, ***kwargs*)

Overloaded function.

1. verbosity_by_category(self: rti.connextdds.Logger, category: rti.connextdds.LogCategory) -> rti.connextdds.Verbosity

Get the verbosity at which RTI Connext is currently logging diagnostic information in the given category.

2. verbosity_by_category(self: rti.connextdds.Logger, category: rti.connextdds.LogCategory, verbosity: rti.connextdds.Verbosity) -> None

Set the verbosity at which RTI Connext will log diagnostic information in the given category.

warning(*self:* rti.connextdds.Logger, *msg: str*) → None

Logs a warning message in the user category.

class rti.connextdds.**LongDouble**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.LongDouble, *arg0:* rti.connextdds.LongDouble) → bool

Test for equality

__getitem__(*self:* rti.connextdds.LongDouble, *arg0: int*) → int

Get a byte from the LongDouble buffer.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.LongDouble) -> None

Creates a LongDouble with value 0.

2. __init__(self: rti.connextdds.LongDouble, byte_sequence: rti.connextdds.Uint8Seq) -> None

Creates a LongDouble from a sequence of bytes.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.LongDouble, *arg0:* rti.connextdds.LongDouble) → bool

Test for inequality

__setitem__(*self:* rti.connextdds.LongDouble, *arg0: int*, *arg1: int*) → None

Set a byte in the LongDouble buffer.

__str__ (*self:* rti.connextdds.LongDouble) → str

rti.connextdds.**LongDoubleType**
alias of *Float128Type*

rti.connextdds.**LongLongSeq**
alias of *Int64Seq*

rti.connextdds.**LongLongType**
alias of *Int64Type*

rti.connextdds.**LongSeq**
alias of *Int32Seq*

rti.connextdds.**LongType**
alias of *Int32Type*

class rti.connextdds.**Member**
Bases: pybind11_object

INVALID_ID = 2147483647

__eq__ (*self:* rti.connextdds.Member, *arg0:* rti.connextdds.Member) → bool
Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)
Overloaded function.

1. __init__(self: rti.connextdds.Member, name: str, data_type: rti.connextdds.DynamicType) -> None
2. __init__(self: rti.connextdds.Member, name: str, data_type: rti.connextdds.DynamicType, id: int = -1, is_key: bool = False, is_optional: bool = False, is_pointer: bool = False) -> None

Create a Member with specified properties

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Member, *arg0:* rti.connextdds.Member) → bool
Test for inequality.

property bitset
Checks if member is a bitset

property id
The member ID.

property is_key
Member key field status.

property name

The member name.

property optional

Member optional status

property pointer

Member pointer status

set_name (*self:* rti.connextdds.Member, *arg0: str*) → *rti.connextdds.Member*

Set the member name.

property type

Gets the member type.

class rti.connextdds.**MemberSeq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.MemberSeq, *arg0:* rti.connextdds.MemberSeq) → *rti.connextdds.MemberSeq*

__bool__ (*self:* rti.connextdds.MemberSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.MemberSeq, *x:* rti.connextdds.Member) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.MemberSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.MemberSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.MemberSeq, *arg0:* rti.connextdds.MemberSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.MemberSeq, s: slice) -> rti.connextdds.MemberSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.MemberSeq, arg0: int) -> rti.connextdds.Member

__hash__ = None

__iadd__ (*self:* rti.connextdds.MemberSeq, *arg0:* rti.connextdds.MemberSeq) → *rti.connextdds.MemberSeq*

__imul__ (*self:* rti.connextdds.MemberSeq, *arg0: int*) → *rti.connextdds.MemberSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.MemberSeq) -> None

2. __init__(self: rti.connextdds.MemberSeq, arg0: rti.connextdds.MemberSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.MemberSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.MemberSeq) → Iterator

__len__ (*self:* rti.connextdds.MemberSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.MemberSeq, *arg0: int*) → *rti.connextdds.MemberSeq*

__ne__ (*self:* rti.connextdds.MemberSeq, *arg0:* rti.connextdds.MemberSeq) → bool

__rmul__ (*self:* rti.connextdds.MemberSeq, *arg0: int*) → *rti.connextdds.MemberSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.MemberSeq, arg0: int, arg1: rti.connextdds.Member) -> None

2. __setitem__(self: rti.connextdds.MemberSeq, arg0: slice, arg1: rti.connextdds.MemberSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.MemberSeq, *x:* rti.connextdds.Member) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.MemberSeq) → None

Clear the contents

count (*self:* rti.connextdds.MemberSeq, *x:* rti.connextdds.Member) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.MemberSeq, L: rti.connextdds.MemberSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.MemberSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.MemberSeq, *i: int*, *x:* rti.connextdds.Member) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.MemberSeq) -> rti.connextdds.Member

Remove and return the last item

2. pop(self: rti.connextdds.MemberSeq, i: int) -> rti.connextdds.Member

Remove and return the item at index `i`

remove (*self:* rti.connextdds.MemberSeq, *x:* rti.connextdds.Member) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**Monitoring**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.Monitoring, *arg0:* rti.connextdds.Monitoring) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.Monitoring) → None

Creates the default policy (monitoring is disabled).

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Monitoring, *arg0:* rti.connextdds.Monitoring) → bool

Test for inequality.

property application_name

The name of the resource that represents this Connext application.

disabled = <rti.connextdds.Monitoring object>

property distribution_settings

Configures the distribution of telemetry data.

property enable

Enables the collection and distribution of telemetry data for a Connext application using the Monitoring Library 2.0.

enabled = <rti.connextdds.Monitoring object>

property telemetry_data

Configures the collection of telemetry data.

class rti.connextdds.**MonitoringDedicatedParticipantSettings**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.MonitoringDedicatedParticipantSettings, *arg0:* rti.connextdds.MonitoringDedicatedParticipantSettings) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.MonitoringDedicatedParticipantSettings) → None

Creates an instance with the default settings.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.MonitoringDedicatedParticipantSettings, *arg0:* rti.connextdds.MonitoringDedicatedParticipantSettings) → bool

Test for inequality.

property collector_initial_peers

Determines the initial list of peers that the discovery process will contact to send announcements about the presence of the dedicated_participant.

property domain_id

The domain ID used in the creation of the Monitoring Library 2.0 DomainParticipant.

property enable

Enables the use of a dedicated DomainParticipant to distribute the Connext application telemetry data.

property participant_qos_profile_name

The fully qualified name of the profile used to configure the DomainParticipant that will be used to distribute telemetry data.

class rti.connextdds.**MonitoringDistributionSettings**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.MonitoringDistributionSettings, *arg0:* rti.connextdds.MonitoringDistributionSettings) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.MonitoringDistributionSettings) → None

Creates an instance with the default settings.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.MonitoringDistributionSettings, *arg0:* rti.connextdds.MonitoringDistributionSettings) → bool

Test for inequality.

property dedicated_participant

Configures the use of a dedicated DomainParticipant to distribute the Connext application telemetry data.

property event_settings

Configures the distribution of event metrics.

property logging_settings

Configures the distribution of logging messages.

property periodic_settings

Configures the distribution of periodic metrics.

property publisher_qos_profile_name

The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.

class rti.connextdds.**MonitoringEventDistributionSettings**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.MonitoringEventDistributionSettings, *arg0:* rti.connextdds.MonitoringEventDistributionSettings) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.MonitoringEventDistributionSettings) → None

Creates an instance with the default settings.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.MonitoringEventDistributionSettings, *arg0:* rti.connextdds.MonitoringEventDistributionSettings) → bool

Test for inequality.

property concurrency_level

Defines how concurrent the push of event metrics to the Monitoring Library 2.0 is

property datawriter_qos_profile_name

The fully qualified name of the profile used to configure the DataWriter that distributes event metrics.

property publication_period

Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.

property thread

The settings of the event metric thread.

class rti.connextdds.**MonitoringLoggingDistributionSettings**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.MonitoringLoggingDistributionSettings, *arg0:* rti.connextdds.MonitoringLoggingDistributionSettings) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.MonitoringLoggingDistributionSettings) → None

Creates an instance with the default settings.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.MonitoringLoggingDistributionSettings, *arg0:* rti.connextdds.MonitoringLoggingDistributionSettings) → bool

Test for inequality.

property concurrency_level

Defines how concurrent the push of log messages to the Monitoring Library 2.0 is.

property datawriter_qos_profile_name

The fully qualified name of the profile used to configure the DataWriter that distributes log messages.

property max_historical_logs

The number of log messages that the Monitoring Library 2.0 will keep as history.

property publication_period

Period at which the logging thread publishes log messages.

property thread

The settings of the logging thread.

class rti.connextdds.**MonitoringLoggingForwardingSettings**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.MonitoringLoggingForwardingSettings, *arg0:* rti.connextdds.MonitoringLoggingForwardingSettings) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.MonitoringLoggingForwardingSettings) → None

Creates an instance with the default settings.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.MonitoringLoggingForwardingSettings, *arg0:* rti.connextdds.MonitoringLoggingForwardingSettings) → bool

Test for inequality.

property middleware_forwarding_level

The forwarding log level for LogFacility.MIDDLEWARE

property security_event_forwarding_level

The forwarding log level for LogFacility.SECURITY_EVENT

property service_forwarding_level

The forwarding log level for LogFacility.SERVICE

property user_forwarding_level

The forwarding log level for LogFacility.USER

class rti.connextdds.**MonitoringMetricSelection**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.MonitoringMetricSelection, *arg0:* rti.connextdds.MonitoringMetricSelection) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.MonitoringMetricSelection) → None

Creates an instance with the default settings.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.MonitoringMetricSelection, *arg0:* rti.connextdds.MonitoringMetricSelection) → bool

Test for inequality.

property disabled_metrics_selection

A sequence of POSIX fnmatch patterns that match the names of the metrics that should not be collected and distributed for the observable resources selected by resource_selection.

property enabled_metrics_selection

A sequence of POSIX fnmatch patterns that match the names of the metrics that should be collected and distributed for the observable resources selected by resource_selection

property resource_selection

An expression pattern that selects a subset of resources by matching the pattern to the resource names. enabled_metrics_selection and disabled_metrics_selection are applied to this subset of resources.

class rti.connextdds.**MonitoringMetricSelectionSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.MonitoringMetricSelectionSeq, *arg0:* rti.connextdds.MonitoringMetricSelectionSeq) → *rti.connextdds.MonitoringMetricSelectionSeq*

__bool__(*self:* rti.connextdds.MonitoringMetricSelectionSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.MonitoringMetricSelectionSeq, *x:* rti.connextdds.MonitoringMetricSelection) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.MonitoringMetricSelectionSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.MonitoringMetricSelectionSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.MonitoringMetricSelectionSeq, *arg0:* rti.connextdds.MonitoringMetricSelectionSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.MonitoringMetricSelectionSeq, s: slice) -> rti.connextdds.MonitoringMetricSelectionSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.MonitoringMetricSelectionSeq, arg0: int) -> rti.connextdds.MonitoringMetricSelection

__hash__ = None

__iadd__ (*self:* rti.connextdds.MonitoringMetricSelectionSeq, *arg0:* rti.connextdds.MonitoringMetricSelectionSeq) → *rti.connextdds.MonitoringMetricSelectionSeq*

__imul__ (*self:* rti.connextdds.MonitoringMetricSelectionSeq, *arg0: int*) → *rti.connextdds.MonitoringMetricSelectionSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.MonitoringMetricSelectionSeq) -> None

2. __init__(self: rti.connextdds.MonitoringMetricSelectionSeq, arg0: rti.connextdds.MonitoringMetricSelectionSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.MonitoringMetricSelectionSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.MonitoringMetricSelectionSeq) → Iterator

__len__ (*self:* rti.connextdds.MonitoringMetricSelectionSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.MonitoringMetricSelectionSeq, *arg0: int*) → *rti.connextdds.MonitoringMetricSelectionSeq*

__ne__ (*self:* rti.connextdds.MonitoringMetricSelectionSeq, *arg0:* rti.connextdds.MonitoringMetricSelectionSeq) → bool

__rmul__ (*self:* rti.connextdds.MonitoringMetricSelectionSeq, *arg0: int*) → *rti.connextdds.MonitoringMetricSelectionSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.MonitoringMetricSelectionSeq, arg0: int, arg1: rti.connextdds.MonitoringMetricSelection) -> None

2. __setitem__(self: rti.connextdds.MonitoringMetricSelectionSeq, arg0: slice, arg1: rti.connextdds.MonitoringMetricSelectionSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.MonitoringMetricSelectionSeq, *x:* rti.connextdds.MonitoringMetricSelection) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.MonitoringMetricSelectionSeq) → None

Clear the contents

count (*self:* rti.connextdds.MonitoringMetricSelectionSeq, *x:* rti.connextdds.MonitoringMetricSelection) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.MonitoringMetricSelectionSeq, L: rti.connextdds.MonitoringMetricSelectionSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.MonitoringMetricSelectionSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.MonitoringMetricSelectionSeq, *i: int*, *x:* rti.connextdds.MonitoringMetricSelection) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.MonitoringMetricSelectionSeq) -> rti.connextdds.MonitoringMetricSelection

Remove and return the last item

2. pop(self: rti.connextdds.MonitoringMetricSelectionSeq, i: int) -> rti.connextdds.MonitoringMetricSelection

Remove and return the item at index i

remove (*self:* rti.connextdds.MonitoringMetricSelectionSeq, *x:* rti.connextdds.MonitoringMetricSelection) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**MonitoringPeriodicDistributionSettings**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.MonitoringPeriodicDistributionSettings, *arg0:* rti.connextdds.MonitoringPeriodicDistributionSettings) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.MonitoringPeriodicDistributionSettings) → None

Creates an instance with the default settings.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.MonitoringPeriodicDistributionSettings, *arg0:* rti.connextdds.MonitoringPeriodicDistributionSettings) → bool

Test for inequality.

property datawriter_qos_profile_name

The fully qualified name of the profile used to configure the DataWriter that distributes periodic metrics.

property polling_period

Period at which the periodic metric thread polls and publishes the periodic metrics.

property thread

The settings of the periodic metric thread.

class rti.connextdds.**MonitoringTelemetryData**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.MonitoringTelemetryData, *arg0:* rti.connextdds.MonitoringTelemetryData) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.MonitoringTelemetryData) → None

Creates an instance with the default settings.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.MonitoringTelemetryData, *arg0:* rti.connextdds.MonitoringTelemetryData) → bool

Test for inequality.

property logs

The MonitoringLoggingForwardingSettings containing the SyslogVerbosity levels that will be forwarded for the different LogFacility.

property metrics

The sequence of MonitoringMetricSelection containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.

class rti.connextdds.**MultiChannel**

Bases: pybind11_object

SQL_FILTER_NAME = 'DDSSQL'

STRINGMATCH_FILTER_NAME = 'DDSSTRINGMATCH'

__eq__(*self:* rti.connextdds.MultiChannel, *arg0:* rti.connextdds.MultiChannel) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.MultiChannel) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.MultiChannel, channels: rti.connextdds.ChannelSettingsSeq, filter_name: str = Filter.stringmatch_filter_name) -> None

Creates an instance with the specified channels and filter name.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.MultiChannel, *arg0:* rti.connextdds.MultiChannel) → bool

Test for inequality.

property channels

A sequence of ChannelSettings used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.

This property's getter returns a deep copy.

property filter_name

Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

class rti.connextdds.**MulticastMapping**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.MulticastMapping, *arg0:* rti.connextdds.MulticastMapping) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.MulticastMapping) -> None

Creates the default mapping

2. __init__(self: rti.connextdds.MulticastMapping, addresses: str, topic_expression: str, mapping_function: rti.connextdds.TransportMulticastMappingFunction) -> None

Creates a mapping with the specified parameters

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.MulticastMapping, *arg0:* rti.connextdds.MulticastMapping) → bool

Test for inequality.

property addresses

Multicast mapping addresses.

property mapping_function

The mapping function.

property topic_expression

The topic expression.

class rti.connextdds.**MulticastMappingSeq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.MulticastMappingSeq, *arg0:* rti.connextdds.MulticastMappingSeq) → *rti.connextdds.MulticastMappingSeq*

__bool__ (*self:* rti.connextdds.MulticastMappingSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.MulticastMappingSeq, *x:* rti.connextdds.MulticastMapping) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.MulticastMappingSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.MulticastMappingSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.MulticastMappingSeq, *arg0:* rti.connextdds.MulticastMappingSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.MulticastMappingSeq, s: slice) -> rti.connextdds.MulticastMappingSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.MulticastMappingSeq, arg0: int) -> rti.connextdds.MulticastMapping

__hash__ = None

__iadd__(*self:* rti.connextdds.MulticastMappingSeq, *arg0:* rti.connextdds.MulticastMappingSeq) → *rti.connextdds.MulticastMappingSeq*

__imul__(*self:* rti.connextdds.MulticastMappingSeq, *arg0: int*) → *rti.connextdds.MulticastMappingSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.MulticastMappingSeq) -> None

2. __init__(self: rti.connextdds.MulticastMappingSeq, arg0: rti.connextdds.MulticastMappingSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.MulticastMappingSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.MulticastMappingSeq) → Iterator

__len__(*self:* rti.connextdds.MulticastMappingSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.MulticastMappingSeq, *arg0: int*) → *rti.connextdds.MulticastMappingSeq*

__ne__(*self:* rti.connextdds.MulticastMappingSeq, *arg0:* rti.connextdds.MulticastMappingSeq) → bool

__rmul__(*self:* rti.connextdds.MulticastMappingSeq, *arg0: int*) → *rti.connextdds.MulticastMappingSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.MulticastMappingSeq, arg0: int, arg1: rti.connextdds.MulticastMapping) -> None

2. __setitem__(self: rti.connextdds.MulticastMappingSeq, arg0: slice, arg1: rti.connextdds.MulticastMappingSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.MulticastMappingSeq, *x:* rti.connextdds.MulticastMapping) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.MulticastMappingSeq) → None

Clear the contents

count (*self:* rti.connextdds.MulticastMappingSeq, *x:* rti.connextdds.MulticastMapping) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.MulticastMappingSeq, L: rti.connextdds.MulticastMappingSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.MulticastMappingSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.MulticastMappingSeq, *i: int*, *x:* rti.connextdds.MulticastMapping) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.MulticastMappingSeq) -> rti.connextdds.MulticastMapping

Remove and return the last item

2. pop(self: rti.connextdds.MulticastMappingSeq, i: int) -> rti.connextdds.MulticastMapping

Remove and return the item at index `i`

remove (*self:* rti.connextdds.MulticastMappingSeq, *x:* rti.connextdds.MulticastMapping) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**NoOpAnyDataReaderListener**

Bases: *AnyDataReaderListener*

__init__ (*self:* rti.connextdds.NoOpAnyDataReaderListener) → None

__module__ = 'rti.connextdds'

on_data_available (*self:* rti.connextdds.NoOpAnyDataReaderListener, *arg0:* rti.connextdds.AnyDataReader) → None

Data available callback.

on_liveliness_changed (*self:* rti.connextdds.NoOpAnyDataReaderListener, *arg0:* rti.connextdds.AnyDataReader, *arg1:* rti.connextdds.LivelinessChangedStatus) → None

Liveliness changed callback.

on_requested_deadline_missed(*self:* rti.connextdds.NoOpAnyDataReaderListener, *arg0:* rti.connextdds.AnyDataReader, *arg1:* rti.connextdds.RequestedDeadlineMissedStatus) → None

Requested deadline missed callback.

on_requested_incompatible_qos(*self:* rti.connextdds.NoOpAnyDataReaderListener, *arg0:* rti.connextdds.AnyDataReader, *arg1:* rti.connextdds.RequestedIncompatibleQosStatus) → None

Requested incompatible QoS callback.

on_sample_lost(*self:* rti.connextdds.NoOpAnyDataReaderListener, *arg0:* rti.connextdds.AnyDataReader, *arg1:* rti.connextdds.SampleLostStatus) → None

Sample lost callback.

on_sample_rejected(*self:* rti.connextdds.NoOpAnyDataReaderListener, *arg0:* rti.connextdds.AnyDataReader, *arg1:* rti.connextdds.SampleRejectedStatus) → None

Sample rejected callback.

on_subscription_matched(*self:* rti.connextdds.NoOpAnyDataReaderListener, *arg0:* rti.connextdds.AnyDataReader, *arg1:* rti.connextdds.SubscriptionMatchedStatus) → None

Subscription matched callback.

class rti.connextdds.**NoOpAnyDataWriterListener**

Bases: *AnyDataWriterListener*

__init__(*self:* rti.connextdds.NoOpAnyDataWriterListener) → None

__module__ = 'rti.connextdds'

on_application_acknowledgment(*self:* rti.connextdds.NoOpAnyDataWriterListener, *arg0:* rti.connextdds.AnyDataWriter, *arg1:* rti.connextdds.AcknowledgmentInfo) → None

On application acknowledgment callback

on_instance_replaced(*self:* rti.connextdds.NoOpAnyDataWriterListener, *arg0:* rti.connextdds.AnyDataWriter, *arg1:* rti.connextdds.InstanceHandle) → None

On instance replaced callback.

on_liveliness_lost(*self:* rti.connextdds.NoOpAnyDataWriterListener, *arg0:* rti.connextdds.AnyDataWriter, *arg1:* rti.connextdds.LivelinessLostStatus) → None

Liveliness lost callback.

on_offered_deadline_missed(*self:* rti.connextdds.NoOpAnyDataWriterListener, *arg0:* rti.connextdds.AnyDataWriter, *arg1:* rti.connextdds.OfferedDeadlineMissedStatus) → None

Offered deadline missed callback.

on_offered_incompatible_qos(*self:* rti.connextdds.NoOpAnyDataWriterListener, *arg0:* rti.connextdds.AnyDataWriter, *arg1:* rti.connextdds.OfferedIncompatibleQosStatus) → None

Offered incompatible QoS callback.

on_publication_matched(*self:* rti.connextdds.NoOpAnyDataWriterListener, *arg0:* rti.connextdds.AnyDataWriter, *arg1:* rti.connextdds.PublicationMatchedStatus) → None

Publication matched callback.

on_reliable_reader_activity_changed(*self:* rti.connextdds.NoOpAnyDataWriterListener, *arg0:* rti.connextdds.AnyDataWriter, *arg1:* rti.connextdds.ReliableReaderActivityChangedStatus) → None

Reliable reader activity changed callback.

on_reliable_writer_cache_changed(*self:* rti.connextdds.NoOpAnyDataWriterListener, *arg0:* rti.connextdds.AnyDataWriter, *arg1:* rti.connextdds.ReliableWriterCacheChangedStatus) → None

Reliable writer cache changed callback.

on_service_request_accepted(*self:* rti.connextdds.NoOpAnyDataWriterListener, *arg0:* rti.connextdds.AnyDataWriter, *arg1:* rti.connextdds.ServiceRequestAcceptedStatus) → None

On service request accepted callback.

class rti.connextdds.**NoOpAnyTopicListener**

Bases: *AnyTopicListener*

__init__(*self:* rti.connextdds.NoOpAnyTopicListener) → None

__module__ = 'rti.connextdds'

on_inconsistent_topic(*self:* rti.connextdds.NoOpAnyTopicListener, *arg0:* rti.connextdds.AnyTopic, *arg1:* rti.connextdds.InconsistentTopicStatus) → None

Inconsistent topic callback.

class rti.connextdds.**NoOpDataReaderListener**

Bases: *DataReaderListener*

__init__(*self:* rti.connextdds.NoOpDataReaderListener) → None

__module__ = 'rti.connextdds'

on_data_available(*self:* rti.connextdds.NoOpDataReaderListener, *arg0:* rti.connextdds.DataReader) → None

Data available callback.

on_liveliness_changed(*self:* rti.connextdds.NoOpDataReaderListener, *arg0:* rti.connextdds.DataReader, *arg1:* rti.connextdds.LivelinessChangedStatus) → None

Liveliness changed callback.

on_requested_deadline_missed(*self:* rti.connextdds.NoOpDataReaderListener, *arg0:* rti.connextdds.DataReader, *arg1:* rti.connextdds.RequestedDeadlineMissedStatus) → None

Requested deadline missed callback.

on_requested_incompatible_qos(*self:* rti.connextdds.NoOpDataReaderListener, *arg0:* rti.connextdds.DataReader, *arg1:* rti.connextdds.RequestedIncompatibleQosStatus) → None

Requested incompatible QoS callback.

on_sample_lost(*self:* rti.connextdds.NoOpDataReaderListener, *arg0:* rti.connextdds.DataReader, *arg1:* rti.connextdds.SampleLostStatus) → None

Sample lost callback.

on_sample_rejected(*self:* rti.connextdds.NoOpDataReaderListener, *arg0:* rti.connextdds.DataReader, *arg1:* rti.connextdds.SampleRejectedStatus) → None

Sample rejected callback.

on_subscription_matched(*self:* rti.connextdds.NoOpDataReaderListener, *arg0:* rti.connextdds.DataReader, *arg1:* rti.connextdds.SubscriptionMatchedStatus) → None

Subscription matched callback.

class rti.connextdds.**NoOpDataWriterListener**

Bases: *DataWriterListener*

__init__(*self:* rti.connextdds.NoOpDataWriterListener) → None

__module__ = 'rti.connextdds'

on_application_acknowledgment(*self:* rti.connextdds.NoOpDataWriterListener, *arg0:* rti.connextdds.DataWriter, *arg1:* rti.connextdds.AcknowledgmentInfo) → None

On application acknowledgment callback

on_instance_replaced(*self:* rti.connextdds.NoOpDataWriterListener, *arg0:* rti.connextdds.DataWriter, *arg1:* rti.connextdds.InstanceHandle) → None

On instance replaced callback.

on_liveliness_lost(*self:* rti.connextdds.NoOpDataWriterListener, *arg0:* rti.connextdds.DataWriter, *arg1:* rti.connextdds.LivelinessLostStatus) → None

Liveliness lost callback.

on_offered_deadline_missed(*self:* rti.connextdds.NoOpDataWriterListener, *arg0:* rti.connextdds.DataWriter, *arg1:* rti.connextdds.OfferedDeadlineMissedStatus) → None

Offered deadline missed callback.

on_offered_incompatible_qos(*self:* rti.connextdds.NoOpDataWriterListener, *arg0:* rti.connextdds.DataWriter, *arg1:* rti.connextdds.OfferedIncompatibleQosStatus) → None

Offered incompatible QoS callback.

on_publication_matched(*self:* rti.connextdds.NoOpDataWriterListener, *arg0:* rti.connextdds.DataWriter, *arg1:* rti.connextdds.PublicationMatchedStatus) → None

Publication matched callback.

on_reliable_reader_activity_changed(*self:* rti.connextdds.NoOpDataWriterListener, *arg0:* rti.connextdds.DataWriter, *arg1:* rti.connextdds.ReliableReaderActivityChangedStatus) → None

Reliable reader activity changed callback.

on_reliable_writer_cache_changed(*self:* rti.connextdds.NoOpDataWriterListener, *arg0:* rti.connextdds.DataWriter, *arg1:* rti.connextdds.ReliableWriterCacheChangedStatus) → None

Reliable writer cache changed callback.

on_service_request_accepted(*self:* rti.connextdds.NoOpDataWriterListener, *arg0:* rti.connextdds.DataWriter, *arg1:* rti.connextdds.ServiceRequestAcceptedStatus) → None

On service request accepted callback.

class rti.connextdds.**NoOpDomainParticipantListener**

Bases: *DomainParticipantListener*

__init__(*self:* rti.connextdds.NoOpDomainParticipantListener) → None

__module__ = 'rti.connextdds'

on_invalid_local_identity_status_advance_notice (*self:* rti.connextdds.NoOpDomainParticipantListener, *arg0:* rti.connextdds.DomainParticipant, *arg1:* rti.connextdds.InvalidLocalIdentityAdvanceNoticeStatus) → None

On invalid local identity status advance notice callback

class rti.connextdds.**NoOpPublisherListener**

Bases: *PublisherListener*

__init__ (*self:* rti.connextdds.NoOpPublisherListener) → None

__module__ = 'rti.connextdds'

class rti.connextdds.**NoOpSubscriberListener**

Bases: *SubscriberListener*

__init__ (*self:* rti.connextdds.NoOpSubscriberListener) → None

__module__ = 'rti.connextdds'

on_data_on_readers (*self:* rti.connextdds.NoOpSubscriberListener, *arg0:* rti.connextdds.Subscriber) → None

Data on datareaders callback.

class rti.connextdds.**NoOpTopicListener**

Bases: *TopicListener*

__init__ (*self:* rti.connextdds.NoOpTopicListener) → None

__module__ = 'rti.connextdds'

on_inconsistent_topic (*self:* rti.connextdds.NoOpTopicListener, *arg0:* rti.connextdds.Topic, *arg1:* rti.connextdds.InconsistentTopicStatus) → None

Inconsistent topic callback.

exception rti.connextdds.**NotAllowedBySecurityError**

Bases: *Exception*

__module__ = 'rti.connextdds'

exception rti.connextdds.**NotEnabledError**

Bases: *Exception*

__module__ = 'rti.connextdds'

exception rti.connextdds.**NullReferenceError**

Bases: *Exception*

__module__ = 'rti.connextdds'

rti.connextdds.**OctetSeq**

alias of *Uint8Seq*

rti.connextdds.**OctetType**

alias of *Uint8Type*

class rti.connextdds.**OfferedDeadlineMissedStatus**

Bases: pybind11_object

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property last_instance_handle

Handle to the last instance in the DataWriter for which an offered deadline was missed.

property total_count

The count of a DataWriter's failures to meet write deadlines.

property total_count_change

The delta in total_count since the last time the listener was called or the status was read.

class rti.connextdds.**OfferedIncompatibleQosStatus**

Bases: pybind11_object

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property last_policy

The policy class of one of the policies that was found to be incompatible the last time an incompatibility was detected.

property policies

A list containing for each policy the total number of times that the concerned DataWriter discovered a DataReader for the same Topic and common partition with a requested QoS that is incompatible with that offered by the DataWriter.

property total_count

Total number of times the concerned DataWriter discovered a DataReader for the same Topic, common partition with a requested QoS that is incompatible with that offered by the DataWriter.

property total_count_change

The delta in total_count since the last time the listener was called or the status was read.

exception rti.connextdds.**OutOfResourcesError**

Bases: *Exception*

__module__ = 'rti.connextdds'

class rti.connextdds.**Ownership**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.Ownership, *arg0:* rti.connextdds.Ownership) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Ownership) -> None

Creates an ownership policy set to shared.

2. __init__(self: rti.connextdds.Ownership, kind: rti.connextdds.OwnershipKind) -> None

Creates an instance with the specified ownership kind.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.Ownership, *arg0:* rti.connextdds.Ownership) → bool

Test for inequality.

exclusive = <rti.connextdds.Ownership object>

property kind

The ownership kind.

shared = <rti.connextdds.Ownership object>

class rti.connextdds.**OwnershipKind**

Bases: pybind11_object

EXCLUSIVE = <OwnershipKind.EXCLUSIVE: 1>

class OwnershipKind

Bases: pybind11_object

Members:

SHARED : [default] Indicates shared ownership for each instance.

Multiple writers are allowed to update the same instance and all the updates are made available to the readers. In other words there is no concept of an owner for the instances.

This is the default behavior if the OWNERSHIP policy is not specified or supported.

EXCLUSIVE : Indicates each instance can only be owned by one DataWriter, but the owner of an instance can change dynamically.

The selection of the owner is controlled by the setting of the OWNERSHIP_STRENGTH policy. The owner is always set to be the highest-strength DataWriter object among the ones currently active (as determined by the LIVELINESS).

EXCLUSIVE = <OwnershipKind.EXCLUSIVE: 1>

SHARED = <OwnershipKind.SHARED: 0>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.OwnershipKind.OwnershipKind) → int

__init__(*self:* rti.connextdds.OwnershipKind.OwnershipKind, *value: int*) → None

__int__(*self:* rti.connextdds.OwnershipKind.OwnershipKind) → int

__members__ = {'EXCLUSIVE': <OwnershipKind.EXCLUSIVE: 1>, 'SHARED': <OwnershipKind.SHARED: 0>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.OwnershipKind.OwnershipKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

SHARED = <OwnershipKind.SHARED: 0>

__eq__(*self:* rti.connextdds.OwnershipKind, *arg0:* rti.connextdds.OwnershipKind) → bool

Apply operator to underlying enumerated values.

__ge__(*self:* rti.connextdds.OwnershipKind, *arg0:* rti.connextdds.OwnershipKind) → bool

Apply operator to underlying enumerated values.

__gt__(*self:* rti.connextdds.OwnershipKind, *arg0:* rti.connextdds.OwnershipKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.OwnershipKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.OwnershipKind, arg0: rti.connextdds.OwnershipKind.OwnershipKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.OwnershipKind) → *rti.connextdds.OwnershipKind.OwnershipKind*

Int conversion.

__le__ (*self:* rti.connextdds.OwnershipKind, *arg0:* rti.connextdds.OwnershipKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.OwnershipKind, *arg0:* rti.connextdds.OwnershipKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.OwnershipKind, *arg0:* rti.connextdds.OwnershipKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.OwnershipKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**OwnershipStrength**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.OwnershipStrength, *arg0:* rti.connextdds.OwnershipStrength) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.OwnershipStrength) -> None

Creates an instance with the default strength (0).

2. __init__(self: rti.connextdds.OwnershipStrength, strength: int) -> None

Creates an instance with the specified strength value.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.OwnershipStrength, *arg0:* rti.connextdds.OwnershipStrength) → bool

Test for inequality.

property value

The ownership kind.

class rti.connextdds.**ParticipantBuiltinTopicData**

Bases: pybind11_object

class ContentFilter

Bases: *ContentFilterBase*

__init__(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilter) → None

__module__ = 'rti.connextdds'

compile(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilter, *expression: str*, *parameters:* rti.connextdds.StringSeq, *type_code: Optional[*rti.connextdds.DynamicType*]*, *type_class_name: str*, *old_compile_data: Optional[object]*) → Optional[object]

Compile an instance of the content filter according to the filter expression and parameters of the given data type.

evaluate(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilter, *compile_data: Optional[object]*, *sample:* rti.connextdds.ParticipantBuiltinTopicData, *meta_data:* rti.connextdds.FilterSampleInfo) → bool

Evaluate whether the sample is passing the filter or not according to the sample content.

finalize(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilter, *compile_data: Optional[object]*) → None

A previously compiled instance of the content filter is no longer in use and resources can now be cleaned up.

class ContentFilteredTopic

Bases: *ITopicDescription*, *IAnyTopic*

__eq__(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic, topic: rti.connextdds.ParticipantBuiltinTopicData.Topic, name: str, contentfilter: rti.connextdds.Filter) -> None

Create a ContentFilteredTopic with a name and Filter.

2. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic, topic_description: rti.connextdds.ParticipantBuiltinTopicData.ITopicDescription) -> None

Cast a TopicDescription to a ContentFilteredTopic.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic) → bool

Test for inequality.

append_to_expression_parameter(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic, *index: int*, *extension: str*) → None

Append the extension to the end of parameter at the provided index, separated by a comma.

property filter_expression

Get the filter expression

property filter_parameters

Get/set the filter parameters.

static find(*participant:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic*]

Look up a ContentFilteredTopic by its name in the DomainParticipant.

remove_from_expression_parameter(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic, *index: int*, *remove_term: str*) → None

Removes the specified term from the parameter at the provided index.

set_filter(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic, *arg0:* rti.connextdds.Filter) → None

Set the filter.

property topic

Get the underlying Topic.

class ContentFilteredTopicSeq

Bases: `pybind11_object`

__add__(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq) → *rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq*

__bool__(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, s: slice) -> rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, arg0: int) -> rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic

__hash__ = None

__iadd__(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq) → *rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq*

__imul__(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq) -> None
2. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, arg0: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq) → Iterator

__len__(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq*

__ne__(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq) → bool

__rmul__(*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, arg0: int, arg1: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic) -> None

2. __setitem__(self: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, arg0: slice, arg1: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq) → None

Clear the contents

count (*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, L: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, *i: int*, *x:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq) -> rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic

Remove and return the last item

2. pop(self: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, i: int) -> rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic

Remove and return the item at index `i`

remove (*self:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopicSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class DataReader

Bases: *IDataReader*

class Selector

Bases: `pybind11_object`

__init__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector, *datareader:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → None

Create a Selector for a DataReader to read/take based on selected conditions

__module__ = 'rti.connextdds'

condition(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector, *condition:* rti.connextdds.IReadCondition) → *rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector*

Select samples based on a ReadCondition.

content(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector, *query:* rti.connextdds.Query) → *rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector*

Select samples based on a Query.

instance(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector*

Select a specific instance to read/take.

max_samples(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector, *max: int*) → *rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector*

Limit the number of samples read/taken by the Select.

next_instance(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector, *previous:* rti.connextdds.InstanceHandle) → *rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector*

Select the instance after the specified instance to read/take.

read(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector) → list

Read copies of available samples (data and info) based on the Selector settings.

read_data(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector) → list

Read copies of available valid data based on the Selector settings.

read_loaned(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector) → *rti.connextdds.ParticipantBuiltinTopicData.LoanedSamples*

Take available samples (data and info) based on the Selector settings and return them in a loaned container.

state(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector, *state:* rti.connextdds.DataState) → *rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector*

Select samples with a specified data state.

take(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector) → list

Take copies of available samples (data and info) based on the Selector settings.

take_data (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector) → list

Take copies of available valid data based on the Selector settings.

take_loaned (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader.Selector) → *rti.connextdds.ParticipantBuiltinTopicData.LoanedSamples*

Read available samples (data and info) based on the Selector settings and return them in a loaned container.

__enter__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → *rti.connextdds.ParticipantBuiltinTopicData.DataReader*

Enter a context for this DataReader, to be cleaned up on exiting context

__eq__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → bool

Test for equality.

__exit__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for this DataReader, cleaning up resources.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReader, topic: rti.connextdds.ParticipantBuiltinTopicData.Topic) -> None

Create a DataReader in the implicit subscriber with default QoS.

2. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReader, topic: rti.connextdds.ParticipantBuiltinTopicData.Topic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.ParticipantBuiltinTopicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader in the implicit subscriber with specific QoS and a listener.

3. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReader, cft: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic) -> None

Create a DataReader with a ContentFilteredTopic in the implicit subscriber with default QoS.

4. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReader, cft: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.ParticipantBuiltinTopicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader with a ContentFilteredTopic in the implicit subscriber with specific QoS.

5. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReader, subscriber: rti.connextdds.Subscriber, topic: rti.connextdds.ParticipantBuiltinTopicData.Topic) -> None

Create a DataReader.

6. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReader, subscriber: rti.connextdds.Subscriber, topic: rti.connextdds.ParticipantBuiltinTopicData.Topic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.ParticipantBuiltinTopicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL)

-> None

Create a DataReader in a subscriber with specific QoS and a listener.

7. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReader, subscriber: rti.connextdds.Subscriber, cft: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic) -> None

Create a DataReader with a ContentFilteredTopic in a subscriber with default QoS.

8. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReader, subscriber: rti.connextdds.Subscriber, cft: rti.connextdds.ParticipantBuiltinTopicData.ContentFilteredTopic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.ParticipantBuiltinTopicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader with a ContentFilteredTopic in a subscriber with specific QoS.

9. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReader, reader: rti.connextdds.IAnyDataReader) -> None

Get a typed DataReader from an AnyDataReader.

10. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReader, entity: rti.connextdds.IEntity) -> None

Get a typed DataReader from an Entity.

__lshift__(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *arg0:* rti.connextdds.DataReaderQos) → *rti.connextdds.ParticipantBuiltinTopicData.DataReader*

Set the DataReaderQos for this DataReader.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → bool

Test for inequality.

__rshift__(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *arg0:* rti.connextdds.DataReaderQos) → *rti.connextdds.ParticipantBuiltinTopicData.DataReader*

Get the DataReaderQos from this DataReader

acknowledge_all(**args*, ***kwargs*)

Overloaded function.

1. acknowledge_all(self: rti.connextdds.ParticipantBuiltinTopicData.DataReader) -> None

Acknowledge all previously accessed samples.

2. acknowledge_all(self: rti.connextdds.ParticipantBuiltinTopicData.DataReader, arg0: rti.connextdds.AckResponseData) -> None

Acknowledge all previously accessed samples.

acknowledge_sample(**args*, ***kwargs*)

Overloaded function.

1. acknowledge_sample(self: rti.connextdds.ParticipantBuiltinTopicData.DataReader, sample_info: rti.connextdds.SampleInfo) -> None

Acknowledge a single sample.

2. acknowledge_sample(self: rti.connextdds.ParticipantBuiltinTopicData.DataReader, sample_info: rti.connextdds.SampleInfo, ack_response_data: rti.connextdds.AckResponseData) -> None

Acknowledge a single sample with ack response data.

close (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → None

Close this DataReader.

property datareader_cache_status

Get the DataReaderCacheStatus for the DataReader.

property datareader_protocol_status

Get the DataReaderProtocolStatus for the DataReader.

property default_filter_state

Returns the filter state for the read/take operations.

static find_all_by_topic (*subscriber:* rti.connextdds.Subscriber, *topic_name: str*) → *rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq*

Retrieve all DataReaders for the given topic name in the subscriber.

static find_by_name (**args*, ***kwargs*)

Overloaded function.

1. find_by_name(participant: rti.connextdds.DomainParticipant, name: str) -> Optional[rti.connextdds.ParticipantBuiltinTopicData.DataReader]

Find DataReader in DomainParticipant with the DataReader's name, returning the first found.

2. find_by_name(subscriber: rti.connextdds.Subscriber, name: str) -> Optional[rti.connextdds.ParticipantBuiltinTopicData.DataReader]

Find DataReader in Subscriber with the DataReader's name, returning the first found.

static find_by_topic (*subscriber:* rti.connextdds.Subscriber, *name: str*) → Optional[*rti.connextdds.ParticipantBuiltinTopicData.DataReader*]

Find DataReader in Subscriber with a topic name, returning the first found.

is_matched_publication_alive (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *arg0:* rti.connextdds.InstanceHandle) → bool

Check if a matched publication is alive.

key_value (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ParticipantBuiltinTopicData*

Retrieve the instance key that corresponds to an instance handle.

property listener

Gets or sets the listener with StatusMask.ALL

property liveliness_changed_status

Get the LivelinessChangedStatus for this DataReader.

lookup_instance (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *key_holder:* rti.connextdds.ParticipantBuiltinTopicData) → *rti.connextdds.InstanceHandle*

Retrieve the instance handle that corresponds to an instance key_holder

matched_publication_data(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.PublicationBuiltinTopicData*

Get the PublicationBuiltinTopicData for a publication matched to this DataReader.

matched_publication_datareader_protocol_status(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *publication_handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.DataReaderProtocolStatus*

Get the DataReaderProtocolStatus for the DataReader based on the matched publication handle.

matched_publication_participant_data(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ParticipantBuiltinTopicData*

Get the ParticipantBuiltinTopicData for a publication matched to this DataReader.

property matched_publications

Get a copy of the list of the currently matched publication handles.

property qos

The DataReaderQos for this DataReader.

This property's getter returns a deep copy.

read(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → list

Read copies of all available samples (data and info).

read_data(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → list

Read copies of all available valid data.

read_loaned(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → *rti.connextdds.ParticipantBuiltinTopicData.LoanedSamples*

Read all available samples (data and info) and return them in a loaned container.

property requested_deadline_missed_status

Get the RequestedDeadlineMissed status for the DataReader

property requested_incompatible_qos_status

Get the RequestedIncompatibleQosStatus for the DataReader.

property sample_lost_status

Get the SampleLostStatus for the DataReader.

property sample_rejected_status

Get the SampleRejectedStatus for the DataReader.

select (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → dds::sub::DataReader<dds::topic::TParticipantBuiltinTopicData<rti::topic::ParticipantBuiltinTopicDataImpl>, rti::sub::DataReaderImpl>::Selector

Get a Selector to perform complex data selections, such as per-instance selection, content, and status filtering.

set_listener (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *listener:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and associated event mask.

property subscriber

Returns the parent Subscriber of the DataReader.

property subscription_matched_status

Get the SubscriptionMatchedStatus for the DataReader.

take (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → list

Take copies of all available samples (data and info).

take_data (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → list

Take copies of all available valid data.

take_loaned (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → *rti.connextdds.ParticipantBuiltinTopicData.LoanedSamples*

Take all available samples (data and info) and return them in a loaned container.

property topic_description

Returns the TopicDescription associated with the DataReader.

property topic_name

Get the topic name associated with this DataReader.

property type_name

Get the type name associated with this DataReader.

wait_for_historical_data (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *max_wait:* rti.connextdds.Duration) → None

Waits until all "historical" data is received for DataReaders that have a non-VOLATILE Durability Qos kind.

wait_for_historical_data_async(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *max_wait:* rti.connextdds.Duration) → object

Waits until all "historical" data is received for DataReaders that have a non-VOLATILE Durability Qos kind. This call is awaitable and only for use with asyncio.

class DataReaderListener

Bases: pybind11_object

__init__(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderListener) → None

__module__ = 'rti.connextdds'

on_data_available(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → None

Data available callback.

on_liveliness_changed(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *arg1:* rti.connextdds.LivelinessChangedStatus) → None

Liveliness changed callback.

on_requested_deadline_missed(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *arg1:* rti.connextdds.RequestedDeadlineMissedStatus) → None

Requested deadline missed callback.

on_requested_incompatible_qos(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *arg1:* rti.connextdds.RequestedIncompatibleQosStatus) → None

Requested incompatible QoS callback.

on_sample_lost(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SampleLostStatus) → None

Sample lost callback.

on_sample_rejected(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SampleRejectedStatus) → None

Sample rejected callback.

on_subscription_matched(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SubscriptionMatchedStatus) → None

Subscription matched callback.

class DataReaderSeq

Bases: `pybind11_object`

__add__(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq) → *rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq*

__bool__(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, s: slice) -> rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, arg0: int) -> rti.connextdds.ParticipantBuiltinTopicData.DataReader

__hash__ = None

__iadd__(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq) → *rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq*

__imul__(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq) -> None
2. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, arg0: rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq) → Iterator

__len__(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq*

__ne__(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq) → bool

__rmul__(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, arg0: int, arg1: rti.connextdds.ParticipantBuiltinTopicData.DataReader) -> None
2. __setitem__(self: rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, arg0: slice, arg1: rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq) -> None

Assign list elements using a slice object

append(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → None

Add an item to the end of the list

clear(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq) → None

Clear the contents

count(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → int

Return the number of times x appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, L: rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, *i: int*, *x:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq) -> rti.connextdds.ParticipantBuiltinTopicData.DataReader

Remove and return the last item

2. pop(self: rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, i: int) -> rti.connextdds.ParticipantBuiltinTopicData.DataReader

Remove and return the item at index `i`

remove (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataReaderSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class DataWriter

Bases: *IEntity*, *IAnyDataWriter*

__enter__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter) → *rti.connextdds.ParticipantBuiltinTopicData.DataWriter*

Enter a context for this DataWriter, to be cleaned up on exiting context

__eq__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter) → bool

Test for equality.

__exit__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for this DataWriter, cleaning up resources.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, topic: rti.connextdds.ParticipantBuiltinTopicData.Topic) -> None

Creates a DataWriter in the implicit publisher with default QoS.

2. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, topic: rti.connextdds.ParticipantBuiltinTopicData.Topic, qos: rti.connextdds.DataWriterQos, listener: rti.connextdds.ParticipantBuiltinTopicData.DataWriterListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Creates a DataWriter in the implicit publisher with specific QoS and optionally a listener.

3. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, pub: rti.connextdds.Publisher, topic: rti.connextdds.ParticipantBuiltinTopicData.Topic) -> None

Creates a DataWriter in a publisher with default QoS.

4. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, pub: rti.connextdds.Publisher, topic: rti.connextdds.ParticipantBuiltinTopicData.Topic, qos: rti.connextdds.DataWriterQos, listener: rti.connextdds.ParticipantBuiltinTopicData.DataWriterListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Creates a DataWriter in a publisher with specific QoS and optionally a listener.

5. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, writer: rti.connextdds.IAnyDataWriter) -> None

Create a typed DataWriter from an AnyDataWriter.

6. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, entity: rti.connextdds.IEntity) -> None

Create a typed DataWriter from an Entity.

__lshift__(**args*, ***kwargs*)

Overloaded function.

1. __lshift__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, arg0: rti.connextdds.DataWriterQos) -> rti.connextdds.ParticipantBuiltinTopicData.DataWriter

Sets the DataWriterQos.

2. __lshift__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, arg0: Tuple[rti.connextdds.ParticipantBuiltinTopicData, rti.connextdds.Time]) -> rti.connextdds.ParticipantBuiltinTopicData.DataWriter

Writes a paired sample with a timestamp.

3. __lshift__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, arg0: Tuple[rti.connextdds.ParticipantBuiltinTopicData, rti.connextdds.InstanceHandle]) -> rti.connextdds.ParticipantBuiltinTopicData.DataWriter

Writes a paired sample with an instance handle.

4. __lshift__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, arg0: rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq) -> rti.connextdds.ParticipantBuiltinTopicData.DataWriter

Writes a sequence of pairs of samples with timestamps.

5. __lshift__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, arg0: rti.connextdds.ParticipantBuiltinTopicDataSeq) -> rti.connextdds.ParticipantBuiltinTopicData.DataWriter

Writes a sequence of samples.

6. __lshift__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, arg0: rti.connextdds.ParticipantBuiltinTopicData) -> rti.connextdds.ParticipantBuiltinTopicData.DataWriter

Writes a sample.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter) → bool

Test for inequality.

__rshift__(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg0:* rti.connextdds.DataWriterQos) → *rti.connextdds.ParticipantBuiltinTopicData.DataWriter*

Get the DataWriterQos.

assert_liveliness (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter) → None

Manually asserts the liveliness of the DataWriter.

close (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter) → None

Close this DataWriter.

create_data (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter) → *rti.connextdds.ParticipantBuiltinTopicData*

Create data of the writer's associated type and initialize it.

property datawriter_cache_status

Get a copy of the cache status for this writer.

property datawriter_protocol_status

Get a copy of the protocol status for this writer.

dispose_instance (**args*, ***kwargs*)

Overloaded function.

1. dispose_instance(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.ParticipantBuiltinTopicData.DataWriter

Dispose an instance.

2. dispose_instance(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> rti.connextdds.ParticipantBuiltinTopicData.DataWriter

Dispose an instance with a timestamp.

3. dispose_instance(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, params: rti.connextdds.WriteParams) -> None

Dispose an instance with params.

dispose_instance_async (**args*, ***kwargs*)

Overloaded function.

1. dispose_instance_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> object

Dispose an instance.

2. dispose_instance_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> object

Dispose an instance with a timestamp.

3. dispose_instance_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, key_holder: rti.connextdds.ParticipantBuiltinTopicData) -> object

Dispose the instance associated with key_holder.

4. dispose_instance_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, key_holder: rti.connextdds.ParticipantBuiltinTopicData, timestamp: rti.connextdds.Time) -> object

Dispose the instance associated with key_holder using a timestamp

5. dispose_instance_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, params: rti.connextdds.WriteParams) -> object

Dispose an instance with params.

static find_all_by_topic(*publisher:* rti.connextdds.Publisher, *topic_name: str*) → *rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq*

Retrieve all DataWriters for the given topic name in the publisher.

static find_by_name(**args*, ***kwargs*)

Overloaded function.

1. find_by_name(participant: rti.connextdds.DomainParticipant, name: str) -> Optional[rti.connextdds.ParticipantBuiltinTopicData.DataWriter]

Find DataWriter in DomainParticipant with the provided name, returning the first found.

2. find_by_name(publisher: rti.connextdds.Publisher, name: str) -> Optional[rti.connextdds.ParticipantBuiltinTopicData.DataWriter]

Find DataWriter in Publisher with the DataReader's name, returning the first found.

static find_by_topic(*publisher:* rti.connextdds.Publisher, *name: str*) → Optional[*rti.connextdds.ParticipantBuiltinTopicData.DataWriter*]

Find DataWriter in publisher with a topic name, returning the first found.

flush(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter) → None

Flushes the batch in progress in the context of thecalling thread.

is_matched_subscription_active(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg0:* rti.connextdds.InstanceHandle) → bool

A boolean indicating whether or not the matched subscription is active.

is_sample_app_acknowledged(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *sample_id:* rti.connextdds.SampleIdentity) → bool

Indicates if a sample is considered application-acknowledged.

key_value(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ParticipantBuiltinTopicData*

Retrieve the instance key that corresponds to an instance handle.

property listener

Get the listener associated with the DataWriter or set the listener.

property liveliness_lost_status

Get a copy of the LivelinessLostStatus.

lookup_instance(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *key_holder:* rti.connextdds.ParticipantBuiltinTopicData) → *rti.connextdds.InstanceHandle*

Retrieve the instance handle that corresponds to an instance key_holder

matched_subscription_data(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.SubscriptionBuiltinTopicData*

Get the SubscriptionBuiltinTopicData for a subscription matched to this DataWriter.

matched_subscription_datawriter_protocol_status(**args*, ***kwargs*)

Overloaded function.

1. matched_subscription_datawriter_protocol_status(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.DataWriterProtocolStatus

Get a copy of the protocol status for this writer per a matched subscription handle.

2. matched_subscription_datawriter_protocol_status(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, locator: rti.connextdds.Locator) -> rti.connextdds.DataWriterProtocolStatus

Get a copy of the protocol status for this writer per a matched subscription locator.

matched_subscription_participant_data(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ParticipantBuiltinTopicData*

Get the ParticipantBuiltinTopicData for a subscription matched to this DataWriter.

property matched_subscriptions

Get a copy of the list of the currently matched subscription handles.

property matched_subscriptions_locators

The locators used to communicate with matched DataReaders.

property offered_deadline_missed_status

Get a copy of the OfferedDeadlineMissedStatus.

property offered_incompatible_qos_status

Get a copy of the OfferedIncompatibleQosStatus

property publication_matched_status

Get a copy of the PublicationMatchedStatus

property publisher

Get the Publisher that owns this DataWriter.

property qos

The DataWriterQos for this DataWriter.This property's getter returns a deep copy.

register_instance(**args*, ***kwargs*)

Overloaded function.

1. register_instance(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, key_holder: rti.connextdds.ParticipantBuiltinTopicData) -> rti.connextdds.InstanceHandle

Informs RTI Connext that the application will be modifying a particular instance.

2. register_instance(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, key_holder: rti.connextdds.ParticipantBuiltinTopicData, timestamp: rti.connextdds.Time) -> rti.connextdds.InstanceHandle

Informs RTI Connext that the application will be modifying a particular instance and specified the timestamp.

3. register_instance(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, key_holder: rti.connextdds.ParticipantBuiltinTopicData, params: rti.connextdds.WriteParams) -> rti.connextdds.InstanceHandle

Registers instance with parameters.

property reliable_reader_activity_changed_status

Get a copy of the reliable reader activity changed status for this writer.

property reliable_writer_cache_changed_status

Get a copy of the reliable cache status for this writer.

property service_request_accepted_status

Get a copy of the service request accepted status for this writer.

set_listener(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *listener:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and event mask for the DataWriter.

property topic

Get the Topic object associated with this DataWriter.

property topic_name

Get the topic name associated with this DataWriter.

property type_name

Get the type name for the topic object associated with this DataWriter.

unregister_instance(**args*, ***kwargs*)

Overloaded function.

1. unregister_instance(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.ParticipantBuiltinTopicData.DataWriter

Unregister an instance.

2. unregister_instance(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> rti.connextdds.ParticipantBuiltinTopicData.DataWriter

Unregister an instance with timestamp.

3. unregister_instance(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, params: rti.connextdds.WriteParams) -> None

Unregister an instance with parameters.

unregister_instance_async(**args*, ***kwargs*)

Overloaded function.

1. unregister_instance_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> object

Unregister an instance.

2. unregister_instance_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> object

Unregister an instance with timestamp.

3. unregister_instance_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, key_holder: rti.connextdds.ParticipantBuiltinTopicData) -> object

Unregister the instance associated with key_holder.

4. unregister_instance_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, key_holder: rti.connextdds.ParticipantBuiltinTopicData, timestamp: rti.connextdds.Time) -> object

Unregister the instance associate with key_holder using a timestamp.

5. unregister_instance_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, params: rti.connextdds.WriteParams) -> object

Unregister an instance with parameters.

wait_for_acknowledgments(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *max_wait:* rti.connextdds.Duration) → None

Blocks the calling thread until all data written by a reliable DataWriter is acknowledged or until the timeout expires.

wait_for_asynchronous_publishing(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *max_wait:* rti.connextdds.Duration) → None

This operation blocks the calling thread (up to max_wait) until all data written by the asynchronous DataWriter is sent and acknowledged (if reliable) by all matched DataReader entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; a time out indicates that max_wait elapsed before all the data was sent and/or acknowledged.

In other words, this guarantees that sending to best effort DataReader is complete in addition to what DataWriter.wait_for_acknowledgments() provides.

If the DataWriter does not have PublishMode kind set to PublishModeKind.ASYNCHRONOUS the operation will complete immediately

wait_for_asynchronous_publishing_async(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *max_wait:* rti.connextdds.Duration) → object

This function is awaitable until either a timeout of max_wait or all data written by the asynchronous DataWriter is sent and acknowledged (if reliable) by all matched DataReader entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; a time out indicates that max_wait elapsed before all the data was sent and/or acknowledged.This function works with asyncio.

In other words, this guarantees that sending to best effort DataReader is complete in addition to what DataWriter.wait_for_acknowledgments() provides.

If the DataWriter does not have PublishMode kind set to PublishModeKind.ASYNCHRONOUS the operation will complete immediately

write(**args*, ***kwargs*)

Overloaded function.

1. write(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, samples: rti.connextdds.ParticipantBuiltinTopicDataSeq) -> None

Write a sequence of samples.

2. write(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, samples: rti.connextdds.ParticipantBuiltinTopicDataSeq, timestamp: rti.connextdds.Time) -> None

Write a sequence of samples with a timestamp.

3. write(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, sample: rti.connextdds.ParticipantBuiltinTopicData) -> None

Write a sample.

4. write(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, sample: rti.connextdds.ParticipantBuiltinTopicData, timestamp: rti.connextdds.Time) -> None

Write a sample with a specified timestamp.

5. write(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, sample: rti.connextdds.ParticipantBuiltinTopicData, handle: rti.connextdds.InstanceHandle) -> None

Write a sample with an instance handle.

6. write(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, sample: rti.connextdds.ParticipantBuiltinTopicData, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> None

Write a sample with an instance handle and specified timestamp.

7. write(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, sample: rti.connextdds.ParticipantBuiltinTopicData, params: rti.connextdds.WriteParams) -> None

Write with advanced parameters.

write_async(**args*, ***kwargs*)

Overloaded function.

1. write_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, sample: rti.connextdds.ParticipantBuiltinTopicData) -> object

Write a sample. This method is awaitable and is only for use with asyncio.

2. write_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, sample: rti.connextdds.ParticipantBuiltinTopicData, timestamp: rti.connextdds.Time) -> object

Write a sample with a specified timestamp. This methods is awaitable and only for use with asyncio.

3. write_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, sample: rti.connextdds.ParticipantBuiltinTopicData, handle: rti.connextdds.InstanceHandle) -> object

Write a sample with an instance handle. This method is awaitable and only for use with asyncio.

4. write_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, sample: rti.connextdds.ParticipantBuiltinTopicData, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> object

Write a sample with an instance handle and specified timestamp. This method is awaitable and only for use with asyncio.

5. write_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, samples: rti.connextdds.ParticipantBuiltinTopicDataSeq) -> object

Write a sequence of samples. This method is awaitable and only for use with asyncio.

6. write_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, samples: rti.connextdds.ParticipantBuiltinTopicDataSeq, timestamp: rti.connextdds.Time) -> object

Write a sequence of samples with a timestamp. This method is awaitable and only for use with asyncio.

7. write_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, samples:

rti.connextdds.ParticipantBuiltinTopicDataSeq, handles: rti.connextdds.InstanceHandleSeq) -> object

Write a sequence of samples with their instance handles. This method is awaitable and only for use with asyncio.

8. write_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, samples: rti.connextdds.ParticipantBuiltinTopicDataSeq, handles: rti.connextdds.InstanceHandleSeq, timestamp: rti.connextdds.Time) -> object

Write a sequence of samples with their instance handles and a timestamp. This method is awaitable and only for use with asyncio.

9. write_async(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriter, sample: rti.connextdds.ParticipantBuiltinTopicData, params: rti.connextdds.WriteParams) -> object

Write with advanced parameters.

class DataWriterListener

Bases: pybind11_object

__init__(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterListener) → None

__module__ = 'rti.connextdds'

on_application_acknowledgment(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.AcknowledgmentInfo) → None

On application acknowledgment callback

on_instance_replaced(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.InstanceHandle) → None

On instance replaced callback.

on_liveliness_lost(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.LivelinessLostStatus) → None

Liveliness lost callback.

on_offered_deadline_missed(*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.OfferedDeadlineMissedStatus) → None

Offered deadline missed callback.

on_offered_incompatible_qos (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.OfferedIncompatibleQosStatus) → None

Offered incompatible QoS callback.

on_publication_matched (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.PublicationMatchedStatus) → None

Publication matched callback.

on_reliable_reader_activity_changed (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ReliableReaderActivityChangedStatus) → None

Reliable reader activity changed callback.

on_reliable_writer_cache_changed (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ReliableWriterCacheChangedStatus) → None

Reliable writer cache changed callback.

on_service_request_accepted (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ServiceRequestAcceptedStatus) → None

On service request accepted callback.

class DataWriterSeq

Bases: `pybind11_object`

__add__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq) → *rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq*

__bool__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, s: slice) -> rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, arg0: int) -> rti.connextdds.ParticipantBuiltinTopicData.DataWriter

__hash__ = None

__iadd__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq) → *rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq*

__imul__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq) -> None
2. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, arg0: rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq) → Iterator

__len__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq*

__ne__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq) → bool

__rmul__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, arg0: int, arg1: rti.connextdds.ParticipantBuiltinTopicData.DataWriter) -> None
2. __setitem__(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, arg0: slice, arg1: rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq) → None

Clear the contents

count (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, L: rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, *i: int*, *x:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq) -> rti.connextdds.ParticipantBuiltinTopicData.DataWriter

Remove and return the last item

2. pop(self: rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, i: int) -> rti.connextdds.ParticipantBuiltinTopicData.DataWriter

Remove and return the item at index `i`

remove (*self:* rti.connextdds.ParticipantBuiltinTopicData.DataWriterSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class ITopicDescription

Bases: *IEntity*

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property name

The name of the entity conforming to the ITopicDescription interface.

property participant

The parent DomainParticipant.

property type_name

The name of the associated type.

class LoanedSample

Bases: pybind11_object

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.LoanedSample) -> None

Basic constructor

2. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.LoanedSample, data: rti.connextdds.ParticipantBuiltinTopicData, info: rti.connextdds.SampleInfo) -> None

Construct LoanedSample with data and info.

__iter__(*self:* rti.connextdds.ParticipantBuiltinTopicData.LoanedSample) → object

__module__ = 'rti.connextdds'

property data

Get the data associated with the sample.

property info

Get the info associated with the sample.

class LoanedSamples

Bases: pybind11_object

__enter__(*self:* rti.connextdds.ParticipantBuiltinTopicData.LoanedSamples) → *rti.connextdds.ParticipantBuiltinTopicData.LoanedSamples*

Enter a context for the loaned samples, loan returned on context exit.

__exit__(*self:* rti.connextdds.ParticipantBuiltinTopicData.LoanedSamples, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for the loaned samples, returning the resources.

__getitem__(*self:* rti.connextdds.ParticipantBuiltinTopicData.LoanedSamples, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicData.LoanedSample*

Access a LoanedSample object in an array-like syntax

__init__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.LoanedSamples) → None

Create an empty LoanedSamples object.

__iter__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.LoanedSamples) → Iterator

__len__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.LoanedSamples) → int

Get the number of samples in the loan.

__module__ = 'rti.connextdds'

property length

Get the number of samples in the loan.

return_loan (*self:* rti.connextdds.ParticipantBuiltinTopicData.LoanedSamples) → None

Returns the loan to the DataReader.

class NoOpDataReaderListener

Bases: *DataReaderListener*

__init__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpDataReaderListener) → None

__module__ = 'rti.connextdds'

on_data_available (*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReader) → None

Data available callback.

on_liveliness_changed (*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *arg1:* rti.connextdds.LivelinessChangedStatus) → None

Liveliness changed callback.

on_requested_deadline_missed (*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *arg1:* rti.connextdds.RequestedDeadlineMissedStatus) → None

Requested deadline missed callback.

on_requested_incompatible_qos (*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *arg1:* rti.connextdds.RequestedIncompatibleQosStatus) → None

Requested incompatible QoS callback.

on_sample_lost(*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SampleLostStatus) → None

Sample lost callback.

on_sample_rejected(*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SampleRejectedStatus) → None

Sample rejected callback.

on_subscription_matched(*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SubscriptionMatchedStatus) → None

Subscription matched callback.

class NoOpDataWriterListener

Bases: *DataWriterListener*

__init__(*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpDataWriterListener) → None

__module__ = 'rti.connextdds'

on_application_acknowledgment(*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.AcknowledgmentInfo) → None

On application acknowledgment callback

on_instance_replaced(*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.InstanceHandle) → None

On instance replaced callback.

on_liveliness_lost(*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.LivelinessLostStatus) → None

Liveliness lost callback.

on_offered_deadline_missed(*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.OfferedDeadlineMissedStatus) → None

Offered deadline missed callback.

on_offered_incompatible_qos(*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.OfferedIncompatibleQosStatus) → None

Offered incompatible QoS callback.

on_publication_matched(*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.PublicationMatchedStatus) → None

Publication matched callback.

on_reliable_reader_activity_changed(*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ReliableReaderActivityChangedStatus) → None

Reliable reader activity changed callback.

on_reliable_writer_cache_changed(*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ReliableWriterCacheChangedStatus) → None

Reliable writer cache changed callback.

on_service_request_accepted(*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ServiceRequestAcceptedStatus) → None

On service request accepted callback.

class NoOpTopicListener

Bases: *TopicListener*

__init__(*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpTopicListener) → None

__module__ = 'rti.connextdds'

on_inconsistent_topic(*self:* rti.connextdds.ParticipantBuiltinTopicData.NoOpTopicListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.Topic, *arg1:* rti.connextdds.InconsistentTopicStatus) → None

Inconsistent topic callback.

class Sample

Bases: pybind11_object

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.Sample, data: rti.connextdds.ParticipantBuiltinTopicData, info: rti.connextdds.SampleInfo) -> None

Construct Sample with data and info.

2. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.Sample, sample: rti.connextdds.ParticipantBuiltinTopicData.Sample) -> None

Copy constructor.

3. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.Sample) -> None

Basic constructor

4. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.Sample, loaned_sample: rti.connextdds.ParticipantBuiltinTopicData.LoanedSample) -> None

Construct a sample with a loaned sample.

__iter__(*self:* rti.connextdds.ParticipantBuiltinTopicData.Sample) → object

__module__ = 'rti.connextdds'

property data

The data associated with the sample.

property info

Get the info associated with the sample.

class SharedSamples

Bases: pybind11_object

__getitem__(*self:* rti.connextdds.ParticipantBuiltinTopicData.SharedSamples, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicData.LoanedSample*

Get the sample at the specified index.

__init__(*self:* rti.connextdds.ParticipantBuiltinTopicData.SharedSamples, *loaned_samples:* rti.connextdds.ParticipantBuiltinTopicData.LoanedSamples) → None

Constructs an instance of SharedSamples, removing ownership of the loan from the Loaned Samples.

__iter__(*self:* rti.connextdds.ParticipantBuiltinTopicData.SharedSamples) → Iterator

Make a sample iterator

__len__(*self:* rti.connextdds.ParticipantBuiltinTopicData.SharedSamples) → int

Returns the number of samples.

__module__ = 'rti.connextdds'

class Topic

Bases: *ITopicDescription*, *IAnyTopic*

__eq__(*self:* rti.connextdds.ParticipantBuiltinTopicData.Topic, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.Topic) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.Topic, entity: rti.connextdds.IEntity) -> None

Cast an Entity to a Topic.

2. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.Topic, topic_description: rti.connextdds.ParticipantBuiltinTopicData.ITopicDescription) -> None

Cast an ITopicDescription to a Topic.

3. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.Topic, topic: rti.connextdds.IAnyTopic) -> None

Create a typed Topic from an AnyTopic.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.ParticipantBuiltinTopicData.Topic, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.Topic) → bool

Test for inequality.

static find(*participant:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.ParticipantBuiltinTopicData.Topic*]

Look up a Topic by its name in the DomainParticipant.

property inconsistent_topic_status

Get a copy of the current InconsistentTopicStatus for this Topic.

property listener

The listener.

property qos

Get the TopicQos for this Topic.

This property's getter returns a deep copy.

set_listener(*self:* rti.connextdds.ParticipantBuiltinTopicData.Topic, *listener:* rti.connextdds.ParticipantBuiltinTopicData.TopicListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and event mask.

class TopicDescription

Bases: *ITopicDescription*

__eq__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicDescription, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.TopicDescription) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.TopicDescription, topic: rti.connextdds.ParticipantBuiltinTopicData.ITopicDescription) -> None

Cast a Topic to a TopicDescription.

2. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.TopicDescription, entity: rti.connextdds.IEntity) -> None

Cast a Topic to a TopicDescription.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicDescription, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.TopicDescription) → bool

Test for inequality.

class TopicListener

Bases: pybind11_object

__init__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicListener) → None

__module__ = 'rti.connextdds'

on_inconsistent_topic (*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicListener, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.Topic, *arg1:* rti.connextdds.InconsistentTopicStatus) → None

Inconsistent topic callback.

class TopicSeq

Bases: pybind11_object

__add__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq) → *rti.connextdds.ParticipantBuiltinTopicData.TopicSeq*

__bool__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData.Topic) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, s: slice) -> rti.connextdds.ParticipantBuiltinTopicData.TopicSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, arg0: int) -> rti.connextdds.ParticipantBuiltinTopicData.Topic

__hash__ = None

__iadd__(*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq) → *rti.connextdds.ParticipantBuiltinTopicData.TopicSeq*

__imul__(*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicData.TopicSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.TopicSeq) -> None
2. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, arg0: rti.connextdds.ParticipantBuiltinTopicData.TopicSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq) → Iterator

__len__(*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicData.TopicSeq*

__ne__(*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq) → bool

__rmul__(*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicData.TopicSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, arg0: int, arg1: rti.connextdds.ParticipantBuiltinTopicData.Topic) -> None
2. __setitem__(self: rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, arg0: slice, arg1: rti.connextdds.ParticipantBuiltinTopicData.TopicSeq) -> None

Assign list elements using a slice object

append(*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData.Topic) → None

Add an item to the end of the list

clear(*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq) → None

Clear the contents

count(*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData.Topic) → int

Return the number of times `x` appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, L: rti.connextdds.ParticipantBuiltinTopicData.TopicSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert(*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, *i: int*, *x:* rti.connextdds.ParticipantBuiltinTopicData.Topic) → None

Insert an item at a given position.

pop(**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.ParticipantBuiltinTopicData.TopicSeq) -> rti.connextdds.ParticipantBuiltinTopicData.Topic

Remove and return the last item

2. pop(self: rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, i: int) -> rti.connextdds.ParticipantBuiltinTopicData.Topic

Remove and return the item at index `i`

remove(*self:* rti.connextdds.ParticipantBuiltinTopicData.TopicSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData.Topic) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class ValidLoanedSamples

Bases: `pybind11_object`

__enter__(*self:* rti.connextdds.ParticipantBuiltinTopicData.ValidLoanedSamples) → *rti.connextdds.ParticipantBuiltinTopicData.ValidLoanedSamples*

__exit__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.ValidLoanedSamples, *arg0: object*, *arg1: object*, *arg2: object*) → None

__init__ (**args*, ***kwargs*)

__iter__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.ValidLoanedSamples) → Iterator

__module__ = 'rti.connextdds'

class WriterContentFilter

Bases: *ContentFilter*

__init__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.WriterContentFilter) → None

__module__ = 'rti.connextdds'

writer_attach (*self:* rti.connextdds.ParticipantBuiltinTopicData.WriterContentFilter) → Optional[object]

A writer-side filtering API to create some state that can facilitate filtering on the writer side.

writer_compile (*self:* rti.connextdds.ParticipantBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *property:* rti.connextdds.ExpressionProperty, *expression: str*, *parameters:* rti.connextdds.StringSeq, *type_code: Optional[*rti.connextdds.DynamicType*]*, *type_class_name: str*, *cookie:* rti.connextdds.Cookie) → None

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching DataReader.

writer_detach (*self:* rti.connextdds.ParticipantBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*) → None

A writer-side filtering API to clean up a previously created state using writer_attach.

writer_evaluate (*self:* rti.connextdds.ParticipantBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *sample:* rti.connextdds.ParticipantBuiltinTopicData, *meta_data:* rti.connextdds.FilterSampleInfo) → *rti.connextdds.CookieVector*

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching DataReader.

writer_finalize (*self:* rti.connextdds.ParticipantBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *cookie:* rti.connextdds.Cookie) → None

A writer-side filtering API to clean up a previously compiled instance of the content filter.

writer_return_loan (*self:* rti.connextdds.ParticipantBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *cookies:* rti.connextdds.CookieVector) → None

A writer-side filtering API to return the loan on the list of DataReaders returned by writer_evaluate.

class WriterContentFilterHelper

Bases: *WriterContentFilter*

__init__ (*self:* rti.connextdds.ParticipantBuiltinTopicData.WriterContentFilterHelper) → None

__module__ = 'rti.connextdds'

add_cookie (*self:* rti.connextdds.ParticipantBuiltinTopicData.WriterContentFilterHelper, *cookie:* rti.connextdds.Cookie) → None

A helper function which will add a Cookie to the Cookie sequence that is then returned by the writer_evaluate function.

writer_evaluate_helper (*self:* rti.connextdds.ParticipantBuiltinTopicData.WriterContentFilterHelper, *writer_filter_data: Optional[object]*, *sample:* rti.connextdds.ParticipantBuiltinTopicData, *meta_data:* rti.connextdds.FilterSampleInfo) → None

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching DataReader.

__eq__ (*self:* rti.connextdds.ParticipantBuiltinTopicData, *arg0:* rti.connextdds.ParticipantBuiltinTopicData) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.ParticipantBuiltinTopicData) → None

Create a default ParticipantBuiltinTopicData.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ParticipantBuiltinTopicData, *arg0:* rti.connextdds.ParticipantBuiltinTopicData) → bool

Test for inequality.

property dds_builtin_endpoints

Bitmap of builtin endpoints supported by the participant.

property default_unicast_locators

Get the unicast locators used when individual entities do not specify unicast locators.

property domain_id

Get the domain ID associated with the discovered DomainParticipant.

property key

Get the DCPS key to distinguish entries.

property partial_configuration

Indicates whether the ParticipantBuiltinTopicData only contains bootstrapping information.

property participant_name

Get the participant name and role name.

property product_version

Get the current version for RTI Connext.

property property

Get the Property policy of the corresponding DomainParticipant.

property reachability_lease_duration

The reachability lease duration.

property rtps_protocol_version

Get the version number of the RTPS wire protocol used.

property rtps_vendor_id

Get the ID of the vendor implementing the RTPS wire protocol.

topic_name = 'DCPSParticipant'

property transport_info

Get the sequence of TransportInfo containing information about each of the installed transports of the discovered DomainParticipant.

property user_data

Get the UserData policy of the corresponding DomainParticipant.

property vendor_builtin_endpoints

Bitmap of vendor-specific builtin endpoints supported by the participant.

class rti.connextdds.**ParticipantBuiltinTopicDataSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.ParticipantBuiltinTopicDataSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicDataSeq) → *rti.connextdds.ParticipantBuiltinTopicDataSeq*

__bool__(*self:* rti.connextdds.ParticipantBuiltinTopicDataSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.ParticipantBuiltinTopicDataSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.ParticipantBuiltinTopicDataSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.ParticipantBuiltinTopicDataSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.ParticipantBuiltinTopicDataSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicDataSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.ParticipantBuiltinTopicDataSeq, s: slice) -> rti.connextdds.ParticipantBuiltinTopicDataSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.ParticipantBuiltinTopicDataSeq, arg0: int) -> rti.connextdds.ParticipantBuiltinTopicData

__hash__ = None

__iadd__ (*self:* rti.connextdds.ParticipantBuiltinTopicDataSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicDataSeq) → *rti.connextdds.ParticipantBuiltinTopicDataSeq*

__imul__ (*self:* rti.connextdds.ParticipantBuiltinTopicDataSeq, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicDataSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ParticipantBuiltinTopicDataSeq) -> None

2. __init__(self: rti.connextdds.ParticipantBuiltinTopicDataSeq, arg0: rti.connextdds.ParticipantBuiltinTopicDataSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.ParticipantBuiltinTopicDataSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.ParticipantBuiltinTopicDataSeq) → Iterator

__len__ (*self:* rti.connextdds.ParticipantBuiltinTopicDataSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.ParticipantBuiltinTopicDataSeq, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicDataSeq*

__ne__ (*self:* rti.connextdds.ParticipantBuiltinTopicDataSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicDataSeq) → bool

__rmul__ (*self:* rti.connextdds.ParticipantBuiltinTopicDataSeq, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicDataSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.ParticipantBuiltinTopicDataSeq, arg0: int, arg1: rti.connextdds.ParticipantBuiltinTopicData) -> None

2. __setitem__(self: rti.connextdds.ParticipantBuiltinTopicDataSeq, arg0: slice, arg1: rti.connextdds.ParticipantBuiltinTopicDataSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.ParticipantBuiltinTopicDataSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.ParticipantBuiltinTopicDataSeq) → None

Clear the contents

count (*self:* rti.connextdds.ParticipantBuiltinTopicDataSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.ParticipantBuiltinTopicDataSeq, L: rti.connextdds.ParticipantBuiltinTopicDataSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.ParticipantBuiltinTopicDataSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.ParticipantBuiltinTopicDataSeq, *i: int*, *x:* rti.connextdds.ParticipantBuiltinTopicData) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.ParticipantBuiltinTopicDataSeq) -> rti.connextdds.ParticipantBuiltinTopicData

Remove and return the last item

2. pop(self: rti.connextdds.ParticipantBuiltinTopicDataSeq, i: int) -> rti.connextdds.ParticipantBuiltinTopicData

Remove and return the item at index `i`

remove (*self:* rti.connextdds.ParticipantBuiltinTopicDataSeq, *x:* rti.connextdds.ParticipantBuiltinTopicData) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**ParticipantBuiltinTopicDataTimestampedSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq) → *rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq*

__bool__(*self:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, *x:* *Tuple[*rti.connextdds.ParticipantBuiltinTopicData, rti.connextdds.Time*]*) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, s: slice) -> rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, arg0: int) -> Tuple[rti.connextdds.ParticipantBuiltinTopicData, rti.connextdds.Time]

__hash__ = None

__iadd__(*self:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq) → *rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq*

__imul__(*self:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq) -> None

2. __init__(self: rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, arg0: rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq) → Iterator

__len__ (*self:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq*

__ne__ (*self:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, *arg0:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__rmul__ (*self:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, *arg0: int*) → *rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, arg0: int, arg1: Tuple[rti.connextdds.ParticipantBuiltinTopicData, rti.connextdds.Time]) -> None

2. __setitem__(self: rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, arg0: slice, arg1: rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, *x: Tuple[*rti.connextdds.ParticipantBuiltinTopicData*,* rti.connextdds.Time*]*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq) → None

Clear the contents

count (*self:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, *x: Tuple[*rti.connextdds.ParticipantBuiltinTopicData*,* rti.connextdds.Time*]*) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, L: rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, *i: int*, *x:* *Tuple[*rti.connextdds.ParticipantBuiltinTopicData, rti.connextdds.Time*]*) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq) -> Tuple[rti.connextdds.ParticipantBuiltinTopicData, rti.connextdds.Time]

Remove and return the last item

2. pop(self: rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, i: int) -> Tuple[rti.connextdds.ParticipantBuiltinTopicData, rti.connextdds.Time]

Remove and return the item at index `i`

remove (*self:* rti.connextdds.ParticipantBuiltinTopicDataTimestampedSeq, *x:* *Tuple[*rti.connextdds.ParticipantBuiltinTopicData, rti.connextdds.Time*]*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class `rti.connextdds.`**Partition**

Bases: `pybind11_object`

__eq__ (*self:* rti.connextdds.Partition, *arg0:* rti.connextdds.Partition) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Partition) -> None

Creates a policy with the default partition.

2. __init__(self: rti.connextdds.Partition, partition: str) -> None

Creates a policy with a single partition with the specified name.

3. __init__(self: rti.connextdds.Partition, partitions: rti.connextdds.StringSeq) -> None

Creates a policy with the partitions specified in a sequence.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Partition, *arg0:* rti.connextdds.Partition) → bool

Test for inequality.

property name

The partition names specified in a sequence.

class rti.connextdds.**PersistentJournalKind**

Bases: pybind11_object

Members:

DELETE : Deletes the rollback journal at the conclusion of each transaction

TRUNCATE : Commits transactions by truncating the rollback journal to zero-length instead of deleting it

PERSIST : Prevents the rollback journal from being deleted at the end of each transaction. Instead, the header of the journal is overwritten with zeros

MEMORY : Stores the rollback journal in volatile RAM. This saves disk I/O

WAL : Uses a write-ahead log instead of a rollback journal to implement transactions

OFF : Completely disables the rollback journal

DELETE = <PersistentJournalKind.DELETE: 0>

MEMORY = <PersistentJournalKind.MEMORY: 3>

OFF = <PersistentJournalKind.OFF: 5>

PERSIST = <PersistentJournalKind.PERSIST: 2>

TRUNCATE = <PersistentJournalKind.TRUNCATE: 1>

WAL = <PersistentJournalKind.WAL: 4>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.PersistentJournalKind) → int

__init__(*self:* rti.connextdds.PersistentJournalKind, *value: int*) → None

__int__(*self:* rti.connextdds.PersistentJournalKind) → int

__members__ = {'DELETE': <PersistentJournalKind.DELETE: 0>, 'MEMORY': <PersistentJournalKind.MEMORY: 3>, 'OFF': <PersistentJournalKind.OFF: 5>, 'PERSIST': <PersistentJournalKind.PERSIST: 2>, 'TRUNCATE': <PersistentJournalKind.TRUNCATE: 1>, 'WAL': <PersistentJournalKind.WAL: 4>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.PersistentJournalKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

class rti.connextdds.**PersistentStorageSettings**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.PersistentStorageSettings, *arg0:* rti.connextdds.PersistentStorageSettings) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.PersistentStorageSettings) → None

Create a PersistentStorageSettings with default settings.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.PersistentStorageSettings, *arg0:* rti.connextdds.PersistentStorageSettings) → bool

Test for inequality.

property enable

Enables durable writer history in a DataWriter and durable reader state in a DataReader.

property file_name

The file name where the durable writer history or durable reader state will be stored.

property journal_kind

Sets the journal mode of the persistent storage.

property reader_checkpoint_frequency

Controls how often the reader state is stored into the database.

property restore

Indicates if the persisted writer history or reader state must be restored.

property synchronization_kind

Sets the journal mode of the persistent storage.

property trace_file_name

The file name where to store the SQL statements executed when loading and storing the durable writer history or durable reader state.

property vacuum

Sets the auto-vacuum status of the storage.

property writer_instance_cache_allocation

Configures the resource limits associated with the instance durable writer history cache.

property writer_memory_state

Determines how much state will be kept in memory by the durable writer history in order to avoid accessing the persistent storage in disk.

property writer_sample_cache_allocation

Configures the resource limits associated with the sample durable writer history cache.

class rti.connextdds.**PersistentSynchronizationKind**

Bases: pybind11_object

Members:

NORMAL : Data (e.g., new sample) is written to disk at critical moments

FULL : Data (e.g., new sample) is written to physical disk immediately

OFF : No synchronization is enforced

FULL = <PersistentSynchronizationKind.FULL: 1>

NORMAL = <PersistentSynchronizationKind.NORMAL: 0>

OFF = <PersistentSynchronizationKind.OFF: 2>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.PersistentSynchronizationKind) → int

__init__(*self:* rti.connextdds.PersistentSynchronizationKind, *value: int*) → None

__int__(*self:* rti.connextdds.PersistentSynchronizationKind) → int

__members__ = {'FULL': <PersistentSynchronizationKind.FULL: 1>, 'NORMAL': <PersistentSynchronizationKind.NORMAL: 0>, 'OFF': <PersistentSynchronizationKind.OFF: 2>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.PersistentSynchronizationKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

exception rti.connextdds.**PreconditionNotMetError**

Bases: *Exception*

__module__ = 'rti.connextdds'

class rti.connextdds.**Presentation**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.Presentation, *arg0:* rti.connextdds.Presentation) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Presentation) -> None

Create an instance with the default value.

2. __init__(self: rti.connextdds.Presentation, access_scope: rti.connextdds.PresentationAccessScopeKind, coherent_access: bool, ordered_access: bool) -> None

Creates an instance with the specified access scope and coherent and ordered access.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.Presentation, *arg0:* rti.connextdds.Presentation) → bool

Test for inequality.

property access_scope

Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.

property coherent_access

Controls whether coherent access is supported within the access_scope.

property drop_incomplete_coherent_set

Indicates whether or not a DataReader should drop samples from an incomplete coherent set

static group_access_scope(*coherent: bool*, *ordered: bool*) → *rti.connextdds.Presentation*

Creates a Presentation instance with group access scope.

static instance_access_scope(*coherent: bool*, *ordered: bool*) → *rti.connextdds.Presentation*

Creates a Presentation instance with instance access scope.

property ordered_access

Controls whether ordered access is supported within the access_scope.

static topic_access_scope(*coherent: bool*, *ordered: bool*) → *rti.connextdds.Presentation*

Creates a Presentation instance with topic access scope.

class rti.connextdds.**PresentationAccessScopeKind**

Bases: pybind11_object

GROUP = <PresentationAccessScopeKind.GROUP: 2>

HIGHEST_OFFERED = <PresentationAccessScopeKind.HIGHEST_OFFERED: 3>

INSTANCE = <PresentationAccessScopeKind.INSTANCE: 0>

class PresentationAccessScopeKind

Bases: pybind11_object

Members:

INSTANCE : [default] Scope spans only a single instance.

Indicates that changes to one instance need not be coherent nor ordered with respect to changes to any other instance. In other words, order and coherent changes apply to each instance separately.

TOPIC : Scope spans to all instances within the same DataWriter (or DataReader), but not across instances in different DataWriter (or DataReader).

GROUP : Scope spans to all instances belonging to DataWriter (or DataReader) entities within the same Publisher (or Subscriber).

HIGHEST_OFFERED : This value only applies to a Subscriber. The Subscriber will use the access scope specified by each remote Publisher.

GROUP = <PresentationAccessScopeKind.GROUP: 2>

HIGHEST_OFFERED = <PresentationAccessScopeKind.HIGHEST_OFFERED: 3>

INSTANCE = <PresentationAccessScopeKind.INSTANCE: 0>

TOPIC = <PresentationAccessScopeKind.TOPIC: 1>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.PresentationAccessScopeKind.PresentationAccessScopeKind) → int

__init__ (*self:* rti.connextdds.PresentationAccessScopeKind.PresentationAccessScopeKind, *value: int*) → None

__int__ (*self:* rti.connextdds.PresentationAccessScopeKind.PresentationAccessScopeKind) → int

__members__ = {'GROUP': <PresentationAccessScopeKind.GROUP: 2>, 'HIGHEST_OFFERED': <PresentationAccessScopeKind.HIGHEST_OFFERED: 3>, 'INSTANCE': <PresentationAccessScopeKind.INSTANCE: 0>, 'TOPIC': <PresentationAccessScopeKind.TOPIC: 1>}

__module__ = 'rti.connextdds'

__ne__ (*self: object*, *other: object*) → bool

__repr__ (*self: object*) → str

__setstate__ (*self:* rti.connextdds.PresentationAccessScopeKind.PresentationAccessScopeKind, *state: int*) → None

__str__ ()

name(self: handle) -> str

property name

property value

TOPIC = <PresentationAccessScopeKind.TOPIC: 1>

__eq__ (*self:* rti.connextdds.PresentationAccessScopeKind, *arg0:* rti.connextdds.PresentationAccessScopeKind) → bool

Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.PresentationAccessScopeKind, *arg0:* rti.connextdds.PresentationAccessScopeKind) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.PresentationAccessScopeKind, *arg0:* rti.connextdds.PresentationAccessScopeKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PresentationAccessScopeKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.PresentationAccessScopeKind, arg0: rti.connextdds.PresentationAccessScopeKind.PresentationAccessScopeKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.PresentationAccessScopeKind) → *rti.connextdds.PresentationAccessScopeKind.PresentationAccessScopeKind*

Int conversion.

__le__ (*self:* rti.connextdds.PresentationAccessScopeKind, *arg0:* rti.connextdds.PresentationAccessScopeKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.PresentationAccessScopeKind, *arg0:* rti.connextdds.PresentationAccessScopeKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.PresentationAccessScopeKind, *arg0:* rti.connextdds.PresentationAccessScopeKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.PresentationAccessScopeKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**PrintFormat**

Bases: pybind11_object

DEBUG = <PrintFormat.DEBUG: 13935>

DEFAULT = <PrintFormat.DEFAULT: 13330>

MAXIMAL = <PrintFormat.MAXIMAL: 16255>

MINIMAL = <PrintFormat.MINIMAL: 17>

class PrintFormat

Bases: pybind11_object

Members:

DEFAULT : Print message, method name, and activity context (default).

TIMESTAMPED : Print message, method name, activity context, and timestamp.

VERBOSE : Print message with all available context information (includes thread identifier, activity context).

VERBOSE_TIMESTAMPED : Print message with all available context information, and timestamp.

DEBUG : Print a set of field that may be useful for internal debug.

MINIMAL : Print only message number and method name.

MAXIMAL : Print all available fields.

DEBUG = <PrintFormat.DEBUG: 13935>

DEFAULT = <PrintFormat.DEFAULT: 13330>

MAXIMAL = <PrintFormat.MAXIMAL: 16255>

MINIMAL = <PrintFormat.MINIMAL: 17>

TIMESTAMPED = <PrintFormat.TIMESTAMPED: 13586>

VERBOSE = <PrintFormat.VERBOSE: 15894>

VERBOSE_TIMESTAMPED = <PrintFormat.VERBOSE_TIMESTAMPED: 16150>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.PrintFormat.PrintFormat) → int

__init__(*self:* rti.connextdds.PrintFormat.PrintFormat, *value: int*) → None

__int__(*self:* rti.connextdds.PrintFormat.PrintFormat) → int

__members__ = {'DEBUG': <PrintFormat.DEBUG: 13935>, 'DEFAULT': <PrintFormat.DEFAULT: 13330>, 'MAXIMAL': <PrintFormat.MAXIMAL: 16255>, 'MINIMAL': <PrintFormat.MINIMAL: 17>, 'TIMESTAMPED': <PrintFormat.TIMESTAMPED: 13586>, 'VERBOSE': <PrintFormat.VERBOSE: 15894>, 'VERBOSE_TIMESTAMPED': <PrintFormat.VERBOSE_TIMESTAMPED: 16150>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.PrintFormat.PrintFormat, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

TIMESTAMPED = <PrintFormat.TIMESTAMPED: 13586>

VERBOSE = <PrintFormat.VERBOSE: 15894>

VERBOSE_TIMESTAMPED = <PrintFormat.VERBOSE_TIMESTAMPED: 16150>

__eq__ (*self:* rti.connextdds.PrintFormat, *arg0:* rti.connextdds.PrintFormat) → bool

Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.PrintFormat, *arg0:* rti.connextdds.PrintFormat) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.PrintFormat, *arg0:* rti.connextdds.PrintFormat) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PrintFormat) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.PrintFormat, arg0: rti.connextdds.PrintFormat.PrintFormat) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.PrintFormat) → *rti.connextdds.PrintFormat.PrintFormat*

Int conversion.

__le__ (*self:* rti.connextdds.PrintFormat, *arg0:* rti.connextdds.PrintFormat) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.PrintFormat, *arg0:* rti.connextdds.PrintFormat) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.PrintFormat, *arg0:* rti.connextdds.PrintFormat) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.PrintFormat) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**PrintFormatKind**

Bases: pybind11_object

DEFAULT = <PrintFormatKind.DEFAULT: 0>

JSON = <PrintFormatKind.JSON: 2>

class PrintFormatKind

Bases: pybind11_object

Members:

DEFAULT : Use a default format specific to RTI Connext to represent the data when converting a data sample to a string.

XML : Use an XML format to represent the data when converting a data sample to a string.

JSON : Use an JSON format to represent the data when converting a data sample to a string.

DEFAULT = <PrintFormatKind.DEFAULT: 0>

JSON = <PrintFormatKind.JSON: 2>

XML = <PrintFormatKind.XML: 1>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.PrintFormatKind.PrintFormatKind) → int

__init__(*self:* rti.connextdds.PrintFormatKind.PrintFormatKind, *value: int*) → None

__int__(*self:* rti.connextdds.PrintFormatKind.PrintFormatKind) → int

__members__ = {'DEFAULT': <PrintFormatKind.DEFAULT: 0>, 'JSON': <PrintFormatKind.JSON: 2>, 'XML': <PrintFormatKind.XML: 1>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.PrintFormatKind.PrintFormatKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

XML = <PrintFormatKind.XML: 1>

__eq__(*self:* rti.connextdds.PrintFormatKind, *arg0:* rti.connextdds.PrintFormatKind) → bool

Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.PrintFormatKind, *arg0:* rti.connextdds.PrintFormatKind) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.PrintFormatKind, *arg0:* rti.connextdds.PrintFormatKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PrintFormatKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.PrintFormatKind, arg0: rti.connextdds.PrintFormatKind.PrintFormatKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.PrintFormatKind) → *rti.connextdds.PrintFormatKind.PrintFormatKind*

Int conversion.

__le__ (*self:* rti.connextdds.PrintFormatKind, *arg0:* rti.connextdds.PrintFormatKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.PrintFormatKind, *arg0:* rti.connextdds.PrintFormatKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.PrintFormatKind, *arg0:* rti.connextdds.PrintFormatKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.PrintFormatKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**PrintFormatProperty**

Bases: pybind11_object

__init__ (*self:* rti.connextdds.PrintFormatProperty, *kind:* rti.connextdds.PrintFormatKind = *PrintFormatKind.DEFAULT*, *is_pretty_print: bool = True*, *is_enum_as_int: bool = False*, *is_include_root_elements: bool = True*) → None

Initalize PrintFormatProperty object.

__module__ = 'rti.connextdds'

default = <rti.connextdds.PrintFormatProperty object>

property enum_as_int

Get/set the value of enum_as_int.

property include_root_elements

Get/set the value of include_root_elements.

json = <rti.connextdds.PrintFormatProperty object>

property kind

Get/set the kind value.

property pretty_print

Get/set the value of pretty_print.

xml = <rti.connextdds.PrintFormatProperty object>

class rti.connextdds.**ProductVersion**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.ProductVersion, *arg0:* rti.connextdds.ProductVersion) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.ProductVersion) → None

Creates the unknown product version.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.ProductVersion, *arg0:* rti.connextdds.ProductVersion) → bool

Test for inequality.

__str__(*self:* rti.connextdds.ProductVersion) → str

current = <rti.connextdds.ProductVersion object>

property major_version

The major product version.

property minor_version

The minor product version.

property native_build_id

A string with the build ID of the native Connext libraries.

property release_version

The release letter of the product version.

property revision_version

The revision number of product.

unknown = <rti.connextdds.ProductVersion object>

class rti.connextdds.**Property**

Bases: pybind11_object

__contains__ (*self:* rti.connextdds.Property, *arg0: str*) → bool

__eq__ (*self:* rti.connextdds.Property, *arg0:* rti.connextdds.Property) → bool

Test for equality.

__getitem__ (*self:* rti.connextdds.Property, *arg0: str*) → str

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Property) -> None

Creates an empty Property container.

2. __init__(self: rti.connextdds.Property, entries: dict, propagate: bool = False) -> None

Creates a Property container with entries from a dictionary of entries

3. __init__(self: rti.connextdds.Property, entries: rti.connextdds.StringPairSeq, propagate: bool = False) -> None

Creates a Property container with entries from a sequence of entries

__len__ (*self:* rti.connextdds.Property) → int

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Property, *arg0:* rti.connextdds.Property) → bool

Test for inequality.

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.Property, arg0: str, arg1: str) -> rti.connextdds.Property

2. __setitem__(self: rti.connextdds.Property, arg0: str, arg1: Tuple[str, bool]) -> rti.connextdds.Property

exists (*self:* rti.connextdds.Property, *key: str*) → bool

Returns true if a property exists.

get (*self:* rti.connextdds.Property, *key: str*) → str

Returns the value of a property identified by a key if it exists.

get_all (*self:* rti.connextdds.Property) → *rti.connextdds.StringMap*

Retrieves a copy of all the entries in a map.

propagate (*self:* rti.connextdds.Property, *key: str*) → bool

Returns whether the 'propagate' attribute for a property is set or not.

remove (*self:* rti.connextdds.Property, *key: str*) → bool

Removes the property identified by a key.

set (**args*, ***kwargs*)

Overloaded function.

1. set(self: rti.connextdds.Property, entry: Tuple[str, str], propagate: bool = False) -> rti.connextdds.Property

Adds or assigns a property from a pair of strings.

2. set(self: rti.connextdds.Property, key: str, value: str, propagate: bool = False) -> rti.connextdds.Property

Adds or assigns a property from a key and a value.

3. set(self: rti.connextdds.Property, entries: rti.connextdds.StringPairSeq, propagate: bool = False) -> rti.connextdds.Property

Adds a range of properties.

4. set(self: rti.connextdds.Property, entries: dict, propagate: bool = False) -> None

Adds a range of properties.

size (*self:* rti.connextdds.Property) → int

Returns the number of properties.

try_get (*self:* rti.connextdds.Property, *key: str*) → Optional[str]

Returns the value of a property identified by a key if it exists.

class rti.connextdds.**ProtocolVersion**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.ProtocolVersion, *arg0:* rti.connextdds.ProtocolVersion) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ProtocolVersion) -> None

Creates an invalid protocol version.

2. __init__(self: rti.connextdds.ProtocolVersion, major: int, minor: int) -> None

Creates a version with the given major and minor values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ProtocolVersion, *arg0:* rti.connextdds.ProtocolVersion) → bool

Test for inequality.

current = <rti.connextdds.ProtocolVersion object>

property major_version

The major version number.

property minor_version

The minor version number.

class rti.connextdds.**PublicationBuiltinTopicData**

Bases: pybind11_object

class ContentFilter

Bases: *ContentFilterBase*

__init__(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilter) → None

__module__ = 'rti.connextdds'

compile(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilter, *expression: str*, *parameters:* rti.connextdds.StringSeq, *type_code: Optional[*rti.connextdds.DynamicType*]*, *type_class_name: str*, *old_compile_data: Optional[object]*) → Optional[object]

Compile an instance of the content filter according to the filter expression and parameters of the given data type.

evaluate(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilter, *compile_data: Optional[object]*, *sample:* rti.connextdds.PublicationBuiltinTopicData, *meta_data:* rti.connextdds.FilterSampleInfo) → bool

Evaluate whether the sample is passing the filter or not according to the sample content.

finalize(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilter, *compile_data: Optional[object]*) → None

A previously compiled instance of the content filter is no longer in use and resources can now be cleaned up.

class ContentFilteredTopic

Bases: *ITopicDescription*, *IAnyTopic*

__eq__(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic, *arg0:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic, topic: rti.connextdds.PublicationBuiltinTopicData.Topic, name: str, contentfilter: rti.connextdds.Filter) -> None

Create a ContentFilteredTopic with a name and Filter.

2. __init__(self: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic, topic_description: rti.connextdds.PublicationBuiltinTopicData.ITopicDescription) -> None

Cast a TopicDescription to a ContentFilteredTopic.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic, *arg0:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic) → bool

Test for inequality.

append_to_expression_parameter(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic, *index: int*, *extension: str*) → None

Append the extension to the end of parameter at the provided index, separated by a comma.

property filter_expression

Get the filter expression

property filter_parameters

Get/set the filter parameters.

static find(*participant:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic*]

Look up a ContentFilteredTopic by its name in the DomainParticipant.

remove_from_expression_parameter(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic, *index: int*, *remove_term: str*) → None

Removes the specified term from the parameter at the provided index.

set_filter(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic, *arg0:* rti.connextdds.Filter) → None

Set the filter.

property topic

Get the underlying Topic.

class ContentFilteredTopicSeq

Bases: `pybind11_object`

__add__(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq) → *rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq*

__bool__(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, *x:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, s: slice) -> rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, arg0: int) -> rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic

__hash__ = None

__iadd__(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq) → *rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq*

__imul__(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq) -> None
2. __init__(self: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, arg0: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq) → Iterator

__len__(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq*

__ne__ (*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq) → bool

__rmul__ (*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, arg0: int, arg1: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic) -> None
2. __setitem__(self: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, arg0: slice, arg1: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, *x:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq) → None

Clear the contents

count (*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, *x:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, L: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, *i: int*, *x:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq) -> rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic

Remove and return the last item

2. pop(self: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, i: int) -> rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic

Remove and return the item at index `i`

remove(*self:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopicSeq, *x:* rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class DataReader

Bases: *IDataReader*

class Selector

Bases: pybind11_object

__init__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector, *datareader:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → None

Create a Selector for a DataReader to read/take based on selected conditions

__module__ = 'rti.connextdds'

condition(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector, *condition:* rti.connextdds.IReadCondition) → *rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector*

Select samples based on a ReadCondition.

content(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector, *query:* rti.connextdds.Query) → *rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector*

Select samples based on a Query.

instance(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector*

Select a specific instance to read/take.

max_samples(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector, *max: int*) → *rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector*

Limit the number of samples read/taken by the Select.

next_instance(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector, *previous:* rti.connextdds.InstanceHandle) → *rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector*

Select the instance after the specified instance to read/take.

read(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector) → list

Read copies of available samples (data and info) based on the Selector settings.

read_data(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector) → list

Read copies of available valid data based on the Selector settings.

read_loaned(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector) → *rti.connextdds.PublicationBuiltinTopicData.LoanedSamples*

Take available samples (data and info) based on the Selector settings and return them in a loaned container.

state (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector, *state:* rti.connextdds.DataState) → *rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector*

Select samples with a specified data state.

take (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector) → list

Take copies of available samples (data and info) based on the Selector settings.

take_data (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector) → list

Take copies of available valid data based on the Selector settings.

take_loaned (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader.Selector) → *rti.connextdds.PublicationBuiltinTopicData.LoanedSamples*

Read available samples (data and info) based on the Selector settings and return them in a loaned container.

__enter__ (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → *rti.connextdds.PublicationBuiltinTopicData.DataReader*

Enter a context for this DataReader, to be cleaned up on exiting context

__eq__ (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → bool

Test for equality.

__exit__ (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for this DataReader, cleaning up resources.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataReader, topic: rti.connextdds.PublicationBuiltinTopicData.Topic) -> None

Create a DataReader in the implicit subscriber with default QoS.

2. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataReader, topic: rti.connextdds.PublicationBuiltinTopicData.Topic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.PublicationBuiltinTopicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader in the implicit subscriber with specific QoS and a listener.

3. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataReader, cft: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic) -> None

Create a DataReader with a ContentFilteredTopic in the implicit subscriber with default QoS.

4. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataReader, cft: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.PublicationBuiltinTopicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader with a ContentFilteredTopic in the implicit subscriber with specific QoS.

5. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataReader, subscriber: rti.connextdds.Subscriber, topic: rti.connextdds.PublicationBuiltinTopicData.Topic) -> None

Create a DataReader.

6. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataReader, subscriber: rti.connextdds.Subscriber, topic: rti.connextdds.PublicationBuiltinTopicData.Topic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.PublicationBuiltinTopicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader in a subscriber with specific QoS and a listener.

7. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataReader, subscriber: rti.connextdds.Subscriber, cft: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic) -> None

Create a DataReader with a ContentFilteredTopic in a subscriber with default QoS.

8. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataReader, subscriber: rti.connextdds.Subscriber, cft: rti.connextdds.PublicationBuiltinTopicData.ContentFilteredTopic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.PublicationBuiltinTopicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader with a ContentFilteredTopic in a subscriber with specific QoS.

9. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataReader, reader: rti.connextdds.IAnyDataReader) -> None

Get a typed DataReader from an AnyDataReader.

10. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataReader, entity: rti.connextdds.IEntity) -> None

Get a typed DataReader from an Entity.

__lshift__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *arg0:* rti.connextdds.DataReaderQos) → *rti.connextdds.PublicationBuiltinTopicData.DataReader*

Set the DataReaderQos for this DataReader.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → bool

Test for inequality.

__rshift__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *arg0:* rti.connextdds.DataReaderQos) → *rti.connextdds.PublicationBuiltinTopicData.DataReader*

Get the DataReaderQos from this DataReader

acknowledge_all(**args*, ***kwargs*)

Overloaded function.

1. acknowledge_all(self: rti.connextdds.PublicationBuiltinTopicData.DataReader) -> None

Acknowledge all previously accessed samples.

2. acknowledge_all(self: rti.connextdds.PublicationBuiltinTopicData.DataReader, arg0: rti.connextdds.AckResponseData) -> None

Acknowledge all previously accessed samples.

acknowledge_sample(**args*, ***kwargs*)

Overloaded function.

1. acknowledge_sample(self: rti.connextdds.PublicationBuiltinTopicData.DataReader, sample_info: rti.connextdds.SampleInfo) -> None

Acknowledge a single sample.

2. acknowledge_sample(self: rti.connextdds.PublicationBuiltinTopicData.DataReader, sample_info: rti.connextdds.SampleInfo, ack_response_data: rti.connextdds.AckResponseData) -> None

Acknowledge a single sample with ack response data.

close(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → None

Close this DataReader.

property datareader_cache_status

Get the DataReaderCacheStatus for the DataReader.

property datareader_protocol_status

Get the DataReaderProtocolStatus for the DataReader.

property default_filter_state

Returns the filter state for the read/take operations.

static find_all_by_topic(*subscriber:* rti.connextdds.Subscriber, *topic_name: str*) → *rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq*

Retrieve all DataReaders for the given topic name in the subscriber.

static find_by_name(**args*, ***kwargs*)

Overloaded function.

1. find_by_name(participant: rti.connextdds.DomainParticipant, name: str) -> Optional[rti.connextdds.PublicationBuiltinTopicData.DataReader]

Find DataReader in DomainParticipant with the DataReader's name, returning the first found.

2. find_by_name(subscriber: rti.connextdds.Subscriber, name: str) -> Optional[rti.connextdds.PublicationBuiltinTopicData.DataReader]

Find DataReader in Subscriber with the DataReader's name, returning the first found.

static find_by_topic(*subscriber:* rti.connextdds.Subscriber, *name: str*) → Optional[*rti.connextdds.PublicationBuiltinTopicData.DataReader*]

Find DataReader in Subscriber with a topic name, returning the first found.

is_matched_publication_alive(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *arg0:* rti.connextdds.InstanceHandle) → bool

Check if a matched publication is alive.

key_value (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.PublicationBuiltinTopicData*

Retrieve the instance key that corresponds to an instance handle.

property listener

Gets or sets the listener with StatusMask.ALL

property liveliness_changed_status

Get the LivelinessChangedStatus for this DataReader.

lookup_instance (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *key_holder:* rti.connextdds.PublicationBuiltinTopicData) → *rti.connextdds.InstanceHandle*

Retrieve the instance handle that corresponds to an instance key_holder

matched_publication_data (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.PublicationBuiltinTopicData*

Get the PublicationBuiltinTopicData for a publication matched to this DataReader.

matched_publication_datareader_protocol_status (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *publication_handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.DataReaderProtocolStatus*

Get the DataReaderProtocolStatus for the DataReader based on the matched publication handle.

matched_publication_participant_data (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ParticipantBuiltinTopicData*

Get the ParticipantBuiltinTopicData for a publication matched to this DataReader.

property matched_publications

Get a copy of the list of the currently matched publication handles.

property qos

The DataReaderQos for this DataReader.

This property's getter returns a deep copy.

read (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → list

Read copies of all available samples (data and info).

read_data (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → list

Read copies of all available valid data.

read_loaned (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → *rti.connextdds.PublicationBuiltinTopicData.LoanedSamples*

Read all available samples (data and info) and return them in a loaned container.

property requested_deadline_missed_status

Get the RequestedDeadlineMissed status for the DataReader

property requested_incompatible_qos_status

Get the RequestedIncompatibleQosStatus for the DataReader.

property sample_lost_status

Get the SampleLostStatus for the DataReader.

property sample_rejected_status

Get the SampleRejectedStatus for the DataReader.

select (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → dds::sub::DataReader<dds::topic::TPublicationBuiltinTopicData<rti::topic::PublicationBuiltinTopicDataImpl>, rti::sub::DataReaderImpl>::Selector

Get a Selector to perform complex data selections, such as per-instance selection, content, and status filtering.

set_listener (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *listener:* rti.connextdds.PublicationBuiltinTopicData.DataReaderListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and associated event mask.

property subscriber

Returns the parent Subscriber of the DataReader.

property subscription_matched_status

Get the SubscriptionMatchedStatus for the DataReader.

take (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → list

Take copies of all available samples (data and info).

take_data (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → list

Take copies of all available valid data.

take_loaned (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → *rti.connextdds.PublicationBuiltinTopicData.LoanedSamples*

Take all available samples (data and info) and return them in a loaned container.

property topic_description

Returns the TopicDescription associated with the DataReader.

property topic_name

Get the topic name associated with this DataReader.

property type_name

Get the type name associated with this DataReader.

wait_for_historical_data(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *max_wait:* rti.connextdds.Duration) → None

Waits until all "historical" data is received for DataReaders that have a non-VOLATILE Durability Qos kind.

wait_for_historical_data_async(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *max_wait:* rti.connextdds.Duration) → object

Waits until all "historical" data is received for DataReaders that have a non-VOLATILE Durability Qos kind. This call is awaitable and only for use with asyncio.

class DataReaderListener

Bases: `pybind11_object`

__init__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderListener) → None

__module__ = 'rti.connextdds'

on_data_available(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → None

Data available callback.

on_liveliness_changed(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *arg1:* rti.connextdds.LivelinessChangedStatus) → None

Liveliness changed callback.

on_requested_deadline_missed(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *arg1:* rti.connextdds.RequestedDeadlineMissedStatus) → None

Requested deadline missed callback.

on_requested_incompatible_qos (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *arg1:* rti.connextdds.RequestedIncompatibleQosStatus) → None

Requested incompatible QoS callback.

on_sample_lost (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SampleLostStatus) → None

Sample lost callback.

on_sample_rejected (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SampleRejectedStatus) → None

Sample rejected callback.

on_subscription_matched (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SubscriptionMatchedStatus) → None

Subscription matched callback.

class DataReaderSeq

Bases: `pybind11_object`

__add__ (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq) → *rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq*

__bool__ (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, *x:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, s: slice) -> rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, arg0: int) -> rti.connextdds.PublicationBuiltinTopicData.DataReader

__hash__ = None

__iadd__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq) → *rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq*

__imul__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq) -> None
2. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, arg0: rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq) → Iterator

__len__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq*

__ne__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq) → bool

__rmul__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, arg0: int, arg1: rti.connextdds.PublicationBuiltinTopicData.DataReader) -> None
2. __setitem__(self: rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, arg0: slice, arg1: rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, *x:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq) → None

Clear the contents

count (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, *x:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, L: rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, *i: int*, *x:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq) -> rti.connextdds.PublicationBuiltinTopicData.DataReader

Remove and return the last item

2. pop(self: rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, i: int) -> rti.connextdds.PublicationBuiltinTopicData.DataReader

Remove and return the item at index `i`

remove (*self:* rti.connextdds.PublicationBuiltinTopicData.DataReaderSeq, *x:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class DataWriter

Bases: *IEntity*, *IAnyDataWriter*

__enter__ (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter) → *rti.connextdds.PublicationBuiltinTopicData.DataWriter*

Enter a context for this DataWriter, to be cleaned up on exiting context

__eq__ (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter) → bool

Test for equality.

__exit__ (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for this DataWriter, cleaning up resources.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, topic: rti.connextdds.PublicationBuiltinTopicData.Topic) -> None

Creates a DataWriter in the implicit publisher with default QoS.

2. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, topic: rti.connextdds.PublicationBuiltinTopicData.Topic, qos: rti.connextdds.DataWriterQos, listener: rti.connextdds.PublicationBuiltinTopicData.DataWriterListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Creates a DataWriter in the implicit publisher with specific QoS and optionally a listener.

3. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, pub: rti.connextdds.Publisher, topic: rti.connextdds.PublicationBuiltinTopicData.Topic) -> None

Creates a DataWriter in a publisher with default QoS.

4. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, pub: rti.connextdds.Publisher, topic: rti.connextdds.PublicationBuiltinTopicData.Topic, qos: rti.connextdds.DataWriterQos, listener: rti.connextdds.PublicationBuiltinTopicData.DataWriterListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Creates a DataWriter in a publisher with specific QoS and optionally a listener.

5. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, writer: rti.connextdds.IAnyDataWriter) -> None

Create a typed DataWriter from an AnyDataWriter.

6. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, entity: rti.connextdds.IEntity) -> None

Create a typed DataWriter from an Entity.

__lshift__ (**args*, ***kwargs*)

Overloaded function.

1. __lshift__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, arg0: rti.connextdds.DataWriterQos) -> rti.connextdds.PublicationBuiltinTopicData.DataWriter

Sets the DataWriterQos.

2. __lshift__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, arg0: Tuple[rti.connextdds.PublicationBuiltinTopicData, rti.connextdds.Time]) -> rti.connextdds.PublicationBuiltinTopicData.DataWriter

Writes a paired sample with a timestamp.

3. __lshift__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, arg0: Tuple[rti.connextdds.PublicationBuiltinTopicData, rti.connextdds.InstanceHandle]) -> rti.connextdds.PublicationBuiltinTopicData.DataWriter

Writes a paired sample with an instance handle.

4. __lshift__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, arg0: rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq) -> rti.connextdds.PublicationBuiltinTopicData.DataWriter

Writes a sequence of pairs of samples with timestamps.

5. __lshift__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, arg0: rti.connextdds.PublicationBuiltinTopicDataSeq) -> rti.connextdds.PublicationBuiltinTopicData.DataWriter

Writes a sequence of samples.

6. __lshift__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, arg0: rti.connextdds.PublicationBuiltinTopicData) -> rti.connextdds.PublicationBuiltinTopicData.DataWriter

Writes a sample.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter) → bool

Test for inequality.

__rshift__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg0:* rti.connextdds.DataWriterQos) → *rti.connextdds.PublicationBuiltinTopicData.DataWriter*

Get the DataWriterQos.

assert_liveliness(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter) → None

Manually asserts the liveliness of the DataWriter.

close(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter) → None

Close this DataWriter.

create_data(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter) → *rti.connextdds.PublicationBuiltinTopicData*

Create data of the writer's associated type and initialize it.

property datawriter_cache_status

Get a copy of the cache status for this writer.

property datawriter_protocol_status

Get a copy of the protocol status for this writer.

dispose_instance(**args*, ***kwargs*)

Overloaded function.

1. dispose_instance(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.PublicationBuiltinTopicData.DataWriter

Dispose an instance.

2. dispose_instance(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> rti.connextdds.PublicationBuiltinTopicData.DataWriter

Dispose an instance with a timestamp.

3. dispose_instance(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, params: rti.connextdds.WriteParams) -> None

Dispose an instance with params.

dispose_instance_async(**args*, ***kwargs*)

Overloaded function.

1. dispose_instance_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> object

Dispose an instance.

2. dispose_instance_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> object

Dispose an instance with a timestamp.

3. dispose_instance_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, key_holder: rti.connextdds.PublicationBuiltinTopicData) -> object

Dispose the instance associated with key_holder.

4. dispose_instance_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, key_holder: rti.connextdds.PublicationBuiltinTopicData, timestamp: rti.connextdds.Time) -> object

Dispose the instance associated with key_holder using a timestamp

5. dispose_instance_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, params: rti.connextdds.WriteParams) -> object

Dispose an instance with params.

static find_all_by_topic(*publisher:* rti.connextdds.Publisher, *topic_name: str*) → *rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq*

Retrieve all DataWriters for the given topic name in the publisher.

static find_by_name(**args*, ***kwargs*)

Overloaded function.

1. find_by_name(participant: rti.connextdds.DomainParticipant, name: str) -> Optional[rti.connextdds.PublicationBuiltinTopicData.DataWriter]

Find DataWriter in DomainParticipant with the provided name, returning the first found.

2. find_by_name(publisher: rti.connextdds.Publisher, name: str) -> Optional[rti.connextdds.PublicationBuiltinTopicData.DataWriter]

Find DataWriter in Publisher with the DataReader's name, returning the first found.

static find_by_topic(*publisher:* rti.connextdds.Publisher, *name: str*) → Optional[*rti.connextdds.PublicationBuiltinTopicData.DataWriter*]

Find DataWriter in publisher with a topic name, returning the first found.

flush(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter) → None

Flushes the batch in progress in the context of thecalling thread.

is_matched_subscription_active(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg0:* rti.connextdds.InstanceHandle) → bool

A boolean indicating whether or not the matched subscription is active.

is_sample_app_acknowledged(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *sample_id:* rti.connextdds.SampleIdentity) → bool

Indicates if a sample is considered application-acknowledged.

key_value (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.PublicationBuiltinTopicData*

Retrieve the instance key that corresponds to an instance handle.

property listener

Get the listener associated with the DataWriter or set the listener.

property liveliness_lost_status

Get a copy of the LivelinessLostStatus.

lookup_instance (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *key_holder:* rti.connextdds.PublicationBuiltinTopicData) → *rti.connextdds.InstanceHandle*

Retrieve the instance handle that corresponds to an instance key_holder

matched_subscription_data (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.SubscriptionBuiltinTopicData*

Get the SubscriptionBuiltinTopicData for a subscription matched to this DataWriter.

matched_subscription_datawriter_protocol_status (**args*, ***kwargs*)

Overloaded function.

1. matched_subscription_datawriter_protocol_status(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.DataWriterProtocolStatus

Get a copy of the protocol status for this writer per a matched subscription handle.

2. matched_subscription_datawriter_protocol_status(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, locator: rti.connextdds.Locator) -> rti.connextdds.DataWriterProtocolStatus

Get a copy of the protocol status for this writer per a matched subscription locator.

matched_subscription_participant_data (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ParticipantBuiltinTopicData*

Get the ParticipantBuiltinTopicData for a subscription matched to this DataWriter.

property matched_subscriptions

Get a copy of the list of the currently matched subscription handles.

property matched_subscriptions_locators

The locators used to communicate with matched DataReaders.

property offered_deadline_missed_status

Get a copy of the OfferedDeadlineMissedStatus.

property offered_incompatible_qos_status

Get a copy of the OfferedIncompatibleQosStatus

property publication_matched_status

Get a copy of the PublicationMatchedStatus

property publisher

Get the Publisher that owns this DataWriter.

property qos

The DataWriterQos for this DataWriter.This property's getter returns a deep copy.

register_instance(**args*, ***kwargs*)

Overloaded function.

1. register_instance(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, key_holder: rti.connextdds.PublicationBuiltinTopicData) -> rti.connextdds.InstanceHandle

Informs RTI Connext that the application will be modifying a particular instance.

2. register_instance(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, key_holder: rti.connextdds.PublicationBuiltinTopicData, timestamp: rti.connextdds.Time) -> rti.connextdds.InstanceHandle

Informs RTI Connext that the application will be modifying a particular instance and specified the timestamp.

3. register_instance(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, key_holder: rti.connextdds.PublicationBuiltinTopicData, params: rti.connextdds.WriteParams) -> rti.connextdds.InstanceHandle

Registers instance with parameters.

property reliable_reader_activity_changed_status

Get a copy of the reliable reader activity changed status for this writer.

property reliable_writer_cache_changed_status

Get a copy of the reliable cache status for this writer.

property service_request_accepted_status

Get a copy of the service request accepted status for this writer.

set_listener(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *listener:* rti.connextdds.PublicationBuiltinTopicData.DataWriterListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and event mask for the DataWriter.

property topic

Get the Topic object associated with this DataWriter.

property topic_name

Get the topic name associated with this DataWriter.

property type_name

Get the type name for the topic object associated with this DataWriter.

unregister_instance(**args*, ***kwargs*)

Overloaded function.

1. unregister_instance(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.PublicationBuiltinTopicData.DataWriter

Unregister an instance.

2. unregister_instance(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> rti.connextdds.PublicationBuiltinTopicData.DataWriter

Unregister an instance with timestamp.

3. unregister_instance(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, params: rti.connextdds.WriteParams) -> None

Unregister an instance with parameters.

unregister_instance_async(**args*, ***kwargs*)

Overloaded function.

1. unregister_instance_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> object

Unregister an instance.

2. unregister_instance_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> object

Unregister an instance with timestamp.

3. unregister_instance_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, key_holder: rti.connextdds.PublicationBuiltinTopicData) -> object

Unregister the instance associated with key_holder.

4. unregister_instance_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, key_holder: rti.connextdds.PublicationBuiltinTopicData, timestamp: rti.connextdds.Time) -> object

Unregister the instance associate with key_holder using a timestamp.

5. unregister_instance_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, params: rti.connextdds.WriteParams) -> object

Unregister an instance with parameters.

wait_for_acknowledgments(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *max_wait:* rti.connextdds.Duration) → None

Blocks the calling thread until all data written by a reliable DataWriter is acknowledged or until the timeout expires.

wait_for_asynchronous_publishing(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *max_wait:* rti.connextdds.Duration) → None

This operation blocks the calling thread (up to max_wait) until all data written by the asynchronous DataWriter is sent and acknowledged (if reliable) by all matched DataReader entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; a time out indicates that max_wait elapsed before all the data was sent and/or acknowledged.

In other words, this guarantees that sending to best effort DataReader is complete in addition to what DataWriter.wait_for_acknowledgments() provides.

If the DataWriter does not have PublishMode kind set to PublishModeKind.ASYNCHRONOUS the operation will complete immediately

wait_for_asynchronous_publishing_async (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *max_wait:* rti.connextdds.Duration) → object

This function is awaitable until either a timeout of max_wait or all data written by the asynchronous DataWriter is sent and acknowledged (if reliable) by all matched DataReader entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; a time out indicates that max_wait elapsed before all the data was sent and/or acknowledged.This function works with asyncio.

In other words, this guarantees that sending to best effort DataReader is complete in addition to what DataWriter.wait_for_acknowledgments() provides.

If the DataWriter does not have PublishMode kind set to PublishModeKind.ASYNCHRONOUS the operation will complete immediately

write (**args*, ***kwargs*)

Overloaded function.

1. write(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, samples: rti.connextdds.PublicationBuiltinTopicDataSeq) -> None

Write a sequence of samples.

2. write(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, samples: rti.connextdds.PublicationBuiltinTopicDataSeq, timestamp: rti.connextdds.Time) -> None

Write a sequence of samples with a timestamp.

3. write(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, sample: rti.connextdds.PublicationBuiltinTopicData) -> None

Write a sample.

4. write(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, sample: rti.connextdds.PublicationBuiltinTopicData, timestamp: rti.connextdds.Time) -> None

Write a sample with a specified timestamp.

5. write(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, sample: rti.connextdds.PublicationBuiltinTopicData, handle: rti.connextdds.InstanceHandle) -> None

Write a sample with an instance handle.

6. write(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, sample: rti.connextdds.PublicationBuiltinTopicData, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> None

Write a sample with an instance handle and specified timestamp.

7. write(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, sample: rti.connextdds.PublicationBuiltinTopicData, params: rti.connextdds.WriteParams) -> None

Write with advanced parameters.

write_async (**args*, ***kwargs*)

Overloaded function.

1. write_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, sample: rti.connextdds.PublicationBuiltinTopicData) -> object

Write a sample. This method is awaitable and is only for use with asyncio.

2. write_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, sample: rti.connextdds.PublicationBuiltinTopicData, timestamp: rti.connextdds.Time) -> object

Write a sample with a specified timestamp. This methods is awaitable and only for use with asyncio.

3. write_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, sample: rti.connextdds.PublicationBuiltinTopicData, handle: rti.connextdds.InstanceHandle) -> object

Write a sample with an instance handle. This method is awaitable and only for use with asyncio.

4. write_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, sample: rti.connextdds.PublicationBuiltinTopicData, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> object

Write a sample with an instance handle and specified timestamp. This method is awaitable and only for use with asyncio.

5. write_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, samples: rti.connextdds.PublicationBuiltinTopicDataSeq) -> object

Write a sequence of samples. This method is awaitable and only for use with asyncio.

6. write_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, samples: rti.connextdds.PublicationBuiltinTopicDataSeq, timestamp: rti.connextdds.Time) -> object

Write a sequence of samples with a timestamp. This method is awaitable and only for use with asyncio.

7. write_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, samples: rti.connextdds.PublicationBuiltinTopicDataSeq, handles: rti.connextdds.InstanceHandleSeq) -> object

Write a sequence of samples with their instance handles. This method is awaitable and only for use with asyncio.

8. write_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, samples: rti.connextdds.PublicationBuiltinTopicDataSeq, handles: rti.connextdds.InstanceHandleSeq, timestamp: rti.connextdds.Time) -> object

Write a sequence of samples with their instance handles and a timestamp. This method is awaitable and only for use with asyncio.

9. write_async(self: rti.connextdds.PublicationBuiltinTopicData.DataWriter, sample: rti.connextdds.PublicationBuiltinTopicData, params: rti.connextdds.WriteParams) -> object

Write with advanced parameters.

class DataWriterListener

Bases: `pybind11_object`

__init__ (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterListener) → None

__module__ = 'rti.connextdds'

on_application_acknowledgment (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.AcknowledgmentInfo) → None

On application acknowledgment callback

on_instance_replaced(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.InstanceHandle) → None

On instance replaced callback.

on_liveliness_lost(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.LivelinessLostStatus) → None

Liveliness lost callback.

on_offered_deadline_missed(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.OfferedDeadlineMissedStatus) → None

Offered deadline missed callback.

on_offered_incompatible_qos(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.OfferedIncompatibleQosStatus) → None

Offered incompatible QoS callback.

on_publication_matched(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.PublicationMatchedStatus) → None

Publication matched callback.

on_reliable_reader_activity_changed(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ReliableReaderActivityChangedStatus) → None

Reliable reader activity changed callback.

on_reliable_writer_cache_changed(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ReliableWriterCacheChangedStatus) → None

Reliable writer cache changed callback.

on_service_request_accepted(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ServiceRequestAcceptedStatus) → None

On service request accepted callback.

class DataWriterSeq

Bases: pybind11_object

__add__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq) → *rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq*

__bool__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, *x:* rti.connextdds.PublicationBuiltinTopicData.DataWriter) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, s: slice) -> rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, arg0: int) -> rti.connextdds.PublicationBuiltinTopicData.DataWriter

__hash__ = None

__iadd__(*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq) → *rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq*

__imul__ (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq) -> None
2. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, arg0: rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq) → Iterator

__len__ (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq*

__ne__ (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq) → bool

__rmul__ (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, arg0: int, arg1: rti.connextdds.PublicationBuiltinTopicData.DataWriter) -> None
2. __setitem__(self: rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, arg0: slice, arg1: rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, *x:* rti.connextdds.PublicationBuiltinTopicData.DataWriter) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq) → None

Clear the contents

count (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, *x:* rti.connextdds.PublicationBuiltinTopicData.DataWriter) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, L: rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, *i: int*, *x:* rti.connextdds.PublicationBuiltinTopicData.DataWriter) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq) -> rti.connextdds.PublicationBuiltinTopicData.DataWriter

Remove and return the last item

2. pop(self: rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, i: int) -> rti.connextdds.PublicationBuiltinTopicData.DataWriter

Remove and return the item at index `i`

remove (*self:* rti.connextdds.PublicationBuiltinTopicData.DataWriterSeq, *x:* rti.connextdds.PublicationBuiltinTopicData.DataWriter) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class ITopicDescription

Bases: *IEntity*

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property name

The name of the entity conforming to the ITopicDescription interface.

property participant

The parent DomainParticipant.

property type_name

The name of the associated type.

class LoanedSample

Bases: `pybind11_object`

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PublicationBuiltinTopicData.LoanedSample) -> None

Basic constructor

2. __init__(self: rti.connextdds.PublicationBuiltinTopicData.LoanedSample, data: rti.connextdds.PublicationBuiltinTopicData, info: rti.connextdds.SampleInfo) -> None

Construct LoanedSample with data and info.

__iter__ (*self:* rti.connextdds.PublicationBuiltinTopicData.LoanedSample) → object

__module__ = 'rti.connextdds'

property data

Get the data associated with the sample.

property info

Get the info associated with the sample.

class LoanedSamples

Bases: pybind11_object

__enter__(*self:* rti.connextdds.PublicationBuiltinTopicData.LoanedSamples) → *rti.connextdds.PublicationBuiltinTopicData.LoanedSamples*

Enter a context for the loaned samples, loan returned on context exit.

__exit__(*self:* rti.connextdds.PublicationBuiltinTopicData.LoanedSamples, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for the loaned samples, returning the resources.

__getitem__(*self:* rti.connextdds.PublicationBuiltinTopicData.LoanedSamples, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicData.LoanedSample*

Access a LoanedSample object in an array-like syntax

__init__(*self:* rti.connextdds.PublicationBuiltinTopicData.LoanedSamples) → None

Create an empty LoanedSamples object.

__iter__(*self:* rti.connextdds.PublicationBuiltinTopicData.LoanedSamples) → Iterator

__len__(*self:* rti.connextdds.PublicationBuiltinTopicData.LoanedSamples) → int

Get the number of samples in the loan.

__module__ = 'rti.connextdds'

property length

Get the number of samples in the loan.

return_loan(*self:* rti.connextdds.PublicationBuiltinTopicData.LoanedSamples) → None

Returns the loan to the DataReader.

class NoOpDataReaderListener

Bases: *DataReaderListener*

__init__(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpDataReaderListener) → None

__module__ = 'rti.connextdds'

on_data_available(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReader) → None

Data available callback.

on_liveliness_changed(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *arg1:* rti.connextdds.LivelinessChangedStatus) → None

Liveliness changed callback.

on_requested_deadline_missed(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *arg1:* rti.connextdds.RequestedDeadlineMissedStatus) → None

Requested deadline missed callback.

on_requested_incompatible_qos(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *arg1:* rti.connextdds.RequestedIncompatibleQosStatus) → None

Requested incompatible QoS callback.

on_sample_lost(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SampleLostStatus) → None

Sample lost callback.

on_sample_rejected(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SampleRejectedStatus) → None

Sample rejected callback.

on_subscription_matched(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SubscriptionMatchedStatus) → None

Subscription matched callback.

class NoOpDataWriterListener

Bases: *DataWriterListener*

__init__(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpDataWriterListener) → None

__module__ = 'rti.connextdds'

on_application_acknowledgment(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.AcknowledgmentInfo) → None

On application acknowledgment callback

on_instance_replaced(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.InstanceHandle) → None

On instance replaced callback.

on_liveliness_lost(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.LivelinessLostStatus) → None

Liveliness lost callback.

on_offered_deadline_missed(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.OfferedDeadlineMissedStatus) → None

Offered deadline missed callback.

on_offered_incompatible_qos(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.OfferedIncompatibleQosStatus) → None

Offered incompatible QoS callback.

on_publication_matched(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.PublicationMatchedStatus) → None

Publication matched callback.

on_reliable_reader_activity_changed(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ReliableReaderActivityChangedStatus) → None

Reliable reader activity changed callback.

on_reliable_writer_cache_changed(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ReliableWriterCacheChangedStatus) → None

Reliable writer cache changed callback.

on_service_request_accepted(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ServiceRequestAcceptedStatus) → None

On service request accepted callback.

class NoOpTopicListener

Bases: *TopicListener*

__init__(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpTopicListener) → None

__module__ = 'rti.connextdds'

on_inconsistent_topic(*self:* rti.connextdds.PublicationBuiltinTopicData.NoOpTopicListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.Topic, *arg1:* rti.connextdds.InconsistentTopicStatus) → None

Inconsistent topic callback.

class Sample

Bases: pybind11_object

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PublicationBuiltinTopicData.Sample, data: rti.connextdds.PublicationBuiltinTopicData, info: rti.connextdds.SampleInfo) -> None

Construct Sample with data and info.

2. __init__(self: rti.connextdds.PublicationBuiltinTopicData.Sample, sample: rti.connextdds.PublicationBuiltinTopicData.Sample) -> None

Copy constructor.

3. __init__(self: rti.connextdds.PublicationBuiltinTopicData.Sample) -> None

Basic constructor

4. __init__(self: rti.connextdds.PublicationBuiltinTopicData.Sample, loaned_sample: rti.connextdds.PublicationBuiltinTopicData.LoanedSample) -> None

Construct a sample with a loaned sample.

__iter__(*self:* rti.connextdds.PublicationBuiltinTopicData.Sample) → object

__module__ = 'rti.connextdds'

property data

The data associated with the sample.

property info

Get the info associated with the sample.

class SharedSamples

Bases: pybind11_object

__getitem__(*self:* rti.connextdds.PublicationBuiltinTopicData.SharedSamples, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicData.LoanedSample*

Get the sample at the specified index.

__init__(*self:* rti.connextdds.PublicationBuiltinTopicData.SharedSamples, *loaned_samples:* rti.connextdds.PublicationBuiltinTopicData.LoanedSamples) → None

Constructs an instance of SharedSamples, removing ownership of the loan from the Loaned Samples.

__iter__(*self:* rti.connextdds.PublicationBuiltinTopicData.SharedSamples) → Iterator

Make a sample iterator

__len__(*self:* rti.connextdds.PublicationBuiltinTopicData.SharedSamples) → int

Returns the number of samples.

__module__ = 'rti.connextdds'

class Topic

Bases: *ITopicDescription*, *IAnyTopic*

__eq__(*self:* rti.connextdds.PublicationBuiltinTopicData.Topic, *arg0:* rti.connextdds.PublicationBuiltinTopicData.Topic) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PublicationBuiltinTopicData.Topic, entity: rti.connextdds.IEntity) -> None

Cast an Entity to a Topic.

2. __init__(self: rti.connextdds.PublicationBuiltinTopicData.Topic, topic_description: rti.connextdds.PublicationBuiltinTopicData.ITopicDescription) -> None

Cast an ITopicDescription to a Topic.

3. __init__(self: rti.connextdds.PublicationBuiltinTopicData.Topic, topic: rti.connextdds.IAnyTopic) -> None

Create a typed Topic from an AnyTopic.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.PublicationBuiltinTopicData.Topic, *arg0:* rti.connextdds.PublicationBuiltinTopicData.Topic) → bool

Test for inequality.

static find(*participant:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.PublicationBuiltinTopicData.Topic*]

Look up a Topic by its name in the DomainParticipant.

property inconsistent_topic_status

Get a copy of the current InconsistentTopicStatus for this Topic.

property listener

The listener.

property qos

Get the TopicQos for this Topic.

This property's getter returns a deep copy.

set_listener(*self:* rti.connextdds.PublicationBuiltinTopicData.Topic, *listener:* rti.connextdds.PublicationBuiltinTopicData.TopicListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and event mask.

class TopicDescription

Bases: *ITopicDescription*

__eq__(*self:* rti.connextdds.PublicationBuiltinTopicData.TopicDescription, *arg0:* rti.connextdds.PublicationBuiltinTopicData.TopicDescription) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PublicationBuiltinTopicData.TopicDescription, topic: rti.connextdds.PublicationBuiltinTopicData.ITopicDescription) -> None

Cast a Topic to a TopicDescription.

2. __init__(self: rti.connextdds.PublicationBuiltinTopicData.TopicDescription, entity: rti.connextdds.IEntity) -> None

Cast a Topic to a TopicDescription.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.PublicationBuiltinTopicData.TopicDescription, *arg0:* rti.connextdds.PublicationBuiltinTopicData.TopicDescription) → bool

Test for inequality.

class TopicListener

Bases: pybind11_object

__init__ (*self:* rti.connextdds.PublicationBuiltinTopicData.TopicListener) → None

__module__ = 'rti.connextdds'

on_inconsistent_topic (*self:* rti.connextdds.PublicationBuiltinTopicData.TopicListener, *arg0:* rti.connextdds.PublicationBuiltinTopicData.Topic, *arg1:* rti.connextdds.InconsistentTopicStatus) → None

Inconsistent topic callback.

class TopicSeq

Bases: pybind11_object

__add__ (*self:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq) → *rti.connextdds.PublicationBuiltinTopicData.TopicSeq*

__bool__ (*self:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq, *x:* rti.connextdds.PublicationBuiltinTopicData.Topic) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.PublicationBuiltinTopicData.TopicSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.PublicationBuiltinTopicData.TopicSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.PublicationBuiltinTopicData.TopicSeq, s: slice) -> rti.connextdds.PublicationBuiltinTopicData.TopicSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.PublicationBuiltinTopicData.TopicSeq, arg0: int) -> rti.connextdds.PublicationBuiltinTopicData.Topic

__hash__ = None

__iadd__ (*self:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq) → *rti.connextdds.PublicationBuiltinTopicData.TopicSeq*

__imul__ (*self:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicData.TopicSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PublicationBuiltinTopicData.TopicSeq) -> None
2. __init__(self: rti.connextdds.PublicationBuiltinTopicData.TopicSeq, arg0: rti.connextdds.PublicationBuiltinTopicData.TopicSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.PublicationBuiltinTopicData.TopicSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq) → Iterator

__len__ (*self:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicData.TopicSeq*

__ne__ (*self:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq) → bool

__rmul__ (*self:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicData.TopicSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.PublicationBuiltinTopicData.TopicSeq, arg0: int, arg1: rti.connextdds.PublicationBuiltinTopicData.Topic) -> None
2. __setitem__(self: rti.connextdds.PublicationBuiltinTopicData.TopicSeq, arg0: slice, arg1: rti.connextdds.PublicationBuiltinTopicData.TopicSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq, *x:* rti.connextdds.PublicationBuiltinTopicData.Topic) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq) → None

Clear the contents

count (*self:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq, *x:* rti.connextdds.PublicationBuiltinTopicData.Topic) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.PublicationBuiltinTopicData.TopicSeq, L: rti.connextdds.PublicationBuiltinTopicData.TopicSeq) -> None

Extend the list by appending all the items in the given list
2. extend(self: rti.connextdds.PublicationBuiltinTopicData.TopicSeq, L: Iterable) -> None
Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq, *i: int*, *x:* rti.connextdds.PublicationBuiltinTopicData.Topic) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.
1. pop(self: rti.connextdds.PublicationBuiltinTopicData.TopicSeq) -> rti.connextdds.PublicationBuiltinTopicData.Topic
Remove and return the last item
2. pop(self: rti.connextdds.PublicationBuiltinTopicData.TopicSeq, i: int) -> rti.connextdds.PublicationBuiltinTopicData.Topic
Remove and return the item at index `i`

remove (*self:* rti.connextdds.PublicationBuiltinTopicData.TopicSeq, *x:* rti.connextdds.PublicationBuiltinTopicData.Topic) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class ValidLoanedSamples

Bases: `pybind11_object`

__enter__ (*self:* rti.connextdds.PublicationBuiltinTopicData.ValidLoanedSamples) → *rti.connextdds.PublicationBuiltinTopicData.ValidLoanedSamples*

__exit__ (*self:* rti.connextdds.PublicationBuiltinTopicData.ValidLoanedSamples, *arg0: object*, *arg1: object*, *arg2: object*) → None

__init__ (**args*, ***kwargs*)

__iter__ (*self:* rti.connextdds.PublicationBuiltinTopicData.ValidLoanedSamples) → Iterator

__module__ = 'rti.connextdds'

class WriterContentFilter

Bases: *ContentFilter*

__init__ (*self:* rti.connextdds.PublicationBuiltinTopicData.WriterContentFilter) → None

__module__ = 'rti.connextdds'

writer_attach (*self:* rti.connextdds.PublicationBuiltinTopicData.WriterContentFilter) → Optional[object]

A writer-side filtering API to create some state that can facilitate filtering on the writer side.

writer_compile (*self:* rti.connextdds.PublicationBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *property:* rti.connextdds.ExpressionProperty, *expression: str*, *parameters:* rti.connextdds.StringSeq, *type_code: Optional[*rti.connextdds.DynamicType*]*, *type_class_name: str*, *cookie:* rti.connextdds.Cookie) → None

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching DataReader.

writer_detach (*self:* rti.connextdds.PublicationBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*) → None

A writer-side filtering API to clean up a previously created state using writer_attach.

writer_evaluate (*self:* rti.connextdds.PublicationBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *sample:* rti.connextdds.PublicationBuiltinTopicData, *meta_data:* rti.connextdds.FilterSampleInfo) → *rti.connextdds.CookieVector*

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching DataReader.

writer_finalize (*self:* rti.connextdds.PublicationBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *cookie:* rti.connextdds.Cookie) → None

A writer-side filtering API to clean up a previously compiled instance of the content filter.

writer_return_loan (*self:* rti.connextdds.PublicationBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *cookies:* rti.connextdds.CookieVector) → None

A writer-side filtering API to return the loan on the list of DataReaders returned by writer_evaluate.

class WriterContentFilterHelper

Bases: *WriterContentFilter*

__init__ (*self:* rti.connextdds.PublicationBuiltinTopicData.WriterContentFilterHelper) → None

__module__ = 'rti.connextdds'

add_cookie (*self:* rti.connextdds.PublicationBuiltinTopicData.WriterContentFilterHelper, *cookie:* rti.connextdds.Cookie) → None

A helper function which will add a Cookie to the Cookie sequence that is then returned by the writer_evaluate function.

writer_evaluate_helper (*self:* rti.connextdds.PublicationBuiltinTopicData.WriterContentFilterHelper, *writer_filter_data: Optional[object]*, *sample:* rti.connextdds.PublicationBuiltinTopicData, *meta_data:* rti.connextdds.FilterSampleInfo) → None

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching DataReader.

__eq__ (*self:* rti.connextdds.PublicationBuiltinTopicData, *arg0:* rti.connextdds.PublicationBuiltinTopicData) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.PublicationBuiltinTopicData) → None

Create a default PublicationBuiltinTopicData.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.PublicationBuiltinTopicData, *arg0:* rti.connextdds.PublicationBuiltinTopicData) → bool

Test for inequality.

builtin_topic_name = 'DCPSPublication'

property data_tag

The DataTag policy of the corresponding DataWriter.

property deadline

The Deadline policy of the corresponding DataWriter.

property destination_order

The DestinationOrder policy of the corresponding DataWriter.

property disable_positive_acks

See whether or not a matching DataReader will send positive acknowledgments for reliability.

property durability

The Durability policy of the corresponding DataWriter.

property durability_service

The DurabilityService policy of the corresponding DataWriter.

property group_data

The GroupData policy of the corresponding DataWriter's Publisher.

property key

The DCPS key to distinguish entries.

property latency_budget

The LatencyBudget policy of the corresponding DataWriter.

property lifespan

The Lifespan policy of the corresponding DataWriter.

property liveliness

The Liveliness policy of the corresponding DataWriter.

property locator_filter

The locator filters of the corresponding DataWriter.

property ownership

The Ownership policy of the corresponding DataWriter.

property ownership_strength

The OwnershipStrength policy of the corresponding DataWriter.

property participant_key

The DCPS key of the DomainParticipant to which the DataWriter belongs.

property partition

The Partition policy of the corresponding DataWriter's Publisher.

property presentation

The Presentation policy of the corresponding DataWriter's Publisher.

property product_version

The current version for RTI Connext.

property property

The propagated name-value properties of the corresponding DataWriter.

property publication_name

The publication name and role name.

property publisher_key

The DCPS key of the Publisher to which the DataWriter belongs.

property reliability

The Reliability policy of the corresponding DataWriter.

property representation

The Representation policy of the corresponding DataWriter.

property rtps_protocol_version

The version number of the RTPS wire protocol used.

property rtps_vendor_id

The ID of the vendor implementing the RTPS wire protocol.

property service

The Service policy

property topic_data

The TopicData policy of the corresponding DataWriter's Topic.

property topic_name

The name of the related Topic.

property type

The type.

property type_name

The name of the type attached to the Topic.

property unicast_locators

The custom unicast locators that the endpoint can specify.

property user_data

The UserData policy of the corresponding DataWriter.

property virtual_guid

The virtual Guid associated to the DataWriter.

class rti.connextdds.**PublicationBuiltinTopicDataSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.PublicationBuiltinTopicDataSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicDataSeq) → *rti.connextdds.PublicationBuiltinTopicDataSeq*

__bool__(*self:* rti.connextdds.PublicationBuiltinTopicDataSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.PublicationBuiltinTopicDataSeq, *x:* rti.connextdds.PublicationBuiltinTopicData) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.PublicationBuiltinTopicDataSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.PublicationBuiltinTopicDataSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.PublicationBuiltinTopicDataSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicDataSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.PublicationBuiltinTopicDataSeq, s: slice) -> rti.connextdds.PublicationBuiltinTopicDataSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.PublicationBuiltinTopicDataSeq, arg0: int) -> rti.connextdds.PublicationBuiltinTopicData

__hash__ = None

__iadd__(*self:* rti.connextdds.PublicationBuiltinTopicDataSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicDataSeq) → *rti.connextdds.PublicationBuiltinTopicDataSeq*

__imul__ (*self:* rti.connextdds.PublicationBuiltinTopicDataSeq, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicDataSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PublicationBuiltinTopicDataSeq) -> None
2. __init__(self: rti.connextdds.PublicationBuiltinTopicDataSeq, arg0: rti.connextdds.PublicationBuiltinTopicDataSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.PublicationBuiltinTopicDataSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.PublicationBuiltinTopicDataSeq) → Iterator

__len__ (*self:* rti.connextdds.PublicationBuiltinTopicDataSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.PublicationBuiltinTopicDataSeq, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicDataSeq*

__ne__ (*self:* rti.connextdds.PublicationBuiltinTopicDataSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicDataSeq) → bool

__rmul__ (*self:* rti.connextdds.PublicationBuiltinTopicDataSeq, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicDataSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.PublicationBuiltinTopicDataSeq, arg0: int, arg1: rti.connextdds.PublicationBuiltinTopicData) -> None
2. __setitem__(self: rti.connextdds.PublicationBuiltinTopicDataSeq, arg0: slice, arg1: rti.connextdds.PublicationBuiltinTopicDataSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.PublicationBuiltinTopicDataSeq, *x:* rti.connextdds.PublicationBuiltinTopicData) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.PublicationBuiltinTopicDataSeq) → None

Clear the contents

count (*self:* rti.connextdds.PublicationBuiltinTopicDataSeq, *x:* rti.connextdds.PublicationBuiltinTopicData) → int

Return the number of times `x` appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.PublicationBuiltinTopicDataSeq, L: rti.connextdds.PublicationBuiltinTopicDataSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.PublicationBuiltinTopicDataSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert(*self:* rti.connextdds.PublicationBuiltinTopicDataSeq, *i: int*, *x:* rti.connextdds.PublicationBuiltinTopicData) → None

Insert an item at a given position.

pop(**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.PublicationBuiltinTopicDataSeq) -> rti.connextdds.PublicationBuiltinTopicData

Remove and return the last item

2. pop(self: rti.connextdds.PublicationBuiltinTopicDataSeq, i: int) -> rti.connextdds.PublicationBuiltinTopicData

Remove and return the item at index `i`

remove(*self:* rti.connextdds.PublicationBuiltinTopicDataSeq, *x:* rti.connextdds.PublicationBuiltinTopicData) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**PublicationBuiltinTopicDataTimestampedSeq**

Bases: `pybind11_object`

__add__(*self:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq) → *rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq*

__bool__(*self:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, *x:* *Tuple[*rti.connextdds.PublicationBuiltinTopicData*,* rti.connextdds.Time*]*) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, s: slice) -> rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, arg0: int) -> Tuple[rti.connextdds.PublicationBuiltinTopicData, rti.connextdds.Time]

__hash__ = None

__iadd__ (*self:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq) → *rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq*

__imul__ (*self:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq) -> None

2. __init__(self: rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, arg0: rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq) → Iterator

__len__ (*self:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq*

__ne__ (*self:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, *arg0:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__rmul__ (*self:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, *arg0: int*) → *rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, arg0: int, arg1: Tuple[rti.connextdds.PublicationBuiltinTopicData, rti.connextdds.Time]) -> None

2. __setitem__(self: rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, arg0: slice, arg1: rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, *x:* *Tuple[*rti.connextdds.PublicationBuiltinTopicData, rti.connextdds.Time*]*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq) → None

Clear the contents

count (*self:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, *x:* *Tuple[*rti.connextdds.PublicationBuiltinTopicData, rti.connextdds.Time*]*) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, L: rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, *i: int*, *x:* *Tuple[*rti.connextdds.PublicationBuiltinTopicData, rti.connextdds.Time*]*) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq) -> Tuple[rti.connextdds.PublicationBuiltinTopicData, rti.connextdds.Time]

Remove and return the last item

2. pop(self: rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, i: int) -> Tuple[rti.connextdds.PublicationBuiltinTopicData, rti.connextdds.Time]

Remove and return the item at index i

remove (*self:* rti.connextdds.PublicationBuiltinTopicDataTimestampedSeq, *x:* *Tuple[*rti.connextdds.PublicationBuiltinTopicData, rti.connextdds.Time*]*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**PublicationMatchedStatus**

Bases: pybind11_object

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property current_count

The number of DataReaders that are currently matched with this DataWriter.

property current_count_change

The delta for the current count since the last time the listener fired or the status was read.

property current_count_peak

The highest value that the current count has reached.

property last_subscription_handle

A handle to the DataReader that caused the most recent change to to this status.

property total_count

Total count of times the DataWriter matched with a DataReader.

property total_count_change

The delta in the total count since the last time the listener fired or the status was read.

class rti.connextdds.**PublishMode**

Bases: pybind11_object

PUBLICATION_PRIORITY_UNDEFINED = 0

__eq__ (*self:* rti.connextdds.PublishMode, *arg0:* rti.connextdds.PublishMode) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.PublishMode) → None

Creates a PublishMode qos policy of synchronous kind.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.PublishMode, *arg0:* rti.connextdds.PublishMode) → bool

Test for inequality.

static asynchronous (**args*, ***kwargs*)

Overloaded function.

1. asynchronous() -> rti.connextdds.PublishMode

Creates a PublishMode qos policy of asynchronous kind with default flow controller and undefined priority.

2. asynchronous(flow_controller_name: str) -> rti.connextdds.PublishMode

Creates a PublishMode qos policy of asynchronous kind with a specific flow controller and undefined priority.

3. asynchronous(flow_controller_name: str, priority: int) -> rti.connextdds.PublishMode

Creates a PublishMode qos policy of asynchronous kind with specific flow controller and priority.

property flow_controller_name

The flow controller name associated to a DataWriter.

property kind

The publish mode (synchronous or asynchronous) for a DataWriter.

property priority

The priority of all samples written by a DataWriter.

synchronous = <rti.connextdds.PublishMode object>

class rti.connextdds.**PublishModeKind**

Bases: pybind11_object

ASYNCHRONOUS = <PublishModeKind.ASYNCHRONOUS: 1>

class PublishModeKind

Bases: pybind11_object

Members:

SYNCHRONOUS : Indicates to send data synchronously.

If DataWriterProtocol.push_on_write is true, data is sent immediately in the context of DataWriter.write().

As data is sent immediately in the context of the user thread, no flow control is applied.

ASYNCHRONOUS : Indicates to send data asynchronously.

Configures the DataWriter to delegate the task of data transmission to a separate publishing thread. The DataWriter.write() call does not send the data, but instead schedules the data to be sent later by its associated Publisher.

Each Publisher uses its dedicated publishing thread (AsynchronousPublisher) to send data for all its asynchronous DataWriters. For each asynchronous DataWriter, the associated FlowController determines when the publishing thread is allowed to send the data.

DataWriter.wait_for_asynchronous_publishing and Publisher.wait_for_asynchronous_publishing enable you to determine when the data has actually been sent.

ASYNCHRONOUS = <PublishModeKind.ASYNCHRONOUS: 1>

SYNCHRONOUS = <PublishModeKind.SYNCHRONOUS: 0>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.PublishModeKind.PublishModeKind) → int

__init__(*self:* rti.connextdds.PublishModeKind.PublishModeKind, *value: int*) → None

__int__(*self:* rti.connextdds.PublishModeKind.PublishModeKind) → int

__members__ = {'ASYNCHRONOUS': <PublishModeKind.ASYNCHRONOUS: 1>, 'SYNCHRONOUS': <PublishModeKind.SYNCHRONOUS: 0>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.PublishModeKind.PublishModeKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

SYNCHRONOUS = <PublishModeKind.SYNCHRONOUS: 0>

__eq__(*self:* rti.connextdds.PublishModeKind, *arg0:* rti.connextdds.PublishModeKind) → bool

Apply operator to underlying enumerated values.

__ge__(*self:* rti.connextdds.PublishModeKind, *arg0:* rti.connextdds.PublishModeKind) → bool

Apply operator to underlying enumerated values.

__gt__(*self:* rti.connextdds.PublishModeKind, *arg0:* rti.connextdds.PublishModeKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PublishModeKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.PublishModeKind, arg0: rti.connextdds.PublishModeKind.PublishModeKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.PublishModeKind) → *rti.connextdds.PublishModeKind.PublishModeKind*

Int conversion.

__le__ (*self:* rti.connextdds.PublishModeKind, *arg0:* rti.connextdds.PublishModeKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.PublishModeKind, *arg0:* rti.connextdds.PublishModeKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.PublishModeKind, *arg0:* rti.connextdds.PublishModeKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.PublishModeKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**Publisher**

Bases: *IEntity*

__eq__ (*self:* rti.connextdds.Publisher, *arg0:* rti.connextdds.Publisher) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Publisher, participant: rti.connextdds.DomainParticipant) -> None

Create a publisher.

2. __init__(self: rti.connextdds.Publisher, participant: rti.connextdds.DomainParticipant, qos: rti.connextdds.PublisherQos, listener: rti.connextdds.PublisherListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a Publisher with the desired QoS policies and specified listener

3. __init__(self: rti.connextdds.Publisher, entity: rti.connextdds.IEntity) -> None

Cast an Entity to a Publisher.

__lshift__ (*self:* rti.connextdds.Publisher, *arg0:* rti.connextdds.PublisherQos) → *rti.connextdds.Publisher*

Set the PublisherQos.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Publisher, *arg0:* rti.connextdds.Publisher) → bool

Test for inequality.

__rshift__ (*self:* rti.connextdds.Publisher, *arg0:* rti.connextdds.PublisherQos) → *rti.connextdds.Publisher*

Copy the PublisherQos.

property default_datawriter_qos

The default DataWriterQos.

This property's getter returns a deep copy.

find_datawriter (*self:* rti.connextdds.Publisher, *name: str*) → Optional[*rti.connextdds.AnyDataWriter*]

Find a DataWriter in this Publisher by its name.

find_datawriter_by_topic_name (*self:* rti.connextdds.Publisher, *topic_name: str*) → Optional[*rti.connextdds.AnyDataWriter*]

Find a DataWriter in this Publisher by its topic name. If more than one exists for this Topic, the first one found is returned.

find_datawriters (*self:* rti.connextdds.Publisher) → *rti.connextdds.AnyDataWriterSeq*

Find all DataWriters in the Publisher.

property listener

Get/set the listener.

property participant

Get the parent DomainParticipant of this Publisher.

property qos

The PublisherQos for this Publisher.

This property's getter returns a deep copy.

set_listener (*self:* rti.connextdds.Publisher, *listener:* rti.connextdds.PublisherListener, *event_mask:* rti.connextdds.StatusMask) → None

Bind the listener and event mask to the Publisher.

wait_for_acknowledgments (*self:* rti.connextdds.Publisher, *max_wait:* rti.connextdds.Duration) → None

Blocks until all data written by this Publisher's reliable DataWriters is acknowledged or the timeout expires.

wait_for_asynchronous_publishing (*self:* rti.connextdds.Publisher, *max_wait:* rti.connextdds.Duration) → None

Blocks until asynchronous sending is complete or timeout expires.

class rti.connextdds.**PublisherListener**

Bases: *AnyDataWriterListener*

__init__ (*self:* rti.connextdds.PublisherListener) → None

__module__ = 'rti.connextdds'

class rti.connextdds.**PublisherQos**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.PublisherQos, *arg0:* rti.connextdds.PublisherQos) → bool

Test for equality

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PublisherQos) -> None

Create a PublisherQos with the default value for each policy.

2. __init__(self: rti.connextdds.PublisherQos, publisher: rti.connextdds.Publisher) -> None

Create a PublisherQos with the same settings as those applied to the provided Publisher object.

3. __init__(self: rti.connextdds.PublisherQos, other: rti.connextdds.PublisherQos) -> None

Create a copy of a PublisherQos object.

__lshift__ (**args*, ***kwargs*)

Overloaded function.

1. __lshift__(self: rti.connextdds.PublisherQos, arg0: rti.connextdds.Presentation) -> rti.connextdds.PublisherQos

Set the PresentationQoS.

2. __lshift__(self: rti.connextdds.PublisherQos, arg0: rti.connextdds.Partition) -> rti.connextdds.PublisherQos

Set the PartitionQoS.

3. __lshift__(self: rti.connextdds.PublisherQos, arg0: rti.connextdds.GroupData) -> rti.connextdds.PublisherQos

Set the GroupDataQoS.

4. __lshift__(self: rti.connextdds.PublisherQos, arg0: rti.connextdds.EntityFactory) -> rti.connextdds.PublisherQos

Set the EntityFactoryQoS.

5. __lshift__(self: rti.connextdds.PublisherQos, arg0: rti.connextdds.AsynchronousPublisher) -> rti.connextdds.PublisherQos

Set the AsynchronousPublisherQoS.

6. __lshift__(self: rti.connextdds.PublisherQos, arg0: rti.connextdds.ExclusiveArea) -> rti.connextdds.PublisherQos

Set the ExclusiveAreaQoS.

7. __lshift__(self: rti.connextdds.PublisherQos, arg0: rti.connextdds.EntityName) -> rti.connextdds.PublisherQos

Set the EntityNameQoS.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.PublisherQos, *arg0:* rti.connextdds.PublisherQos) → bool

Test for inequality.

__rshift__(**args*, ***kwargs*)

Overloaded function.

1. __rshift__(self: rti.connextdds.PublisherQos, arg0: rti.connextdds.Presentation) -> rti.connextdds.PublisherQos

Get the PresentationQoS.

2. __rshift__(self: rti.connextdds.PublisherQos, arg0: rti.connextdds.Partition) -> rti.connextdds.PublisherQos

Get the PartitionQoS.

3. __rshift__(self: rti.connextdds.PublisherQos, arg0: rti.connextdds.GroupData) -> rti.connextdds.PublisherQos

Get the GroupDataQoS.

4. __rshift__(self: rti.connextdds.PublisherQos, arg0: rti.connextdds.EntityFactory) -> rti.connextdds.PublisherQos

Get the EntityFactoryQoS.

5. __rshift__(self: rti.connextdds.PublisherQos, arg0: rti.connextdds.AsynchronousPublisher) -> rti.connextdds.PublisherQos

Get the AsynchronousPublisherQoS.

6. __rshift__(self: rti.connextdds.PublisherQos, arg0: rti.connextdds.ExclusiveArea) -> rti.connextdds.PublisherQos

Get the ExclusiveAreaQoS.

7. __rshift__(self: rti.connextdds.PublisherQos, arg0: rti.connextdds.EntityName) -> rti.connextdds.PublisherQos

Get the EntityNameQoS.

__str__(*self:* rti.connextdds.PublisherQos) → str

property asynchronous_publisher

Get/set AsynchronousPublisher QoS.

property entity_factory

Get/set EntityFactory QoS.

property entity_name

Get/set EntityName QoS.

property exclusive_area

Get/set ExclusiveArea QoS.

property group_data

Get/set GroupData QoS.

property partition

Get/set Partition QoS.

property presentation

Get/set Presentation QoS.

to_string(*self:* rti.connextdds.PublisherQos, *format:* rti.connextdds.QosPrintFormat = *QosPrintFormat()*, *base: Optional[*rti.connextdds.PublisherQos*] = None*, *print_all: bool = False*) → str

Convert QoS to string based on params.

class rti.connextdds.**PublisherSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.PublisherSeq, *arg0:* rti.connextdds.PublisherSeq) → *rti.connextdds.PublisherSeq*

__bool__(*self:* rti.connextdds.PublisherSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.PublisherSeq, *x:* rti.connextdds.Publisher) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.PublisherSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.PublisherSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.PublisherSeq, *arg0:* rti.connextdds.PublisherSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.PublisherSeq, s: slice) -> rti.connextdds.PublisherSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.PublisherSeq, arg0: int) -> rti.connextdds.Publisher

__hash__ = None

__iadd__(*self:* rti.connextdds.PublisherSeq, *arg0:* rti.connextdds.PublisherSeq) → *rti.connextdds.PublisherSeq*

__imul__(*self:* rti.connextdds.PublisherSeq, *arg0: int*) → *rti.connextdds.PublisherSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.PublisherSeq) -> None

2. __init__(self: rti.connextdds.PublisherSeq, arg0: rti.connextdds.PublisherSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.PublisherSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.PublisherSeq) → Iterator

__len__(*self:* rti.connextdds.PublisherSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.PublisherSeq, *arg0: int*) → *rti.connextdds.PublisherSeq*

__ne__(*self:* rti.connextdds.PublisherSeq, *arg0:* rti.connextdds.PublisherSeq) → bool

__rmul__(*self:* rti.connextdds.PublisherSeq, *arg0: int*) → *rti.connextdds.PublisherSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.PublisherSeq, arg0: int, arg1: rti.connextdds.Publisher) -> None

2. __setitem__(self: rti.connextdds.PublisherSeq, arg0: slice, arg1: rti.connextdds.PublisherSeq) -> None

Assign list elements using a slice object

append(*self:* rti.connextdds.PublisherSeq, *x:* rti.connextdds.Publisher) → None

Add an item to the end of the list

clear(*self:* rti.connextdds.PublisherSeq) → None

Clear the contents

count(*self:* rti.connextdds.PublisherSeq, *x:* rti.connextdds.Publisher) → int

Return the number of times x appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.PublisherSeq, L: rti.connextdds.PublisherSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.PublisherSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.PublisherSeq, *i: int*, *x:* rti.connextdds.Publisher) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.PublisherSeq) -> rti.connextdds.Publisher

Remove and return the last item

2. pop(self: rti.connextdds.PublisherSeq, i: int) -> rti.connextdds.Publisher

Remove and return the item at index `i`

remove (*self:* rti.connextdds.PublisherSeq, *x:* rti.connextdds.Publisher) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**QosPolicyCount**

Bases: `pybind11_object`

__eq__ (*self:* rti.connextdds.QosPolicyCount, *arg0:* rti.connextdds.QosPolicyCount) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.QosPolicyCount, *arg0:* rti.connextdds.QosPolicyCount) → bool

Test for inequality.

property count

The counter.

property policy

The policy class.

class rti.connextdds.**QosPolicyCountSeq**

Bases: `pybind11_object`

__add__ (*self:* rti.connextdds.QosPolicyCountSeq, *arg0:* rti.connextdds.QosPolicyCountSeq) → *rti.connextdds.QosPolicyCountSeq*

__bool__ (*self:* rti.connextdds.QosPolicyCountSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.QosPolicyCountSeq, *x:* rti.connextdds.QosPolicyCount) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.QosPolicyCountSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.QosPolicyCountSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.QosPolicyCountSeq, *arg0:* rti.connextdds.QosPolicyCountSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.QosPolicyCountSeq, s: slice) -> rti.connextdds.QosPolicyCountSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.QosPolicyCountSeq, arg0: int) -> rti.connextdds.QosPolicyCount

__hash__ = None

__iadd__ (*self:* rti.connextdds.QosPolicyCountSeq, *arg0:* rti.connextdds.QosPolicyCountSeq) → *rti.connextdds.QosPolicyCountSeq*

__imul__ (*self:* rti.connextdds.QosPolicyCountSeq, *arg0: int*) → *rti.connextdds.QosPolicyCountSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.QosPolicyCountSeq) -> None

2. __init__(self: rti.connextdds.QosPolicyCountSeq, arg0: rti.connextdds.QosPolicyCountSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.QosPolicyCountSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.QosPolicyCountSeq) → Iterator

__len__ (*self:* rti.connextdds.QosPolicyCountSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.QosPolicyCountSeq, *arg0: int*) → *rti.connextdds.QosPolicyCountSeq*

__ne__ (*self:* rti.connextdds.QosPolicyCountSeq, *arg0:* rti.connextdds.QosPolicyCountSeq) → bool

__rmul__ (*self:* rti.connextdds.QosPolicyCountSeq, *arg0: int*) → *rti.connextdds.QosPolicyCountSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.QosPolicyCountSeq, arg0: int, arg1: rti.connextdds.QosPolicyCount) -> None

2. __setitem__(self: rti.connextdds.QosPolicyCountSeq, arg0: slice, arg1: rti.connextdds.QosPolicyCountSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.QosPolicyCountSeq, *x:* rti.connextdds.QosPolicyCount) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.QosPolicyCountSeq) → None

Clear the contents

count (*self:* rti.connextdds.QosPolicyCountSeq, *x:* rti.connextdds.QosPolicyCount) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.QosPolicyCountSeq, L: rti.connextdds.QosPolicyCountSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.QosPolicyCountSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.QosPolicyCountSeq, *i: int*, *x:* rti.connextdds.QosPolicyCount) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.QosPolicyCountSeq) -> rti.connextdds.QosPolicyCount

Remove and return the last item

2. pop(self: rti.connextdds.QosPolicyCountSeq, i: int) -> rti.connextdds.QosPolicyCount

Remove and return the item at index `i`

remove (*self:* rti.connextdds.QosPolicyCountSeq, *x:* rti.connextdds.QosPolicyCount) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**QosPrintFormat**

Bases: `pybind11_object`

__eq__ (*self:* rti.connextdds.QosPrintFormat, *arg0:* rti.connextdds.QosPrintFormat) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.QosPrintFormat) -> None

Initializes the properties with the default values.

2. __init__(self: rti.connextdds.QosPrintFormat, indent: int, private: bool, is_standalone: bool) -> None

Initializes the properties.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.QosPrintFormat, *arg0:* rti.connextdds.QosPrintFormat) → bool

Test for inequality.

property indent

The value of indent.

property is_standalone

Print XML preample toggle.

property print_private

Print private QoS field toggle.

class rti.connextdds.**QosProvider**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.QosProvider, *arg0:* rti.connextdds.QosProvider) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.QosProvider, uri: str, profile: str) -> None

Creates a QosProvider fetching QoS configuration from the specified URI.

2. __init__(self: rti.connextdds.QosProvider, uri: str) -> None

Create a QosProvider fetching QoS configuration from the specified URI.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.QosProvider, *arg0:* rti.connextdds.QosProvider) → bool

Test for inequality.

create_participant_from_config(**args*, ***kwargs*)

Overloaded function.

1. create_participant_from_config(self: rti.connextdds.QosProvider, config: str) -> rti.connextdds.DomainParticipant

Create a DomainParticipant given a configuration name from a description provided in an XML configuration file that has been loaded by this QosProvider with default parameters.

2. create_participant_from_config(self: rti.connextdds.QosProvider, config: str, params: rti.connextdds.DomainParticipantConfigParams) -> rti.connextdds.DomainParticipant

Create a DomainParticipant given a configuration name from a description provided in an XML configuration file that has been loaded by this QosProvider.

property datareader_qos

Get a copy of the DataReaderQos currently associated with the QosProvider.

datareader_qos_from_profile (*self:* rti.connextdds.QosProvider, *profile: str*) → *rti.connextdds.DataReaderQos*

Get the DataReaderQos from a qos profile.

property datawriter_qos

Get a copy of the DataWriterQos currently associated with the QosProvider.

datawriter_qos_from_profile (*self:* rti.connextdds.QosProvider, *profile: str*) → *rti.connextdds.DataWriterQos*

Get the DataWriterQos from a qos profile.

default = <rti.connextdds.QosProvider object>

property default_library

The default library associated with this QosProvider (None if not set).

property default_profile

The default profile associated with this QosProvider (None if not set).

property default_profile_library

The library of the default profile associated with this QosProvider (None if not set).

default_provider_params = <rti.connextdds.QosProviderParams object>

get_topic_datareader_qos (*self:* rti.connextdds.QosProvider, *topic_name: str*) → *rti.connextdds.DataReaderQos*

Get the DataReaderQos associated with a given Topic name.

get_topic_datawriter_qos (*self:* rti.connextdds.QosProvider, *topic_name: str*) → *rti.connextdds.DataWriterQos*

Get the DataWriterQos associated with a given Topic name.

get_topic_name_qos (*self:* rti.connextdds.QosProvider, *topic_name: str*) → *rti.connextdds.TopicQos*

Get the TopicQos associated with a given Topic name.

load_profiles (*self:* rti.connextdds.QosProvider) → None

Load the XML QoS profiles from this QosProvider.

property participant_qos

Get a copy of the DomainParticipantQos currently associated with the QosProvider.

participant_qos_from_profile (*self:* rti.connextdds.QosProvider, *profile_name: str*) → *rti.connextdds.DomainParticipantQos*

Get the DomainParticipantQos from a qos profile.

property profiles_loaded

Check if the profiles from this QosProvider have been loaded.

property provider_params

Get a copy of or set the QosProviderParams for this QosProvider.

property publisher_qos

Get a copy of the PublisherQos currently associated with the QosProvider.

publisher_qos_from_profile (*self:* rti.connextdds.QosProvider, *profile: str*) → *rti.connextdds.PublisherQos*

Get the PublisherQos from a qos profile.

property qos_profile_libraries

Get a list of the QoS profile libraries loaded by the QosProvider.

qos_profiles (*self:* rti.connextdds.QosProvider, *library: str*) → *rti.connextdds.StringSeq*

Get a list of the QoS profiles loaded the specified library of the QosProvider.

reload_profiles (*self:* rti.connextdds.QosProvider) → None

Reload the XML QoS profiles from this QosProvider.

static reset_default () → None

Reset the settings of the default QosProvider.

set_topic_datareader_qos (*self:* rti.connextdds.QosProvider, *profile_name: str*, *topic_name: str*) → *rti.connextdds.DataReaderQos*

Set the DataReaderQos for a given Topic name.

set_topic_datawriter_qos (*self:* rti.connextdds.QosProvider, *profile_name: str*, *topic_name: str*) → *rti.connextdds.DataWriterQos*

Set the DataWriterQos for a given Topic name.

set_topic_name_qos (*self:* rti.connextdds.QosProvider, *profile_name: str*, *topic_name: str*) → *rti.connextdds.TopicQos*

Set the TopicQos for a given Topic name.

property subscriber_qos

Get a copy of the SubscriberQos currently associated with this QosProvider.

subscriber_qos_from_profile (*self:* rti.connextdds.QosProvider, *profile: str*) → *rti.connextdds.SubscriberQos*

Get the SubscriberQos from a qos profile.

property topic_qos

Get a copy of the TopicQos currently associated with the QosProvider.

topic_qos_from_profile (*self:* rti.connextdds.QosProvider, *profile_name: str*) → *rti.connextdds.TopicQos*

Get the TopicQos from a qos profile.

type (**args*, ***kwargs*)

Overloaded function.

1. type(self: rti.connextdds.QosProvider, library: str, name: str) -> object

Get a DynamicType from a type library in the QosProvider.

2. type(self: rti.connextdds.QosProvider, name: str) -> object

Get a DynamicType from the QosProvider.

property type_libraries

Get a list of the type libraries loaded by this QosProvider.

unload_profiles (*self:* rti.connextdds.QosProvider) → None

Unload the XML QoS profiles from this QosProvider.

class rti.connextdds.**QosProviderParams**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.QosProviderParams, *arg0:* rti.connextdds.QosProviderParams) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.QosProviderParams) → None

Create a QosProviderParams with default settings.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.QosProviderParams, *arg0:* rti.connextdds.QosProviderParams) → bool

Test for inequality.

property ignore_environment_profile

Choose whether or not to ignore the NDDS_QOS_PROFILES.

property ignore_resource_profile

Choose whether or not to ignore NDDS_QOS_PROFILES.xml.

property ignore_user_profile

Choose whether or not to ignore USER_QOS_PROFILES.xml.

property string_profile

Sequence of strings containing a XML document to load.

property url_profile

Sequence of XML documents to load.

class rti.connextdds.**Query**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.Query, *arg0:* rti.connextdds.Query) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Query, reader: rti.connextdds.IDataReader, expression: str) -> None

Init a Query for a DataReader with an expression.

2. __init__(self: rti.connextdds.Query, reader: rti.connextdds.IDataReader, expression: str, params: rti.connextdds.StringSeq) -> None

Init a Query for a DataReader with an expression and parameters.

__iter__(*self:* rti.connextdds.Query) → Iterator

__len__(*self:* rti.connextdds.Query) → int

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.Query, *arg0:* rti.connextdds.Query) → bool

Test for inequality.

add_parameter(*self:* rti.connextdds.Query, *arg0: str*) → None

Appends a parameter.

property data_reader

The DataReader as an AnyDataReader.

property expression

The expression.

property name

The filter name.

property parameters

The parameters for the expression.

property parameters_length

The parameter sequence length.

class rti.connextdds.**QueryCondition**

Bases: *IReadCondition*

__eq__ (*self:* rti.connextdds.QueryCondition, *arg0:* rti.connextdds.QueryCondition) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.QueryCondition, query: rti.connextdds.Query, status: rti.connextdds.DataState) -> None

Create a QueryCondition.

2. __init__(self: rti.connextdds.QueryCondition, query: rti.connextdds.Query, status: rti.connextdds.DataState, handler: Callable[[rti.connextdds.QueryCondition], None]) -> None

Create a QueryCondition.

3. __init__(self: rti.connextdds.QueryCondition, condition: rti.connextdds.ICondition) -> None

Cast a condition to a QueryCondition.

__len__ (*self:* rti.connextdds.QueryCondition) → int

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.QueryCondition, *arg0:* rti.connextdds.QueryCondition) → bool

Test for inequality.

property expression

The expression.

property parameters

The parameters for the expression.

property parameters_length

The parameter sequence length.

reset_handler (*self:* rti.connextdds.QueryCondition) → None

Resets the handler for this QueryCondition.

set_handler (*self:* rti.connextdds.QueryCondition, *func:* *Callable[[*rti.connextdds.QueryCondition*], None]*) → None

Set a handler function for this QueryCondition.

set_handler_no_args (*self:* rti.connextdds.QueryCondition, *func: Callable[[], None]*) → None

Set a handler function receiving no arguments.

class rti.connextdds.**Rank**

Bases: pybind11_object

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Rank) -> None

Create a default Rank object.

2. __init__(self: rti.connextdds.Rank, sample_rank: int, generation_rank: int, absolute_generation_rank: int) -> None

Create a GenerationCount object with the provided disposed_count and no_writers count.

__module__ = 'rti.connextdds'

property absolute_generation

Get the absolute generation rank of the sample.

property generation

Get the generation rank of the sample.

property sample

Get the sample rank of the sample.

class rti.connextdds.**ReadCondition**

Bases: *IReadCondition*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ReadCondition, reader: rti.connextdds.IAnyDataReader, status: rti.connextdds.DataState) -> None

Create a ReadCondition.

2. __init__(self: rti.connextdds.ReadCondition, reader: rti.connextdds.IAnyDataReader, status: rti.connextdds.DataState, handler: Callable[[rti.connextdds.ReadCondition], None]) -> None

Create a ReadCondition.

3. __init__(self: rti.connextdds.ReadCondition, arg0: rti.connextdds.ICondition) -> None

Cast a compatible Condition to a ReadCondition.

__module__ = 'rti.connextdds'

reset_handler (*self:* rti.connextdds.ReadCondition) → None

Resets the handler for this ReadCondition.

set_handler_no_args (*self:* rti.connextdds.ReadCondition, *func: Callable[[], None]*) → None

Set a handler function receiving no arguments.

class rti.connextdds.**ReaderDataLifecycle**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.ReaderDataLifecycle, *arg0:* rti.connextdds.ReaderDataLifecycle) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ReaderDataLifecycle) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.ReaderDataLifecycle, nowriter_delay: rti.connextdds.Duration, disposed_samples_delay: rti.connextdds.Duration) -> None

Creates an instance with the specified nowriter and disposed-samples purge delays.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ReaderDataLifecycle, *arg0:* rti.connextdds.ReaderDataLifecycle) → bool

Test for inequality.

static auto_purge_disposed_samples (*delay:* rti.connextdds.Duration) → *rti.connextdds.ReaderDataLifecycle*

Returns a policy where only the disposed-samples purge delay is enabled with a specified duration.

static auto_purge_no_writer_samples (*delay:* rti.connextdds.Duration) → *rti.connextdds.ReaderDataLifecycle*

Returns a policy where only the no-writer purge delay is enabled with a specified duration.

property autopurge_disposed_instances_delay

Minimum duration for which the DataReader will maintain an instance once its instance_state becomes InstanceStateKind.NOT_ALIVE_DISPOSED.

This property's getter returns a deep copy.

property autopurge_disposed_samples_delay

Minimum duration for which the DataReader will maintain information regarding an instance once its instance_state becomes InstanceStateKind.NOT_ALIVE_DISPOSED.

This property's getter returns a deep copy.

property autopurge_nowriter_instances_delay

Minimum duration for which the DDSDataReader will maintain information about a received instance once its instance_state becomes DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE and there are no samples for the instance in the DataReader queue.

property autopurge_nowriter_samples_delay

Minimum duration for which the DataReader will maintain information regarding an instance once its instance_state becomes InstanceStateKind.NOT_ALIVE_NO_WRITERS.

This property's getter returns a deep copy.

no_auto_purge = <rti.connextdds.ReaderDataLifecycle object>

class rti.connextdds.**ReceiverPool**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.ReceiverPool, *arg0:* rti.connextdds.ReceiverPool) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ReceiverPool) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.ReceiverPool, thread: rti.connextdds.ThreadSettings, buffer_size: int, buffer_alignment: int) -> None

Creates an instance with the thread settings, buffer size and buffer alignment configuration.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.ReceiverPool, *arg0:* rti.connextdds.ReceiverPool) → bool

Test for inequality.

property buffer_alignment

The receive buffer alignment.

property buffer_size

The length of the buffer used to store the incoming raw data.

property thread

Configures the receiver pool thread(s).

class rti.connextdds.**Reliability**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.Reliability, *arg0:* rti.connextdds.Reliability) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Reliability) -> None

Creates a best-effort policy.

2. __init__(self: rti.connextdds.Reliability, kind: rti.connextdds.ReliabilityKind, max_blocking_time: rti.connextdds.Duration = Duration.from_milliseconds(100)) -> None

Creates an instance with the specified reliability kind and optionally a specific maximum blocking time.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Reliability, *arg0:* rti.connextdds.Reliability) → bool

Test for inequality.

property acknowledgment_kind

The kind of reliable acknowledgment.

best_effort = <rti.connextdds.Reliability object>

property instance_state_consistency_kind

Whether instance state consistency is enabled.

property kind

The reliability kind.

property max_blocking_time

The maximum time a DataWriter may block on a call to write().

This property's getter returns a deep copy.

static reliable(*max_blocking_time:* rti.connextdds.Duration = *Duration.from_milliseconds(100)*) → *rti.connextdds.Reliability*

Creates a policy with ReliabilityKind.RELIABLE and optionally a max blocking time.

class rti.connextdds.**ReliabilityKind**

Bases: pybind11_object

BEST_EFFORT = <ReliabilityKind.BEST_EFFORT: 0>

RELIABLE = <ReliabilityKind.RELIABLE: 1>

class ReliabilityKind

Bases: pybind11_object

Members:

BEST_EFFORT : Indicates that it is acceptable to not retry propagation of any samples.

Presumably new values for the samples are generated often enough that it is not necessary to re-send or acknowledge any samples.

[default] for DataReader and Topic.

RELIABLE : Specifies that RTI Connext will attempt to deliver all samples in its history. Missed samples may be retried.

In steady-state (no modifications communicated via the DataWriter), RTI Connext guarantees that all samples in the DataWriter history will eventually be delivered to all the DataReader objects (subject to timeouts that indicate loss of communication with a particular Subscriber).

Outside steady state, the HISTORY and RESOURCE_LIMITS policies will determine how samples become part of the history and whether samples can be discarded from it.

[default] for DataWriter.

BEST_EFFORT = <ReliabilityKind.BEST_EFFORT: 0>

RELIABLE = <ReliabilityKind.RELIABLE: 1>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.ReliabilityKind.ReliabilityKind) → int

__init__(*self:* rti.connextdds.ReliabilityKind.ReliabilityKind, *value: int*) → None

__int__(*self:* rti.connextdds.ReliabilityKind.ReliabilityKind) → int

__members__ = {'BEST_EFFORT': <ReliabilityKind.BEST_EFFORT: 0>, 'RELIABLE': <ReliabilityKind.RELIABLE: 1>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.ReliabilityKind.ReliabilityKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

__eq__(*self:* rti.connextdds.ReliabilityKind, *arg0:* rti.connextdds.ReliabilityKind) → bool

Apply operator to underlying enumerated values.

__ge__(*self:* rti.connextdds.ReliabilityKind, *arg0:* rti.connextdds.ReliabilityKind) → bool

Apply operator to underlying enumerated values.

__gt__(*self:* rti.connextdds.ReliabilityKind, *arg0:* rti.connextdds.ReliabilityKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ReliabilityKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.ReliabilityKind, arg0: rti.connextdds.ReliabilityKind.ReliabilityKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.ReliabilityKind) → *rti.connextdds.ReliabilityKind.ReliabilityKind*

Int conversion.

__le__ (*self:* rti.connextdds.ReliabilityKind, *arg0:* rti.connextdds.ReliabilityKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.ReliabilityKind, *arg0:* rti.connextdds.ReliabilityKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ReliabilityKind, *arg0:* rti.connextdds.ReliabilityKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.ReliabilityKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**ReliableReaderActivityChangedStatus**

Bases: pybind11_object

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property active_count

The current number of reliable DataReaders currently matched with this reliable DataWriter.

property inactive_count

The number of reliable DataReaders that have been dropped by this reliable DataWriter because they failed to send acknowledgments in a timely fashion.

property last_instance_handle

The instance handle of the last reliable remote DataReader to be determined inactive.

class rti.connextdds.**ReliableWriterCacheChangedStatus**

Bases: pybind11_object

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property empty_reliable_writer_cache

The number of times the reliable DataWriters's cache of unacknowledged samples has become empty.

property full_reliable_writer_cache

The number of times the reliable DataWriters's cache (or send window) of unacknowledged samples has become full.

property high_watermark_reliable_writer_cache

The number of times the reliable DataWriter's cache of unacknowledged samples has risen to the high watermark.

property low_watermark_reliable_writer_cache

The number of times the reliable DataWriter's cache of unacknowledged samples has fallen to the low watermark.

property replaced_unacknowledged_sample_count

The number of unacknowledged samples that have been replaced in the writer's cache.

property unacknowledged_sample_count

The current number of unacknowledged samples in the DataWriter's cache.

property unacknowledged_sample_count_peak

The highest value that unacknowledged_sample_count has reached until now.

class rti.connextdds.**RemoteParticipantPurgeKind**

Bases: pybind11_object

LIVELINESS_BASED = <RemoteParticipantPurgeKind.LIVELINESS_BASED: 0>

NO_PURGE = <RemoteParticipantPurgeKind.NO_PURGE: 1>

class RemoteParticipantPurgeKind

Bases: pybind11_object

Members:

LIVELINESS_BASED : [default] Maintain knowledge of the remote participant for as long as it maintains its liveliness contract.

A participant will continue attempting communication with its peers, even if discovery communication with them is lost, as long as the remote participants maintain their liveliness. If both discovery communication and participant liveliness are lost, however, the local participant will remove all records of the remote participant and its contained endpoints, and no further data communication with them will occur until and unless they are rediscovered.

The liveliness contract a participant promises to its peers (its "liveliness lease duration") is specified in its DiscoveryConfig.participant_liveliness_lease_duration QoS field. It maintains that contract by writing data to those other participants with a writer that has a LivelinessKind of LivelinessKind.AUTOMATIC or LivelinessKind.MANUAL_BY_PARTICIPANT and by asserting itself (at the DiscoveryConfig.participant_liveliness_assert_period) over the Simple Discovery Protocol.

NO_PURGE : Never "forget" a remote participant with which discovery communication has been lost.

If a participant with this behavior loses discovery communication with a remote participant, it will nevertheless remember that remote participant and its endpoints and continue attempting to communicate with them indefinitely.

This value has consequences for a participant's resource usage. If discovery communication with a remote participant is lost, but the same participant is later rediscovered, any relevant records that remain in the database will be reused. However, if it is not rediscovered, the records will continue to take up space in the database for as long as the local participant remains in existence.

LIVELINESS_BASED =
<RemoteParticipantPurgeKind.LIVELINESS_BASED: 0>

NO_PURGE = <RemoteParticipantPurgeKind.NO_PURGE: 1>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.RemoteParticipantPurgeKind.RemoteParticipantPurgeKind) → int

__init__(*self:* rti.connextdds.RemoteParticipantPurgeKind.RemoteParticipantPurgeKind, *value: int*) → None

__int__(*self:* rti.connextdds.RemoteParticipantPurgeKind.RemoteParticipantPurgeKind) → int

__members__ = {'LIVELINESS_BASED':
<RemoteParticipantPurgeKind.LIVELINESS_BASED: 0>, 'NO_PURGE':
<RemoteParticipantPurgeKind.NO_PURGE: 1>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.RemoteParticipantPurgeKind.RemoteParticipantPurgeKind, *state: int*) → None

__str__()
name(self: handle) -> str

property name

property value

__eq__ (*self:* rti.connextdds.RemoteParticipantPurgeKind, *arg0:* rti.connextdds.RemoteParticipantPurgeKind) → bool

Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.RemoteParticipantPurgeKind, *arg0:* rti.connextdds.RemoteParticipantPurgeKind) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.RemoteParticipantPurgeKind, *arg0:* rti.connextdds.RemoteParticipantPurgeKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.RemoteParticipantPurgeKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.RemoteParticipantPurgeKind, arg0: rti.connextdds.RemoteParticipantPurgeKind.RemoteParticipantPurgeKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.RemoteParticipantPurgeKind) → *rti.connextdds.RemoteParticipantPurgeKind.RemoteParticipantPurgeKind*

Int conversion.

__le__ (*self:* rti.connextdds.RemoteParticipantPurgeKind, *arg0:* rti.connextdds.RemoteParticipantPurgeKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.RemoteParticipantPurgeKind, *arg0:* rti.connextdds.RemoteParticipantPurgeKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.RemoteParticipantPurgeKind, *arg0:* rti.connextdds.RemoteParticipantPurgeKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.RemoteParticipantPurgeKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**RequestedDeadlineMissedStatus**

Bases: pybind11_object

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property last_instance_handle

Handle to the last instance in the DataReader for which a missed deadline was detected.

property total_count

Total count of the missed deadlines detected for any instance read by the DataReader.

property total_count_change

The delta in missed deadlines detected since the last time the listener was called or the status was read.

class rti.connextdds.**RequestedIncompatibleQosStatus**

Bases: pybind11_object

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property last_policy

The policy class of one of the policies that was found to be incompatible the last time an incompatibility was detected.

property policies

A list containing, for each policy, the total number of times that the concerned DataReader discovered a DataWriter for the same Topic with an offered QoS that is incompatible with that requested by the DataReader.

total_count(*self:* rti.connextdds.RequestedIncompatibleQosStatus) → int

Total count of how many times the concerned DataReader discovered a DataWriter for the same Topic with an offered QoS that is incompatible with that requested by the DataReader.

property total_count_change

The delta in total_count since the last time the listener was called or the status was read.

class rti.connextdds.**ResourceLimits**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.ResourceLimits, *arg0:* rti.connextdds.ResourceLimits) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ResourceLimits) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.ResourceLimits, max_samples: int, max_instances: int, max_samples_per_instance: int) -> None

Creates the default policy.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.ResourceLimits, *arg0:* rti.connextdds.ResourceLimits) → bool

Test for inequality.

property initial_instances

The number of instances that a DataWriter or a DataWriter will preallocate.

property initial_samples

The maximum number of data samples per instance that a DataWriter or a DataReader can manage.

property instance_hash_buckets

The number of instances that a DataWriter or a DataWriter will preallocate.

property max_instances

Sets the maximum number of instances that a DataWriter or a DataReader can manage.

property max_samples

Sets the maximum number of data samples that a DataWriter or a DataReader can manage across all instances.

property max_samples_per_instance

Sets the maximum number of data samples per instance that a DataWriter or a DataReader can manage.

class rti.connextdds.**RtpsReliableReaderProtocol**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.RtpsReliableReaderProtocol, *arg0:* rti.connextdds.RtpsReliableReaderProtocol) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.RtpsReliableReaderProtocol) → None

Create an RtpsReliableReaderProtocol policy with default settings.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.RtpsReliableReaderProtocol, *arg0:* rti.connextdds.RtpsReliableReaderProtocol) → bool

Test for inequality.

property app_ack_period

Get/set the period at which application-level acknowledgment messages are sent.

property heartbeat_suppression_duration

Get/set the duration to ignore consecutive heartbeats from a DataWriter.

property max_heartbeat_response_delay

Get/set the maximum time to respond with an ACKNACK to a DataWriter's heartbeat.

property min_heartbeat_response_delay

Get/set the minimum time to respond with an ACKNACK to a DataWriter's heartbeat.

property nack_period

Get/set the period at which to send NACKs for receiving historical data.

property receive_window_size

Get/set the number of received out-of-order samples a reader can keep at a time.

property round_trip_time

Get/set the estimated duration from sending a NACK to receiving a repair of a sample.

property samples_per_app_ack

Get/set the minimum number of samples acknowledged by one application-level acknowledgment message.

class rti.connextdds.**RtpsReliableWriterProtocol**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.RtpsReliableWriterProtocol, *arg0:* rti.connextdds.RtpsReliableWriterProtocol) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.RtpsReliableWriterProtocol) → None

Create a default RtpsReliableWriterProtocol policy.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.RtpsReliableWriterProtocol, *arg0:* rti.connextdds.RtpsReliableWriterProtocol) → bool

Test for inequality.

property disable_positive_acks_decrease_sample_keep_duration_factor

Controls rate of contraction of dynamic sample keep duration.

property disable_positive_acks_enable_adaptive_sample_keep_duration

Enables dynamic adjustment of sample keep duration in response to congestion.

property disable_positive_acks_increase_sample_keep_duration_factor

Controls rate of growth of dynamic sample keep duration.

property disable_positive_acks_max_sample_keep_duration

The maximum duration a sample is queued for ACK-disabled readers.

This property's getter returns a deep copy.

property disable_positive_acks_min_sample_keep_duration

The minimum duration a sample is queued for ACK-disabled readers.

This property's getter returns a deep copy.

property disable_repair_piggyback_heartbeat

Prevents piggyback heartbeats from being sent with repair samples.

property enable_multicast_periodic_heartbeat

Whether periodic heartbeat messages are sent over multicast.

property fast_heartbeat_period

An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.

This property's getter returns a deep copy.

property heartbeat_period

The period at which to send heartbeats.

This property's getter returns a deep copy.

property heartbeats_per_max_samples

The number of heartbeats per current send window.

property high_watermark

When the number of unacknowledged samples in the current send window of a reliable writer meets or exceeds this threshold, StatusMask.reliable_writer_cache_changed is considered to have changed.

property inactivate_nonprogressing_readers

Whether to treat remote readers as inactive when their NACKs do not progress.

property late_joiner_heartbeat_period

An alternative heartbeat period used when a reliable reader joins late and needs to be caught up on cached samples of a reliable writer more quickly than the normal heartbeat rate.

This property's getter returns a deep copy.

property low_watermark

When the number of unacknowledged samples in the current send window of a reliable writer meets or falls below this threshold, the StatusMask.reliable_writer_cache_changed is considered to have changed.

property max_bytes_per_nack_response

The maximum total message size when resending dropped samples.

property max_heartbeat_retries

The maximum number of periodic heartbeat retries before marking a remote reader as inactive.

property max_nack_response_delay

The maximum delay to respond to a NACK.

This property's getter returns a deep copy.

property max_send_window_size

Maximum size of send window of unacknowledged samples.

property min_nack_response_delay

The minimum delay to respond to a NACK.

This property's getter returns a deep copy.

property min_send_window_size

Minimum size of send window of unacknowledged samples.

property multicast_resend_threshold

The minimum number of requesting readers needed to trigger a multicast resend.

property nack_suppression_duration

The duration for ignoring consecutive NACKs that may trigger redundant repairs.

This property's getter returns a deep copy.

property samples_per_virtual_heartbeat

The number of samples that a reliable writer has to publish before sending a virtual heartbeat.

property send_window_decrease_factor

Decreases send window size by this percentage when reacting dynamically to network conditions.

property send_window_increase_factor

Increases send window size by this percentage when reacting dynamically to network conditions.

property send_window_update_period

Period in which send window may be dynamically changed.

This property's getter returns a deep copy.

property virtual_heartbeat_period

The period at which to send virtual heartbeats. Virtual heartbeats inform the reliable reader about the range of samples currently present, for each virtual GUID, in the reliable writer's queue.

This property's getter returns a deep copy.

class rti.connextdds.**RtpsReservedPortKindMask**

Bases: pybind11_object

ALL = 1111

BUILTIN_MULTICAST = 0010

BUILTIN_UNICAST = 0001

DEFAULT_MASK = 0111

NONE = 0000

USER_MULTICAST = 1000

USER_UNICAST = 0100

__and__(*self:* rti.connextdds.RtpsReservedPortKindMask, *arg0:* rti.connextdds.RtpsReservedPortKindMask) → *rti.connextdds.RtpsReservedPortKindMask*

Bitwise logical AND of masks.

__bool__(*self:* rti.connextdds.RtpsReservedPortKindMask) → bool

Test if any bits are set.

__contains__(*self:* rti.connextdds.RtpsReservedPortKindMask, *arg0:* rti.connextdds.RtpsReservedPortKindMask) → bool

__eq__(**args*, ***kwargs*)

Overloaded function.

1. __eq__(self: rti.connextdds.RtpsReservedPortKindMask, arg0: rti.connextdds.RtpsReservedPortKindMask) -> bool

Compare masks for equality.

2. __eq__(self: rti.connextdds.RtpsReservedPortKindMask, arg0: rti.connextdds.RtpsReservedPortKindMask) -> bool

Test for equality.

__getitem__(*self:* rti.connextdds.RtpsReservedPortKindMask, *arg0: int*) → bool

Get individual mask bit.

__hash__ = None

__iand__(*self:* rti.connextdds.RtpsReservedPortKindMask, *arg0:* rti.connextdds.RtpsReservedPortKindMask) → *rti.connextdds.RtpsReservedPortKindMask*

Set mask to logical AND with another mask.

__ilshift__(*self:* rti.connextdds.RtpsReservedPortKindMask, *arg0: int*) → *rti.connextdds.RtpsReservedPortKindMask*

Left shift bits in mask.

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.RtpsReservedPortKindMask) -> None

Create a RtpsReservedPortKindMask equivalent to RtpsReservedPortKindMask.NONE

2. __init__(self: rti.connextdds.RtpsReservedPortKindMask, value: int) -> None

Creates a mask from the bits in an integer.

__int__(*self:* rti.connextdds.RtpsReservedPortKindMask) → int

Convert mask to int.

__ior__ (*self:* rti.connextdds.RtpsReservedPortKindMask, *arg0:* rti.connextdds.RtpsReservedPortKindMask) → *rti.connextdds.RtpsReservedPortKindMask*

Set mask to logical OR with another mask.

__irshift__ (*self:* rti.connextdds.RtpsReservedPortKindMask, *arg0: int*) → *rti.connextdds.RtpsReservedPortKindMask*

Right shift bits in mask.

__ixor__ (*self:* rti.connextdds.RtpsReservedPortKindMask, *arg0:* rti.connextdds.RtpsReservedPortKindMask) → *rti.connextdds.RtpsReservedPortKindMask*

Set mask to logical XOR with another mask.

__lshift__ (*self:* rti.connextdds.RtpsReservedPortKindMask, *arg0: int*) → *rti.connextdds.RtpsReservedPortKindMask*

Left shift bits in mask.

__module__ = 'rti.connextdds'

__ne__ (**args*, ***kwargs*)

Overloaded function.

1. __ne__(self: rti.connextdds.RtpsReservedPortKindMask, arg0: rti.connextdds.RtpsReservedPortKindMask) -> bool

Compare masks for inequality.

2. __ne__(self: rti.connextdds.RtpsReservedPortKindMask, arg0: rti.connextdds.RtpsReservedPortKindMask) -> bool

Test for inequality.

__or__ (*self:* rti.connextdds.RtpsReservedPortKindMask, *arg0:* rti.connextdds.RtpsReservedPortKindMask) → *rti.connextdds.RtpsReservedPortKindMask*

Bitwise logical OR of masks.

__repr__ (*self:* rti.connextdds.RtpsReservedPortKindMask) → str

Convert mask to string.

__rshift__ (*self:* rti.connextdds.RtpsReservedPortKindMask, *arg0: int*) → *rti.connextdds.RtpsReservedPortKindMask*

Right shift bits in mask.

__setitem__ (*self:* rti.connextdds.RtpsReservedPortKindMask, *arg0: int*, *arg1: bool*) → None

Set individual mask bit

__str__ (*self:* rti.connextdds.RtpsReservedPortKindMask) → str

Convert mask to string.

__xor__ (*self:* rti.connextdds.RtpsReservedPortKindMask, *arg0:* rti.connextdds.RtpsReservedPortKindMask) → *rti.connextdds.RtpsReservedPortKindMask*

Bitwise logical XOR of masks.

property count

Returns the number of bits set in the mask.

flip(**args*, ***kwargs*)

Overloaded function.

1. flip(self: rti.connextdds.RtpsReservedPortKindMask) -> rti.connextdds.RtpsReservedPortKindMask

Flip all bits in the mask.

2. flip(self: rti.connextdds.RtpsReservedPortKindMask, pos: int) -> rti.connextdds.RtpsReservedPortKindMask

Flip the mask bit at the specified position.

reset(**args*, ***kwargs*)

Overloaded function.

1. reset(self: rti.connextdds.RtpsReservedPortKindMask) -> rti.connextdds.RtpsReservedPortKindMask

Clear all bits in the mask.

2. reset(self: rti.connextdds.RtpsReservedPortKindMask, pos: int) -> rti.connextdds.RtpsReservedPortKindMask

Clear the mask bit at the specified position.

set(**args*, ***kwargs*)

Overloaded function.

1. set(self: rti.connextdds.RtpsReservedPortKindMask) -> rti.connextdds.RtpsReservedPortKindMask

Set all bits in the mask.

2. set(self: rti.connextdds.RtpsReservedPortKindMask, pos: int, value: bool = True) -> rti.connextdds.RtpsReservedPortKindMask

Set the mask bit at the specified position to the provided value (default: true).

property size

Returns the number of bits in the mask type.

test(*self:* rti.connextdds.RtpsReservedPortKindMask, *pos: int*) → bool

Test whether the mask bit at position "pos" is set.

test_all(*self:* rti.connextdds.RtpsReservedPortKindMask) → bool

Test if all bits are set.

test_any(*self:* rti.connextdds.RtpsReservedPortKindMask) → bool

Test if any bits are set.

test_none(*self:* rti.connextdds.RtpsReservedPortKindMask) → bool

Test if none of the bits are set.

class rti.connextdds.**RtpsWellKnownPorts**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.RtpsWellKnownPorts, *arg0:* rti.connextdds.RtpsWellKnownPorts) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.RtpsWellKnownPorts) -> None

Creates an instance that contains the default RTPS well-known ports.

2. __init__(self: rti.connextdds.RtpsWellKnownPorts, port_base: int, domain_id_gain: int, participant_id_gain: int, builtin_multicast_port_offset: int, builtin_unicast_port_offset: int, user_multicast_port_offset: int, user_unicast_port_offset: int) -> None

Creates an instance with the specified ports.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.RtpsWellKnownPorts, *arg0:* rti.connextdds.RtpsWellKnownPorts) → bool

Test for inequality.

backwards_compatible = <rti.connextdds.RtpsWellKnownPorts object>

property builtin_multicast_port_offset

Additional offset for metatraffic multicast port.

property builtin_unicast_port_offset

Additional offset for metatraffic unicast port.

property domain_id_gain

Tunable domain gain parameter.

interoperable = <rti.connextdds.RtpsWellKnownPorts object>

property participant_id_gain

Tunable participant gain parameter.

property port_base

The port base offset.

property user_multicast_port_offset

Additional offset for usertraffic multicast port.

property user_unicast_port_offset

Additional offset for usertraffic unicast port.

class rti.connextdds.**SampleFlag**

Bases: pybind11_object

INTERMEDIATE_REPLY_SEQUENCE = <rti.connextdds.SampleFlag object>

INTERMEDIATE_TOPIC_QUERY_SAMPLE = <rti.connextdds.SampleFlag object>

LAST_SHARED_READER_QUEUE = <rti.connextdds.SampleFlag object>

REDELIVERED = <rti.connextdds.SampleFlag object>

REPLICATE = <rti.connextdds.SampleFlag object>

__and__(*self:* rti.connextdds.SampleFlag, *arg0:* rti.connextdds.SampleFlag) → *rti.connextdds.SampleFlag*

Bitwise logical AND of masks.

__bool__(*self:* rti.connextdds.SampleFlag) → bool

Test if any bits are set.

__contains__(*self:* rti.connextdds.SampleFlag, *arg0:* rti.connextdds.SampleFlag) → bool

__eq__(*self:* rti.connextdds.SampleFlag, *arg0:* rti.connextdds.SampleFlag) → bool

Compare masks for equality.

__getitem__(*self:* rti.connextdds.SampleFlag, *arg0: int*) → bool

Get individual mask bit.

__hash__ = None

__iand__(*self:* rti.connextdds.SampleFlag, *arg0:* rti.connextdds.SampleFlag) → *rti.connextdds.SampleFlag*

Set mask to logical AND with another mask.

__ilshift__(*self:* rti.connextdds.SampleFlag, *arg0: int*) → *rti.connextdds.SampleFlag*

Left shift bits in mask.

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SampleFlag) -> None

Construct an empty SampleFlag with no bits set.

2. __init__(self: rti.connextdds.SampleFlag, value: int) -> None

Creates a mask from the bits in an integer.

__int__(*self:* rti.connextdds.SampleFlag) → int

Convert mask to int.

__ior__ (*self:* rti.connextdds.SampleFlag, *arg0:* rti.connextdds.SampleFlag) → *rti.connextdds.SampleFlag*

Set mask to logical OR with another mask.

__irshift__ (*self:* rti.connextdds.SampleFlag, *arg0: int*) → *rti.connextdds.SampleFlag*

Right shift bits in mask.

__ixor__ (*self:* rti.connextdds.SampleFlag, *arg0:* rti.connextdds.SampleFlag) → *rti.connextdds.SampleFlag*

Set mask to logical XOR with another mask.

__lshift__ (*self:* rti.connextdds.SampleFlag, *arg0: int*) → *rti.connextdds.SampleFlag*

Left shift bits in mask.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.SampleFlag, *arg0:* rti.connextdds.SampleFlag) → bool

Compare masks for inequality.

__or__ (*self:* rti.connextdds.SampleFlag, *arg0:* rti.connextdds.SampleFlag) → *rti.connextdds.SampleFlag*

Bitwise logical OR of masks.

__rshift__ (*self:* rti.connextdds.SampleFlag, *arg0: int*) → *rti.connextdds.SampleFlag*

Right shift bits in mask.

__setitem__ (*self:* rti.connextdds.SampleFlag, *arg0: int*, *arg1: bool*) → None

Set individual mask bit

__str__ (*self:* rti.connextdds.SampleFlag) → str

__xor__ (*self:* rti.connextdds.SampleFlag, *arg0:* rti.connextdds.SampleFlag) → *rti.connextdds.SampleFlag*

Bitwise logical XOR of masks.

property count

Returns the number of bits set in the mask.

flip (**args*, ***kwargs*)

Overloaded function.

1. flip(self: rti.connextdds.SampleFlag) -> rti.connextdds.SampleFlag

Flip all bits in the mask.

2. flip(self: rti.connextdds.SampleFlag, pos: int) -> rti.connextdds.SampleFlag

Flip the mask bit at the specified position.

reset (**args*, ***kwargs*)

Overloaded function.

1. reset(self: rti.connextdds.SampleFlag) -> rti.connextdds.SampleFlag

Clear all bits in the mask.

2. reset(self: rti.connextdds.SampleFlag, pos: int) -> rti.connextdds.SampleFlag

Clear the mask bit at the specified position.

set (**args*, ***kwargs*)

Overloaded function.

1. set(self: rti.connextdds.SampleFlag) -> rti.connextdds.SampleFlag

Set all bits in the mask.

2. set(self: rti.connextdds.SampleFlag, pos: int, value: bool = True) -> rti.connextdds.SampleFlag

Set the mask bit at the specified position to the provided value (default: true).

property size

Returns the number of bits in the mask type.

test (*self:* rti.connextdds.SampleFlag, *pos: int*) → bool

Test whether the mask bit at position "pos" is set.

test_all (*self:* rti.connextdds.SampleFlag) → bool

Test if all bits are set.

test_any (*self:* rti.connextdds.SampleFlag) → bool

Test if any bits are set.

test_none (*self:* rti.connextdds.SampleFlag) → bool

Test if none of the bits are set.

class rti.connextdds.**SampleIdentity**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.SampleIdentity, *arg0:* rti.connextdds.SampleIdentity) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SampleIdentity) -> None

Create a default SampleIdentity object.

2. __init__(self: rti.connextdds.SampleIdentity, writer_guid: rti.connextdds.Guid, sequence_number: rti.connextdds.SequenceNumber) -> None

Create a SampleIdentity with the provided Guid and SequenceNumber.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.SampleIdentity, *arg0:* rti.connextdds.SampleIdentity) → bool

Test for inequality.

__str__ (*self:* rti.connextdds.SequenceNumber) → str

automatic = <rti.connextdds.SampleIdentity object>

property sequence_number

Get the monotonically increasing 64-bit SequenceNumber that identifies the sample in the data source.

unknown = <rti.connextdds.SampleIdentity object>

property writer_guid

Get the 16-byte identifier identifying the virtual Guid.

class rti.connextdds.**SampleInfo**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.SampleInfo, *arg0:* rti.connextdds.SampleInfo) → bool

Check if two SampleInfo are equal.

__hash__ = None

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.SampleInfo, *arg0:* rti.connextdds.SampleInfo) → bool

Check if two SampleInfo are not equal.

__repr__ (*self:* rti.connextdds.SampleInfo) → str

property coherent_set_info

TWhen set, this field provides the information about the coherent set associated with the sample.

property encapsulation_id

The encapsulation kind.

property flag

Flags associated with the sample.

property generation_count

The GenerationCount of the sample.

property instance_handle

Identifies locally the corresponding instance.

property original_publication_virtual_guid

Original publication virtual GUID.

If the Publisher's access_scope is GROUP, this field contains the Publisher virtual GUID that uniquely identifies the DataWriter group.

property original_publication_virtual_sample_identity

Retrieves the information provided by original_publication_virtual_guid and original_publication_virtual_sequence_number combined in a SampleIdentity instance.

property original_publication_virtual_sequence_number

Original publication virtual sequence number.

If the Publisher's access_scope is GROUP, this field contains the Publisher virtual sequence number that uniquely identifies a DDS sample within the DataWriter group.

property publication_handle

Identifies locally the DataWriter that modified the instance.

property publication_sequence_number

Publication sequence number assigned when the DDS sample was written by the DataWriter.

property rank

Get the Rank of the sample.

property reception_sequence_number

Reception sequence number assigned when the DDS sample was committed by the DataReader.

property reception_timestamp

The timestamp when the sample was committed by a DataReader.

property related_original_publication_virtual_guid

The original publication virtual GUID of a related sample.

property related_original_publication_virtual_sample_identity

Retrieves the information provided by related_original_publication_virtual_guid and related_original_publication_virtual_sequence_number combined in a SampleIdentity instance.

property related_original_publication_virtual_sequence_number

The original publication virtual sequence number of a related sample.

property related_source_guid

The application logical data source that is related to the sample.

property related_subscription_guid

The related_reader_guid associated with the sample.

property source_guid

The application logical data source associated with the sample.

property source_timestamp

The DataWriter's write timestamp.

property state

Get the DataState of the sample.

property topic_query_guid

The GUID of the TopicQuery that is related to the sample.

property valid

Indicates whether the DataSample contains data or else it is only used to communicate a change in the InstanceState of the instance.

class rti.connextdds.**SampleLostState**

Bases: pybind11_object

LOST_BY_AVAILABILITY_WAITING_TIME = <rti.connextdds.SampleLostState object>

LOST_BY_DECODE_FAILURE = <rti.connextdds.SampleLostState object>

LOST_BY_DESERIALIZATION_FAILURE = <rti.connextdds.SampleLostState object>

LOST_BY_INCOMPLETE_COHERENT_SET = <rti.connextdds.SampleLostState object>

LOST_BY_INSTANCES_LIMIT = <rti.connextdds.SampleLostState object>

LOST_BY_LARGE_COHERENT_SET = <rti.connextdds.SampleLostState object>

LOST_BY_OUT_OF_MEMORY = <rti.connextdds.SampleLostState object>

LOST_BY_REMOTE_WRITERS_PER_INSTANCE_LIMIT = <rti.connextdds.SampleLostState object>

LOST_BY_REMOTE_WRITERS_PER_SAMPLE_LIMIT = <rti.connextdds.SampleLostState object>

LOST_BY_REMOTE_WRITERS_PER_VIRTUAL_QUEUE_LIMIT = <rti.connextdds.SampleLostState object>

LOST_BY_SAMPLES_LIMIT = <rti.connextdds.SampleLostState object>

LOST_BY_SAMPLES_PER_INSTANCE_LIMIT = <rti.connextdds.SampleLostState object>

LOST_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT = <rti.connextdds.SampleLostState object>

LOST_BY_UNKNOWN_INSTANCE = <rti.connextdds.SampleLostState object>

LOST_BY_VIRTUAL_WRITERS_LIMIT = <rti.connextdds.SampleLostState object>

LOST_BY_WRITER = <rti.connextdds.SampleLostState object>

NOT_LOST = <rti.connextdds.SampleLostState object>

__and__ (*self:* rti.connextdds.SampleLostState, *arg0:* rti.connextdds.SampleLostState) → *rti.connextdds.SampleLostState*

Bitwise logical AND of masks.

__bool__ (*self:* rti.connextdds.SampleLostState) → bool

Test if any bits are set.

__contains__ (*self:* rti.connextdds.SampleLostState, *arg0:* rti.connextdds.SampleLostState) → bool

__eq__ (*self:* rti.connextdds.SampleLostState, *arg0:* rti.connextdds.SampleLostState) → bool

Compare masks for equality.

__getitem__ (*self:* rti.connextdds.SampleLostState, *arg0: int*) → bool

Get individual mask bit.

__hash__ = None

__iand__ (*self:* rti.connextdds.SampleLostState, *arg0:* rti.connextdds.SampleLostState) → *rti.connextdds.SampleLostState*

Set mask to logical AND with another mask.

__ilshift__ (*self:* rti.connextdds.SampleLostState, *arg0: int*) → *rti.connextdds.SampleLostState*

Left shift bits in mask.

__init__ (*self:* rti.connextdds.SampleLostState) → None

Creates SampleLostState.NOT_LOST

__int__ (*self:* rti.connextdds.SampleLostState) → int

Convert mask to int.

__ior__ (*self:* rti.connextdds.SampleLostState, *arg0:* rti.connextdds.SampleLostState) → *rti.connextdds.SampleLostState*

Set mask to logical OR with another mask.

__irshift__ (*self:* rti.connextdds.SampleLostState, *arg0: int*) → *rti.connextdds.SampleLostState*

Right shift bits in mask.

__ixor__ (*self:* rti.connextdds.SampleLostState, *arg0:* rti.connextdds.SampleLostState) → *rti.connextdds.SampleLostState*

Set mask to logical XOR with another mask.

__lshift__ (*self:* rti.connextdds.SampleLostState, *arg0: int*) → *rti.connextdds.SampleLostState*

Left shift bits in mask.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.SampleLostState, *arg0:* rti.connextdds.SampleLostState) → bool

Compare masks for inequality.

__or__ (*self:* rti.connextdds.SampleLostState, *arg0:* rti.connextdds.SampleLostState) → *rti.connextdds.SampleLostState*

Bitwise logical OR of masks.

__rshift__ (*self:* rti.connextdds.SampleLostState, *arg0: int*) → *rti.connextdds.SampleLostState*

Right shift bits in mask.

__setitem__ (*self:* rti.connextdds.SampleLostState, *arg0: int*, *arg1: bool*) → None

Set individual mask bit

__str__ (*self:* rti.connextdds.SampleLostState) → str

__xor__ (*self:* rti.connextdds.SampleLostState, *arg0:* rti.connextdds.SampleLostState) → *rti.connextdds.SampleLostState*

Bitwise logical XOR of masks.

property count

Returns the number of bits set in the mask.

flip (**args*, ***kwargs*)

Overloaded function.

1. flip(self: rti.connextdds.SampleLostState) -> rti.connextdds.SampleLostState

Flip all bits in the mask.

2. flip(self: rti.connextdds.SampleLostState, pos: int) -> rti.connextdds.SampleLostState

Flip the mask bit at the specified position.

reset (**args*, ***kwargs*)

Overloaded function.

1. reset(self: rti.connextdds.SampleLostState) -> rti.connextdds.SampleLostState

Clear all bits in the mask.

2. reset(self: rti.connextdds.SampleLostState, pos: int) -> rti.connextdds.SampleLostState

Clear the mask bit at the specified position.

set (**args*, ***kwargs*)

Overloaded function.

1. set(self: rti.connextdds.SampleLostState) -> rti.connextdds.SampleLostState

Set all bits in the mask.

2. set(self: rti.connextdds.SampleLostState, pos: int, value: bool = True) -> rti.connextdds.SampleLostState

Set the mask bit at the specified position to the provided value (default: true).

property size

Returns the number of bits in the mask type.

test (*self:* rti.connextdds.SampleLostState, *pos: int*) → bool

Test whether the mask bit at position "pos" is set.

test_all (*self:* rti.connextdds.SampleLostState) → bool

Test if all bits are set.

test_any (*self:* rti.connextdds.SampleLostState) → bool

Test if any bits are set.

test_none (*self:* rti.connextdds.SampleLostState) → bool

Test if none of the bits are set.

class rti.connextdds.**SampleLostStatus**

Bases: pybind11_object

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property last_reason

The reason for the most recent sample loss.

property total_count

Total count of all samples lost across all instances of data published under the Topic.

property total_count_change

The incremental number of samples lost since the last time the listener was called or the status was read.

class rti.connextdds.**SampleRejectedState**

Bases: pybind11_object

NOT_REJECTED = <rti.connextdds.SampleRejectedState object>

REJECTED_BY_DECODE_FAILURE = <rti.connextdds.SampleRejectedState object>

REJECTED_BY_INSTANCES_LIMIT = <rti.connextdds.SampleRejectedState object>

REJECTED_BY_REMOTE_WRITERS_PER_VIRTUAL_QUEUE_LIMIT = <rti.connextdds.SampleRejectedState object>

REJECTED_BY_SAMPLES_LIMIT = <rti.connextdds.SampleRejectedState object>

REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT = <rti.connextdds.SampleRejectedState object>

REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT = <rti.connextdds.SampleRejectedState object>

__and__ (*self:* rti.connextdds.SampleRejectedState, *arg0:* rti.connextdds.SampleRejectedState) → *rti.connextdds.SampleRejectedState*

Bitwise logical AND of masks.

__bool__ (*self:* rti.connextdds.SampleRejectedState) → bool

Test if any bits are set.

__contains__ (*self:* rti.connextdds.SampleRejectedState, *arg0:* rti.connextdds.SampleRejectedState) → bool

__eq__ (*self:* rti.connextdds.SampleRejectedState, *arg0:* rti.connextdds.SampleRejectedState) → bool

Compare masks for equality.

__getitem__ (*self:* rti.connextdds.SampleRejectedState, *arg0: int*) → bool

Get individual mask bit.

__hash__ = None

__iand__ (*self:* rti.connextdds.SampleRejectedState, *arg0:* rti.connextdds.SampleRejectedState) → *rti.connextdds.SampleRejectedState*

Set mask to logical AND with another mask.

__ilshift__ (*self:* rti.connextdds.SampleRejectedState, *arg0: int*) → *rti.connextdds.SampleRejectedState*

Left shift bits in mask.

__init__ (*self:* rti.connextdds.SampleRejectedState) → None

Creates SampleRejectedState.NOT_REJECTED

__int__ (*self:* rti.connextdds.SampleRejectedState) → int

Convert mask to int.

__ior__ (*self:* rti.connextdds.SampleRejectedState, *arg0:* rti.connextdds.SampleRejectedState) → *rti.connextdds.SampleRejectedState*

Set mask to logical OR with another mask.

__irshift__ (*self:* rti.connextdds.SampleRejectedState, *arg0: int*) → *rti.connextdds.SampleRejectedState*

Right shift bits in mask.

__ixor__ (*self:* rti.connextdds.SampleRejectedState, *arg0:* rti.connextdds.SampleRejectedState) → *rti.connextdds.SampleRejectedState*

Set mask to logical XOR with another mask.

__lshift__ (*self:* rti.connextdds.SampleRejectedState, *arg0: int*) → *rti.connextdds.SampleRejectedState*

Left shift bits in mask.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.SampleRejectedState, *arg0:* rti.connextdds.SampleRejectedState) → bool

Compare masks for inequality.

__or__ (*self:* rti.connextdds.SampleRejectedState, *arg0:* rti.connextdds.SampleRejectedState) → *rti.connextdds.SampleRejectedState*

Bitwise logical OR of masks.

__rshift__ (*self:* rti.connextdds.SampleRejectedState, *arg0: int*) → *rti.connextdds.SampleRejectedState*

Right shift bits in mask.

__setitem__ (*self:* rti.connextdds.SampleRejectedState, *arg0: int*, *arg1: bool*) → None

Set individual mask bit

__str__ (*self:* rti.connextdds.SampleRejectedState) → str

__xor__ (*self:* rti.connextdds.SampleRejectedState, *arg0:* rti.connextdds.SampleRejectedState) → *rti.connextdds.SampleRejectedState*

Bitwise logical XOR of masks.

property count

Returns the number of bits set in the mask.

flip (**args*, ***kwargs*)

Overloaded function.

1. flip(self: rti.connextdds.SampleRejectedState) -> rti.connextdds.SampleRejectedState

Flip all bits in the mask.

2. flip(self: rti.connextdds.SampleRejectedState, pos: int) -> rti.connextdds.SampleRejectedState

Flip the mask bit at the specified position.

reset (**args*, ***kwargs*)

Overloaded function.

1. reset(self: rti.connextdds.SampleRejectedState) -> rti.connextdds.SampleRejectedState

Clear all bits in the mask.

2. reset(self: rti.connextdds.SampleRejectedState, pos: int) -> rti.connextdds.SampleRejectedState

Clear the mask bit at the specified position.

set (**args*, ***kwargs*)

Overloaded function.

1. set(self: rti.connextdds.SampleRejectedState) -> rti.connextdds.SampleRejectedState

Set all bits in the mask.

2. set(self: rti.connextdds.SampleRejectedState, pos: int, value: bool = True) -> rti.connextdds.SampleRejectedState

Set the mask bit at the specified position to the provided value (default: true).

property size

Returns the number of bits in the mask type.

test (*self:* rti.connextdds.SampleRejectedState, *pos: int*) → bool

Test whether the mask bit at position "pos" is set.

test_all (*self:* rti.connextdds.SampleRejectedState) → bool

Test if all bits are set.

test_any (*self:* rti.connextdds.SampleRejectedState) → bool

Test if any bits are set.

test_none (*self:* rti.connextdds.SampleRejectedState) → bool

Test if none of the bits are set.

class rti.connextdds.**SampleRejectedStatus**

Bases: pybind11_object

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property last_instance_handle

Handle for the instance of the sample that was most recently rejected.

property last_reason

Reason for the DataReader's most recent sample rejection.

property total_count

Total count of samples rejected by the DataReader.

property total_count_change

The delta number of samples rejected since the last time the listener fired or the status was read.

class rti.connextdds.**SampleState**

Bases: pybind11_object

ANY = <rti.connextdds.SampleState object>

NOT_READ = <rti.connextdds.SampleState object>

READ = <rti.connextdds.SampleState object>

__and__ (*self:* rti.connextdds.SampleState, *arg0:* rti.connextdds.SampleState) → *rti.connextdds.SampleState*

Bitwise logical AND of masks.

__bool__ (*self:* rti.connextdds.SampleState) → *rti.connextdds.SampleState*

Test if any bits are set.

__contains__ (*self:* rti.connextdds.SampleState, *arg0:* rti.connextdds.SampleState) → bool

__eq__ (*self:* rti.connextdds.SampleState, *arg0:* rti.connextdds.SampleState) → bool

Compare masks for equality.

__getitem__ (*self:* rti.connextdds.SampleState, *arg0: int*) → bool

Get individual mask bit.

__hash__ = None

__iand__ (*self:* rti.connextdds.SampleState, *arg0:* rti.connextdds.SampleState) → *rti.connextdds.SampleState*

Set mask to logical AND with another mask.

__ilshift__ (*self:* rti.connextdds.SampleState, *arg0: int*) → *rti.connextdds.SampleState*

Left shift bits in mask.

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SampleState) -> None

Create a SampleState with no bits set.

2. __init__(self: rti.connextdds.SampleState, value: int) -> None

Creates a mask from the bits in an integer.

__int__ (*self:* rti.connextdds.SampleState) → int

Convert mask to int.

__ior__ (*self:* rti.connextdds.SampleState, *arg0:* rti.connextdds.SampleState) → *rti.connextdds.SampleState*

Set mask to logical OR with another mask.

__irshift__ (*self:* rti.connextdds.SampleState, *arg0: int*) → *rti.connextdds.SampleState*

Right shift bits in mask.

__ixor__ (*self:* rti.connextdds.SampleState, *arg0:* rti.connextdds.SampleState) → *rti.connextdds.SampleState*

Set mask to logical XOR with another mask.

__lshift__ (*self:* rti.connextdds.SampleState, *arg0: int*) → *rti.connextdds.SampleState*

Left shift bits in mask.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.SampleState, *arg0:* rti.connextdds.SampleState) → bool

Compare masks for inequality.

__or__ (*self:* rti.connextdds.SampleState, *arg0:* rti.connextdds.SampleState) → *rti.connextdds.SampleState*

Bitwise logical OR of masks.

__rshift__ (*self:* rti.connextdds.SampleState, *arg0: int*) → *rti.connextdds.SampleState*

Right shift bits in mask.

__setitem__ (*self:* rti.connextdds.SampleState, *arg0: int*, *arg1: bool*) → None

Set individual mask bit

__str__ (*self:* rti.connextdds.SampleState) → str

__xor__ (*self:* rti.connextdds.SampleState, *arg0:* rti.connextdds.SampleState) → *rti.connextdds.SampleState*

Bitwise logical XOR of masks.

property count

Returns the number of bits set in the mask.

flip (**args*, ***kwargs*)

Overloaded function.

1. flip(self: rti.connextdds.SampleState) -> rti.connextdds.SampleState

Flip all bits in the mask.

2. flip(self: rti.connextdds.SampleState, pos: int) -> rti.connextdds.SampleState

Flip the mask bit at the specified position.

reset (**args*, ***kwargs*)

Overloaded function.

1. reset(self: rti.connextdds.SampleState) -> rti.connextdds.SampleState

Clear all bits in the mask.

2. reset(self: rti.connextdds.SampleState, pos: int) -> rti.connextdds.SampleState

Clear the mask bit at the specified position.

set (**args*, ***kwargs*)

Overloaded function.

1. set(self: rti.connextdds.SampleState) -> rti.connextdds.SampleState

Set all bits in the mask.

2. set(self: rti.connextdds.SampleState, pos: int, value: bool = True) -> rti.connextdds.SampleState

Set the mask bit at the specified position to the provided value (default: true).

property size

Returns the number of bits in the mask type.

test (*self:* rti.connextdds.SampleState, *pos: int*) → bool

Test whether the mask bit at position "pos" is set.

test_all (*self:* rti.connextdds.SampleState) → bool

Test if all bits are set.

test_any (*self:* rti.connextdds.SampleState) → bool

Test if any bits are set.

test_none (*self:* rti.connextdds.SampleState) → bool

Test if none of the bits are set.

class rti.connextdds.**SequenceNumber**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.SequenceNumber, *arg0:* rti.connextdds.SequenceNumber) → *rti.connextdds.SequenceNumber*

Add two SequenceNumbers.

__eq__ (*self:* rti.connextdds.SequenceNumber, *arg0:* rti.connextdds.SequenceNumber) → bool

Compare SequenceNumbers for equality.

__ge__ (*self:* rti.connextdds.SequenceNumber, *arg0:* rti.connextdds.SequenceNumber) → bool

Compare two SequenceNumbers for a greater-than-or-equal relationship.

__gt__ (*self:* rti.connextdds.SequenceNumber, *arg0:* rti.connextdds.SequenceNumber) → bool

Compare two SequenceNumbers for a greater-than relationship.

__hash__ = None

__iadd__ (*self:* rti.connextdds.SequenceNumber, *arg0:* rti.connextdds.SequenceNumber) → *rti.connextdds.SequenceNumber*

Add a SequenceNumber to another

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SequenceNumber) -> None

Create a default SequenceNumber, equal to SequenceNumber.unknown

2. __init__(self: rti.connextdds.SequenceNumber, value: int) -> None

Creates a SequenceNumber from an integer.

__int__ (*self:* rti.connextdds.SequenceNumber) → int

Convert SequenceNumber to integer.

__isub__ (*self:* rti.connextdds.SequenceNumber, *arg0:* rti.connextdds.SequenceNumber) → *rti.connextdds.SequenceNumber*

Subtract a SequenceNumber from another.

__le__ (*self:* rti.connextdds.SequenceNumber, *arg0:* rti.connextdds.SequenceNumber) → bool

Compare two SequenceNumbers for a less-than-or-equal relationship.

__lt__ (*self:* rti.connextdds.SequenceNumber, *arg0:* rti.connextdds.SequenceNumber) → bool

Compare two SequenceNumbers for a less-than relationship.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.SequenceNumber, *arg0:* rti.connextdds.SequenceNumber) → bool

Compare SequenceNumbers for inequality.

__repr__ (*self:* rti.connextdds.SequenceNumber) → str

Convert SequenceNumber to string.

__sub__ (*self:* rti.connextdds.SequenceNumber, *arg0:* rti.connextdds.SequenceNumber) → *rti.connextdds.SequenceNumber*

Subtract one SequenceNumber from another.

automatic = -1

maximum = 9223372036854775807

unknown = -1

property value

Get/set the SequenceNumber value.

zero = 0

class rti.connextdds.**SequenceType**

Bases: *UnidimensionalCollectionType*

__eq__ (*self:* rti.connextdds.SequenceType, *arg0:* rti.connextdds.SequenceType) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SequenceType, data_type: rti.connextdds.DynamicType) -> None

Creates an unbounded collection with an element type.

2. __init__(self: rti.connextdds.SequenceType, data_type: rti.connextdds.DynamicType, bounds: int) -> None

Creates a bounded collection with an element type.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.SequenceType, *arg0:* rti.connextdds.SequenceType) → bool

Test for inequality.

class rti.connextdds.**Service**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.Service, *arg0:* rti.connextdds.Service) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Service) -> None

Creates the default policy (no service).

2. __init__(self: rti.connextdds.Service, kind: rti.connextdds.ServiceKind) -> None

Creates an instance with the specified service kind.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.Service, *arg0:* rti.connextdds.Service) → bool

Test for inequality.

property kind

The service kind.

class rti.connextdds.**ServiceKind**

Bases: pybind11_object

DATABASE_INTEGRATION = <ServiceKind.DATABASE_INTEGRATION: 6>

NO_SERVICE = <ServiceKind.NO_SERVICE: 0>

PERSISTENCE = <ServiceKind.PERSISTENCE: 1>

QUEUING = <ServiceKind.QUEUING: 2>

RECORDING = <ServiceKind.RECORDING: 4>

REPLAY = <ServiceKind.REPLAY: 5>

ROUTING = <ServiceKind.ROUTING: 3>

class ServiceKind

Bases: pybind11_object

Members:

NO_SERVICE : There is no service associated to the Entity.

PERSISTENCE : The Entity is associated to RTI Persistence Service.

QUEUING : The Entity is associated to RTI Queuing Service.

ROUTING : The Entity is associated to RTI Routing Service.

RECORDING : The Entity is associated to RTI Recording Service.

REPLAY : The Entity is associated to RTI Replay Service.

DATABASE_INTEGRATION : The Entity is associated to RTI Database Integration Service.

WEB_INTEGRATION : The Entity is associated to RTI Web Integration Service.

DATABASE_INTEGRATION = <ServiceKind.DATABASE_INTEGRATION: 6>

NO_SERVICE = <ServiceKind.NO_SERVICE: 0>

PERSISTENCE = <ServiceKind.PERSISTENCE: 1>

QUEUING = <ServiceKind.QUEUING: 2>

RECORDING = <ServiceKind.RECORDING: 4>

REPLAY = <ServiceKind.REPLAY: 5>

ROUTING = <ServiceKind.ROUTING: 3>

WEB_INTEGRATION = <ServiceKind.WEB_INTEGRATION: 7>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.ServiceKind.ServiceKind) → int

__init__(*self:* rti.connextdds.ServiceKind.ServiceKind, *value: int*) → None

__int__(*self:* rti.connextdds.ServiceKind.ServiceKind) → int

__members__ = {'DATABASE_INTEGRATION': <ServiceKind.DATABASE_INTEGRATION: 6>, 'NO_SERVICE': <ServiceKind.NO_SERVICE: 0>, 'PERSISTENCE': <ServiceKind.PERSISTENCE: 1>, 'QUEUING': <ServiceKind.QUEUING: 2>, 'RECORDING': <ServiceKind.RECORDING: 4>, 'REPLAY': <ServiceKind.REPLAY: 5>, 'ROUTING': <ServiceKind.ROUTING: 3>, 'WEB_INTEGRATION': <ServiceKind.WEB_INTEGRATION: 7>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.ServiceKind.ServiceKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

WEB_INTEGRATION = <ServiceKind.WEB_INTEGRATION: 7>

__eq__ (*self:* rti.connextdds.ServiceKind, *arg0:* rti.connextdds.ServiceKind) → bool

Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.ServiceKind, *arg0:* rti.connextdds.ServiceKind) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.ServiceKind, *arg0:* rti.connextdds.ServiceKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ServiceKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.ServiceKind, arg0: rti.connextdds.ServiceKind.ServiceKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.ServiceKind) → *rti.connextdds.ServiceKind.ServiceKind*

Int conversion.

__le__ (*self:* rti.connextdds.ServiceKind, *arg0:* rti.connextdds.ServiceKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.ServiceKind, *arg0:* rti.connextdds.ServiceKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ServiceKind, *arg0:* rti.connextdds.ServiceKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.ServiceKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**ServiceRequest**

Bases: pybind11_object

class ContentFilter

Bases: *ContentFilterBase*

__init__ (*self:* rti.connextdds.ServiceRequest.ContentFilter) → None

__module__ = 'rti.connextdds'

compile (*self:* rti.connextdds.ServiceRequest.ContentFilter, *expression: str*, *parameters:* rti.connextdds.StringSeq, *type_code: Optional[*rti.connextdds.DynamicType*]*, *type_class_name: str*, *old_compile_data: Optional[object]*) → Optional[object]

Compile an instance of the content filter according to the filter expression and parameters of the given data type.

evaluate (*self:* rti.connextdds.ServiceRequest.ContentFilter, *compile_data: Optional[object]*, *sample:* rti.connextdds.ServiceRequest, *meta_data:* rti.connextdds.FilterSampleInfo) → bool

Evaluate whether the sample is passing the filter or not according to the sample content.

finalize (*self:* rti.connextdds.ServiceRequest.ContentFilter, *compile_data: Optional[object]*) → None

A previously compiled instance of the content filter is no longer in use and resources can now be cleaned up.

class ContentFilteredTopic

Bases: *ITopicDescription*, *IAnyTopic*

__eq__ (*self:* rti.connextdds.ServiceRequest.ContentFilteredTopic, *arg0:* rti.connextdds.ServiceRequest.ContentFilteredTopic) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ServiceRequest.ContentFilteredTopic, topic: rti.connextdds.ServiceRequest.Topic, name: str, contentfilter: rti.connextdds.Filter) -> None

Create a ContentFilteredTopic with a name and Filter.

2. __init__(self: rti.connextdds.ServiceRequest.ContentFilteredTopic, topic_description: rti.connextdds.ServiceRequest.ITopicDescription) -> None

Cast a TopicDescription to a ContentFilteredTopic.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ServiceRequest.ContentFilteredTopic, *arg0:* rti.connextdds.ServiceRequest.ContentFilteredTopic) → bool

Test for inequality.

append_to_expression_parameter (*self:* rti.connextdds.ServiceRequest.ContentFilteredTopic, *index: int*, *extension: str*) → None

Append the extension to the end of parameter at the provided index, separated by a comma.

property filter_expression

Get the filter expression

property filter_parameters

Get/set the filter parameters.

static find(*participant:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.ServiceRequest.ContentFilteredTopic*]

Look up a ContentFilteredTopic by its name in the DomainParticipant.

remove_from_expression_parameter(*self:* rti.connextdds.ServiceRequest.ContentFilteredTopic, *index: int*, *remove_term: str*) → None

Removes the specified term from the parameter at the provided index.

set_filter(*self:* rti.connextdds.ServiceRequest.ContentFilteredTopic, *arg0:* rti.connextdds.Filter) → None

Set the filter.

property topic

Get the underlying Topic.

class ContentFilteredTopicSeq

Bases: `pybind11_object`

__add__(*self:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, *arg0:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq) → *rti.connextdds.ServiceRequest.ContentFilteredTopicSeq*

__bool__(*self:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, *x:* rti.connextdds.ServiceRequest.ContentFilteredTopic) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, *arg0:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, s: slice) -> rti.connextdds.ServiceRequest.ContentFilteredTopicSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, arg0: int) -> rti.connextdds.ServiceRequest.ContentFilteredTopic

__hash__ = None

__iadd__(*self:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, *arg0:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq) → *rti.connextdds.ServiceRequest.ContentFilteredTopicSeq*

__imul__(*self:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, *arg0: int*) → *rti.connextdds.ServiceRequest.ContentFilteredTopicSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ServiceRequest.ContentFilteredTopicSeq) -> None
2. __init__(self: rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, arg0: rti.connextdds.ServiceRequest.ContentFilteredTopicSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq) → Iterator

__len__(*self:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, *arg0: int*) → *rti.connextdds.ServiceRequest.ContentFilteredTopicSeq*

__ne__(*self:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, *arg0:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq) → bool

__rmul__(*self:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, *arg0: int*) → *rti.connextdds.ServiceRequest.ContentFilteredTopicSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, arg0: int, arg1: rti.connextdds.ServiceRequest.ContentFilteredTopic) -> None
2. __setitem__(self: rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, arg0: slice, arg1: rti.connextdds.ServiceRequest.ContentFilteredTopicSeq) -> None

Assign list elements using a slice object

append(*self:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, *x:* rti.connextdds.ServiceRequest.ContentFilteredTopic) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq) → None

Clear the contents

count (*self:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, *x:* rti.connextdds.ServiceRequest.ContentFilteredTopic) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, L: rti.connextdds.ServiceRequest.ContentFilteredTopicSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, *i: int*, *x:* rti.connextdds.ServiceRequest.ContentFilteredTopic) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.ServiceRequest.ContentFilteredTopicSeq) -> rti.connextdds.ServiceRequest.ContentFilteredTopic

Remove and return the last item

2. pop(self: rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, i: int) -> rti.connextdds.ServiceRequest.ContentFilteredTopic

Remove and return the item at index i

remove (*self:* rti.connextdds.ServiceRequest.ContentFilteredTopicSeq, *x:* rti.connextdds.ServiceRequest.ContentFilteredTopic) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class DataReader

Bases: *IDataReader*

class Selector

Bases: pybind11_object

__init__ (*self:* rti.connextdds.ServiceRequest.DataReader.Selector, *datareader:* rti.connextdds.ServiceRequest.DataReader) → None

Create a Selector for a DataReader to read/take based on selected conditions

__module__ = 'rti.connextdds'

condition (*self:* rti.connextdds.ServiceRequest.DataReader.Selector, *condition:* rti.connextdds.IReadCondition) → *rti.connextdds.ServiceRequest.DataReader.Selector*

Select samples based on a ReadCondition.

content (*self:* rti.connextdds.ServiceRequest.DataReader.Selector, *query:* rti.connextdds.Query) → *rti.connextdds.ServiceRequest.DataReader.Selector*

Select samples based on a Query.

instance (*self:* rti.connextdds.ServiceRequest.DataReader.Selector, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ServiceRequest.DataReader.Selector*

Select a specific instance to read/take.

max_samples (*self:* rti.connextdds.ServiceRequest.DataReader.Selector, *max: int*) → *rti.connextdds.ServiceRequest.DataReader.Selector*

Limit the number of samples read/taken by the Select.

next_instance (*self:* rti.connextdds.ServiceRequest.DataReader.Selector, *previous:* rti.connextdds.InstanceHandle) → *rti.connextdds.ServiceRequest.DataReader.Selector*

Select the instance after the specified instance to read/take.

read (*self:* rti.connextdds.ServiceRequest.DataReader.Selector) → list

Read copies of available samples (data and info) based on the Selector settings.

read_data (*self:* rti.connextdds.ServiceRequest.DataReader.Selector) → list

Read copies of available valid data based on the Selector settings.

read_loaned (*self:* rti.connextdds.ServiceRequest.DataReader.Selector) → *rti.connextdds.ServiceRequest.LoanedSamples*

Take available samples (data and info) based on the Selector settings and return them in a loaned container.

state (*self:* rti.connextdds.ServiceRequest.DataReader.Selector, *state:* rti.connextdds.DataState) → *rti.connextdds.ServiceRequest.DataReader.Selector*

Select samples with a specified data state.

take (*self:* rti.connextdds.ServiceRequest.DataReader.Selector) → list

Take copies of available samples (data and info) based on the Selector settings.

take_data (*self:* rti.connextdds.ServiceRequest.DataReader.Selector) → list

Take copies of available valid data based on the Selector settings.

take_loaned (*self:* rti.connextdds.ServiceRequest.DataReader.Selector) → *rti.connextdds.ServiceRequest.LoanedSamples*

Read available samples (data and info) based on the Selector settings and return them in a loaned container.

__enter__ (*self:* rti.connextdds.ServiceRequest.DataReader) → *rti.connextdds.ServiceRequest.DataReader*

Enter a context for this DataReader, to be cleaned up on exiting context

__eq__ (*self:* rti.connextdds.ServiceRequest.DataReader, *arg0:* rti.connextdds.ServiceRequest.DataReader) → bool

Test for equality.

__exit__ (*self:* rti.connextdds.ServiceRequest.DataReader, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for this DataReader, cleaning up resources.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ServiceRequest.DataReader, topic: rti.connextdds.ServiceRequest.Topic) -> None

Create a DataReader in the implicit subscriber with default QoS.

2. __init__(self: rti.connextdds.ServiceRequest.DataReader, topic: rti.connextdds.ServiceRequest.Topic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.ServiceRequest.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader in the implicit subscriber with specific QoS and a listener.

3. __init__(self: rti.connextdds.ServiceRequest.DataReader, cft: rti.connextdds.ServiceRequest.ContentFilteredTopic) -> None

Create a DataReader with a ContentFilteredTopic in the implicit subscriber with default QoS.

4. __init__(self: rti.connextdds.ServiceRequest.DataReader, cft: rti.connextdds.ServiceRequest.ContentFilteredTopic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.ServiceRequest.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader with a ContentFilteredTopic in the implicit subscriber with specific QoS.

5. __init__(self: rti.connextdds.ServiceRequest.DataReader, subscriber: rti.connextdds.Subscriber, topic: rti.connextdds.ServiceRequest.Topic) -> None

Create a DataReader.

6. __init__(self: rti.connextdds.ServiceRequest.DataReader, subscriber: rti.connextdds.Subscriber, topic: rti.connextdds.ServiceRequest.Topic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.ServiceRequest.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader in a subscriber with specific QoS and a listener.

7. __init__(self: rti.connextdds.ServiceRequest.DataReader, subscriber: rti.connextdds.Subscriber, cft: rti.connextdds.ServiceRequest.ContentFilteredTopic) -> None

Create a DataReader with a ContentFilteredTopic in a subscriber with default QoS.

8. __init__(self: rti.connextdds.ServiceRequest.DataReader, subscriber: rti.connextdds.Subscriber, cft: rti.connextdds.ServiceRequest.ContentFilteredTopic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.ServiceRequest.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader with a ContentFilteredTopic in a subscriber with specific QoS.

9. __init__(self: rti.connextdds.ServiceRequest.DataReader, reader: rti.connextdds.IAnyDataReader) -> None

Get a typed DataReader from an AnyDataReader.

10. __init__(self: rti.connextdds.ServiceRequest.DataReader, entity: rti.connextdds.IEntity) -> None

Get a typed DataReader from an Entity.

__lshift__(*self:* rti.connextdds.ServiceRequest.DataReader, *arg0:* rti.connextdds.DataReaderQos) → *rti.connextdds.ServiceRequest.DataReader*

Set the DataReaderQos for this DataReader.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.ServiceRequest.DataReader, *arg0:* rti.connextdds.ServiceRequest.DataReader) → bool

Test for inequality.

__rshift__(*self:* rti.connextdds.ServiceRequest.DataReader, *arg0:* rti.connextdds.DataReaderQos) → *rti.connextdds.ServiceRequest.DataReader*

Get the DataReaderQos from this DataReader

acknowledge_all(**args*, ***kwargs*)

Overloaded function.

1. acknowledge_all(self: rti.connextdds.ServiceRequest.DataReader) -> None

Acknowledge all previously accessed samples.

2. acknowledge_all(self: rti.connextdds.ServiceRequest.DataReader, arg0: rti.connextdds.AckResponseData) -> None

Acknowledge all previously accessed samples.

acknowledge_sample(**args*, ***kwargs*)

Overloaded function.

1. acknowledge_sample(self: rti.connextdds.ServiceRequest.DataReader, sample_info: rti.connextdds.SampleInfo) -> None

Acknowledge a single sample.

2. acknowledge_sample(self: rti.connextdds.ServiceRequest.DataReader, sample_info: rti.connextdds.SampleInfo, ack_response_data: rti.connextdds.AckResponseData) -> None

Acknowledge a single sample with ack response data.

close(*self:* rti.connextdds.ServiceRequest.DataReader) → None

Close this DataReader.

property datareader_cache_status

Get the DataReaderCacheStatus for the DataReader.

property datareader_protocol_status

Get the DataReaderProtocolStatus for the DataReader.

property default_filter_state

Returns the filter state for the read/take operations.

static find_all_by_topic(*subscriber:* rti.connextdds.Subscriber, *topic_name: str*) → *rti.connextdds.ServiceRequest.DataReaderSeq*

Retrieve all DataReaders for the given topic name in the subscriber.

static find_by_name(**args*, ***kwargs*)

Overloaded function.

1. find_by_name(participant: rti.connextdds.DomainParticipant, name: str) -> Optional[rti.connextdds.ServiceRequest.DataReader]

Find DataReader in DomainParticipant with the DataReader's name, returning the first found.

2. find_by_name(subscriber: rti.connextdds.Subscriber, name: str) -> Optional[rti.connextdds.ServiceRequest.DataReader]

Find DataReader in Subscriber with the DataReader's name, returning the first found.

static find_by_topic(*subscriber:* rti.connextdds.Subscriber, *name: str*) → Optional[*rti.connextdds.ServiceRequest.DataReader*]

Find DataReader in Subscriber with a topic name, returning the first found.

is_matched_publication_alive(*self:* rti.connextdds.ServiceRequest.DataReader, *arg0:* rti.connextdds.InstanceHandle) → bool

Check if a matched publication is alive.

key_value(*self:* rti.connextdds.ServiceRequest.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ServiceRequest*

Retrieve the instance key that corresponds to an instance handle.

property listener

Gets or sets the listener with StatusMask.ALL

property liveliness_changed_status

Get the LivelinessChangedStatus for this DataReader.

lookup_instance(*self:* rti.connextdds.ServiceRequest.DataReader, *key_holder:* rti.connextdds.ServiceRequest) → *rti.connextdds.InstanceHandle*

Retrieve the instance handle that corresponds to an instance key_holder

matched_publication_data(*self:* rti.connextdds.ServiceRequest.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.PublicationBuiltinTopicData*

Get the PublicationBuiltinTopicData for a publication matched to this DataReader.

matched_publication_datareader_protocol_status(*self:* rti.connextdds.ServiceRequest.DataReader, *publication_handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.DataReaderProtocolStatus*

Get the DataReaderProtocolStatus for the DataReader based on the matched publication handle.

matched_publication_participant_data(*self:* rti.connextdds.ServiceRequest.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ParticipantBuiltinTopicData*

Get the ParticipantBuiltinTopicData for a publication matched to this DataReader.

property matched_publications

Get a copy of the list of the currently matched publication handles.

property qos

The DataReaderQos for this DataReader.

This property's getter returns a deep copy.

read(*self:* rti.connextdds.ServiceRequest.DataReader) → list

Read copies of all available samples (data and info).

read_data(*self:* rti.connextdds.ServiceRequest.DataReader) → list

Read copies of all available valid data.

read_loaned(*self:* rti.connextdds.ServiceRequest.DataReader) →
rti.connextdds.ServiceRequest.LoanedSamples

Read all available samples (data and info) and return them in a loaned container.

property requested_deadline_missed_status

Get the RequestedDeadlineMissed status for the DataReader

property requested_incompatible_qos_status

Get the RequestedIncompatibleQosStatus for the DataReader.

property sample_lost_status

Get the SampleLostStatus for the DataReader.

property sample_rejected_status

Get the SampleRejectedStatus for the DataReader.

select(*self:* rti.connextdds.ServiceRequest.DataReader) →
dds::sub::DataReader<rti::topic::ServiceRequest, rti::sub::DataReaderImpl>::Selector

Get a Selector to perform complex data selections, such as per-instance selection, content, and status filtering.

set_listener(*self:* rti.connextdds.ServiceRequest.DataReader, *listener:*
rti.connextdds.ServiceRequest.DataReaderListener, *event_mask:*
rti.connextdds.StatusMask) → None

Set the listener and associated event mask.

property subscriber

Returns the parent Subscriber of the DataReader.

property subscription_matched_status

Get the SubscriptionMatchedStatus for the DataReader.

take(*self:* rti.connextdds.ServiceRequest.DataReader) → list

Take copies of all available samples (data and info).

take_data(*self:* rti.connextdds.ServiceRequest.DataReader) → list

Take copies of all available valid data.

take_loaned(*self:* rti.connextdds.ServiceRequest.DataReader) → *rti.connextdds.ServiceRequest.LoanedSamples*

Take all available samples (data and info) and return them in a loaned container.

property topic_description

Returns the TopicDescription associated with the DataReader.

property topic_name

Get the topic name associated with this DataReader.

property type_name

Get the type name associated with this DataReader.

wait_for_historical_data(*self:* rti.connextdds.ServiceRequest.DataReader, *max_wait:* rti.connextdds.Duration) → None

Waits until all "historical" data is received for DataReaders that have a non-VOLATILE Durability Qos kind.

wait_for_historical_data_async(*self:* rti.connextdds.ServiceRequest.DataReader, *max_wait:* rti.connextdds.Duration) → object

Waits until all "historical" data is received for DataReaders that have a non-VOLATILE Durability Qos kind. This call is awaitable and only for use with asyncio.

class DataReaderListener

Bases: `pybind11_object`

__init__(*self:* rti.connextdds.ServiceRequest.DataReaderListener) → None

__module__ = 'rti.connextdds'

on_data_available(*self:* rti.connextdds.ServiceRequest.DataReaderListener, *arg0:* rti.connextdds.ServiceRequest.DataReader) → None

Data available callback.

on_liveliness_changed(*self:* rti.connextdds.ServiceRequest.DataReaderListener, *arg0:* rti.connextdds.ServiceRequest.DataReader, *arg1:* rti.connextdds.LivelinessChangedStatus) → None

Liveliness changed callback.

on_requested_deadline_missed(*self:* rti.connextdds.ServiceRequest.DataReaderListener, *arg0:* rti.connextdds.ServiceRequest.DataReader, *arg1:* rti.connextdds.RequestedDeadlineMissedStatus) → None

Requested deadline missed callback.

on_requested_incompatible_qos(*self:* rti.connextdds.ServiceRequest.DataReaderListener, *arg0:* rti.connextdds.ServiceRequest.DataReader, *arg1:* rti.connextdds.RequestedIncompatibleQosStatus) → None

Requested incompatible QoS callback.

on_sample_lost(*self:* rti.connextdds.ServiceRequest.DataReaderListener, *arg0:* rti.connextdds.ServiceRequest.DataReader, *arg1:* rti.connextdds.SampleLostStatus) → None

Sample lost callback.

on_sample_rejected(*self:* rti.connextdds.ServiceRequest.DataReaderListener, *arg0:* rti.connextdds.ServiceRequest.DataReader, *arg1:* rti.connextdds.SampleRejectedStatus) → None

Sample rejected callback.

on_subscription_matched(*self:* rti.connextdds.ServiceRequest.DataReaderListener, *arg0:* rti.connextdds.ServiceRequest.DataReader, *arg1:* rti.connextdds.SubscriptionMatchedStatus) → None

Subscription matched callback.

class DataReaderSeq

Bases: `pybind11_object`

__add__(*self:* rti.connextdds.ServiceRequest.DataReaderSeq, *arg0:* rti.connextdds.ServiceRequest.DataReaderSeq) → *rti.connextdds.ServiceRequest.DataReaderSeq*

__bool__(*self:* rti.connextdds.ServiceRequest.DataReaderSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.ServiceRequest.DataReaderSeq, *x:* rti.connextdds.ServiceRequest.DataReader) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.ServiceRequest.DataReaderSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.ServiceRequest.DataReaderSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.ServiceRequest.DataReaderSeq, *arg0:* rti.connextdds.ServiceRequest.DataReaderSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.ServiceRequest.DataReaderSeq, s: slice) -> rti.connextdds.ServiceRequest.DataReaderSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.ServiceRequest.DataReaderSeq, arg0: int) -> rti.connextdds.ServiceRequest.DataReader

__hash__ = None

__iadd__(*self:* rti.connextdds.ServiceRequest.DataReaderSeq, *arg0:* rti.connextdds.ServiceRequest.DataReaderSeq) → *rti.connextdds.ServiceRequest.DataReaderSeq*

__imul__(*self:* rti.connextdds.ServiceRequest.DataReaderSeq, *arg0: int*) → *rti.connextdds.ServiceRequest.DataReaderSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ServiceRequest.DataReaderSeq) -> None
2. __init__(self: rti.connextdds.ServiceRequest.DataReaderSeq, arg0: rti.connextdds.ServiceRequest.DataReaderSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.ServiceRequest.DataReaderSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.ServiceRequest.DataReaderSeq) → Iterator

__len__(*self:* rti.connextdds.ServiceRequest.DataReaderSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.ServiceRequest.DataReaderSeq, *arg0: int*) → *rti.connextdds.ServiceRequest.DataReaderSeq*

__ne__(*self:* rti.connextdds.ServiceRequest.DataReaderSeq, *arg0:* rti.connextdds.ServiceRequest.DataReaderSeq) → bool

__rmul__(*self:* rti.connextdds.ServiceRequest.DataReaderSeq, *arg0: int*) → *rti.connextdds.ServiceRequest.DataReaderSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.ServiceRequest.DataReaderSeq, arg0: int, arg1: rti.connextdds.ServiceRequest.DataReader) -> None
2. __setitem__(self: rti.connextdds.ServiceRequest.DataReaderSeq, arg0: slice, arg1: rti.connextdds.ServiceRequest.DataReaderSeq) -> None

Assign list elements using a slice object

append(*self:* rti.connextdds.ServiceRequest.DataReaderSeq, *x:* rti.connextdds.ServiceRequest.DataReader) → None

Add an item to the end of the list

clear(*self:* rti.connextdds.ServiceRequest.DataReaderSeq) → None

Clear the contents

count(*self:* rti.connextdds.ServiceRequest.DataReaderSeq, *x:* rti.connextdds.ServiceRequest.DataReader) → int

Return the number of times x appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.ServiceRequest.DataReaderSeq, L: rti.connextdds.ServiceRequest.DataReaderSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.ServiceRequest.DataReaderSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert(*self:* rti.connextdds.ServiceRequest.DataReaderSeq, *i: int*, *x:* rti.connextdds.ServiceRequest.DataReader) → None

Insert an item at a given position.

pop(**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.ServiceRequest.DataReaderSeq) -> rti.connextdds.ServiceRequest.DataReader

Remove and return the last item

2. pop(self: rti.connextdds.ServiceRequest.DataReaderSeq, i: int) -> rti.connextdds.ServiceRequest.DataReader

Remove and return the item at index `i`

remove(*self:* rti.connextdds.ServiceRequest.DataReaderSeq, *x:* rti.connextdds.ServiceRequest.DataReader) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class DataWriter

Bases: *IEntity*, *IAnyDataWriter*

__enter__(*self:* rti.connextdds.ServiceRequest.DataWriter) → *rti.connextdds.ServiceRequest.DataWriter*

Enter a context for this DataWriter, to be cleaned up on exiting context

__eq__(*self:* rti.connextdds.ServiceRequest.DataWriter, *arg0:* rti.connextdds.ServiceRequest.DataWriter) → bool

Test for equality.

__exit__(*self:* rti.connextdds.ServiceRequest.DataWriter, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for this DataWriter, cleaning up resources.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ServiceRequest.DataWriter, topic: rti.connextdds.ServiceRequest.Topic) -> None

Creates a DataWriter in the implicit publisher with default QoS.

2. __init__(self: rti.connextdds.ServiceRequest.DataWriter, topic: rti.connextdds.ServiceRequest.Topic, qos: rti.connextdds.DataWriterQos, listener: rti.connextdds.Ser-

viceRequest.DataWriterListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Creates a DataWriter in the implicit publisher with specific QoS and optionally a listener.

3. __init__(self: rti.connextdds.ServiceRequest.DataWriter, pub: rti.connextdds.Publisher, topic: rti.connextdds.ServiceRequest.Topic) -> None

Creates a DataWriter in a publisher with default QoS.

4. __init__(self: rti.connextdds.ServiceRequest.DataWriter, pub: rti.connextdds.Publisher, topic: rti.connextdds.ServiceRequest.Topic, qos: rti.connextdds.DataWriterQos, listener: rti.connextdds.ServiceRequest.DataWriterListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Creates a DataWriter in a publisher with specific QoS and optionally a listener.

5. __init__(self: rti.connextdds.ServiceRequest.DataWriter, writer: rti.connextdds.IAnyDataWriter) -> None

Create a typed DataWriter from an AnyDataWriter.

6. __init__(self: rti.connextdds.ServiceRequest.DataWriter, entity: rti.connextdds.IEntity) -> None

Create a typed DataWriter from an Entity.

__lshift__(**args*, ***kwargs*)

Overloaded function.

1. __lshift__(self: rti.connextdds.ServiceRequest.DataWriter, arg0: rti.connextdds.DataWriterQos) -> rti.connextdds.ServiceRequest.DataWriter

Sets the DataWriterQos.

2. __lshift__(self: rti.connextdds.ServiceRequest.DataWriter, arg0: Tuple[rti.connextdds.ServiceRequest, rti.connextdds.Time]) -> rti.connextdds.ServiceRequest.DataWriter

Writes a paired sample with a timestamp.

3. __lshift__(self: rti.connextdds.ServiceRequest.DataWriter, arg0: Tuple[rti.connextdds.ServiceRequest, rti.connextdds.InstanceHandle]) -> rti.connextdds.ServiceRequest.DataWriter

Writes a paired sample with an instance handle.

4. __lshift__(self: rti.connextdds.ServiceRequest.DataWriter, arg0: rti.connextdds.ServiceRequestTimestampedSeq) -> rti.connextdds.ServiceRequest.DataWriter

Writes a sequence of pairs of samples with timestamps.

5. __lshift__(self: rti.connextdds.ServiceRequest.DataWriter, arg0: rti.connextdds.ServiceRequestSeq) -> rti.connextdds.ServiceRequest.DataWriter

Writes a sequence of samples.

6. __lshift__(self: rti.connextdds.ServiceRequest.DataWriter, arg0: rti.connextdds.ServiceRequest) -> rti.connextdds.ServiceRequest.DataWriter

Writes a sample.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.ServiceRequest.DataWriter, *arg0:* rti.connextdds.ServiceRequest.DataWriter) → bool

Test for inequality.

__rshift__(*self:* rti.connextdds.ServiceRequest.DataWriter, *arg0:* rti.connextdds.DataWriterQos) → *rti.connextdds.ServiceRequest.DataWriter*

Get the DataWriterQos.

assert_liveliness (*self:* rti.connextdds.ServiceRequest.DataWriter) → None

Manually asserts the liveliness of the DataWriter.

close (*self:* rti.connextdds.ServiceRequest.DataWriter) → None

Close this DataWriter.

create_data (*self:* rti.connextdds.ServiceRequest.DataWriter) → *rti.connextdds.ServiceRequest*

Create data of the writer's associated type and initialize it.

property datawriter_cache_status

Get a copy of the cache status for this writer.

property datawriter_protocol_status

Get a copy of the protocol status for this writer.

dispose_instance (**args*, ***kwargs*)

Overloaded function.

1. dispose_instance(self: rti.connextdds.ServiceRequest.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.ServiceRequest.DataWriter

Dispose an instance.

2. dispose_instance(self: rti.connextdds.ServiceRequest.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> rti.connextdds.ServiceRequest.DataWriter

Dispose an instance with a timestamp.

3. dispose_instance(self: rti.connextdds.ServiceRequest.DataWriter, params: rti.connextdds.WriteParams) -> None

Dispose an instance with params.

dispose_instance_async (**args*, ***kwargs*)

Overloaded function.

1. dispose_instance_async(self: rti.connextdds.ServiceRequest.DataWriter, handle: rti.connextdds.InstanceHandle) -> object

Dispose an instance.

2. dispose_instance_async(self: rti.connextdds.ServiceRequest.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> object

Dispose an instance with a timestamp.

3. dispose_instance_async(self: rti.connextdds.ServiceRequest.DataWriter, key_holder: rti.connextdds.ServiceRequest) -> object

Dispose the instance associated with key_holder.

4. dispose_instance_async(self: rti.connextdds.ServiceRequest.DataWriter, key_holder: rti.connextdds.ServiceRequest, timestamp: rti.connextdds.Time) -> object

Dispose the instance associated with key_holder using a timestamp

5. dispose_instance_async(self: rti.connextdds.ServiceRequest.DataWriter, params: rti.connextdds.WriteParams) -> object

Dispose an instance with params.

static find_all_by_topic(*publisher:* rti.connextdds.Publisher, *topic_name: str*) → *rti.connextdds.ServiceRequest.DataWriterSeq*

Retrieve all DataWriters for the given topic name in the publisher.

static find_by_name(**args*, ***kwargs*)

Overloaded function.

1. find_by_name(participant: rti.connextdds.DomainParticipant, name: str) -> Optional[rti.connextdds.ServiceRequest.DataWriter]

Find DataWriter in DomainParticipant with the provided name, returning the first found.

2. find_by_name(publisher: rti.connextdds.Publisher, name: str) -> Optional[rti.connextdds.ServiceRequest.DataWriter]

Find DataWriter in Publisher with the DataReader's name, returning the first found.

static find_by_topic(*publisher:* rti.connextdds.Publisher, *name: str*) → Optional[*rti.connextdds.ServiceRequest.DataWriter*]

Find DataWriter in publisher with a topic name, returning the first found.

flush(*self:* rti.connextdds.ServiceRequest.DataWriter) → None

Flushes the batch in progress in the context of thecalling thread.

is_matched_subscription_active(*self:* rti.connextdds.ServiceRequest.DataWriter, *arg0:* rti.connextdds.InstanceHandle) → bool

A boolean indicating whether or not the matched subscription is active.

is_sample_app_acknowledged(*self:* rti.connextdds.ServiceRequest.DataWriter, *sample_id:* rti.connextdds.SampleIdentity) → bool

Indicates if a sample is considered application-acknowledged.

key_value(*self:* rti.connextdds.ServiceRequest.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ServiceRequest*

Retrieve the instance key that corresponds to an instance handle.

property listener

Get the listener associated with the DataWriter or set the listener.

property liveliness_lost_status

Get a copy of the LivelinessLostStatus.

lookup_instance(*self:* rti.connextdds.ServiceRequest.DataWriter, *key_holder:* rti.connextdds.ServiceRequest) → *rti.connextdds.InstanceHandle*

Retrieve the instance handle that corresponds to an instance key_holder

matched_subscription_data(*self:* rti.connextdds.ServiceRequest.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.SubscriptionBuiltinTopicData*

Get the SubscriptionBuiltinTopicData for a subscription matched to this DataWriter.

matched_subscription_datawriter_protocol_status(**args*, ***kwargs*)

Overloaded function.

1. matched_subscription_datawriter_protocol_status(self: rti.connextdds.ServiceRequest.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.DataWriterProtocolStatus

Get a copy of the protocol status for this writer per a matched subscription handle.

2. matched_subscription_datawriter_protocol_status(self: rti.connextdds.ServiceRequest.DataWriter, locator: rti.connextdds.Locator) -> rti.connextdds.DataWriterProtocolStatus

Get a copy of the protocol status for this writer per a matched subscription locator.

matched_subscription_participant_data(*self:* rti.connextdds.ServiceRequest.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ParticipantBuiltinTopicData*

Get the ParticipantBuiltinTopicData for a subscription matched to this DataWriter.

property matched_subscriptions

Get a copy of the list of the currently matched subscription handles.

property matched_subscriptions_locators

The locators used to communicate with matched DataReaders.

property offered_deadline_missed_status

Get a copy of the OfferedDeadlineMissedStatus.

property offered_incompatible_qos_status

Get a copy of the OfferedIncompatibleQosStatus

property publication_matched_status

Get a copy of the PublicationMatchedStatus

property publisher

Get the Publisher that owns this DataWriter.

property qos

The DataWriterQos for this DataWriter.This property's getter returns a deep copy.

register_instance(**args*, ***kwargs*)

Overloaded function.

1. register_instance(self: rti.connextdds.ServiceRequest.DataWriter, key_holder: rti.connextdds.ServiceRequest) -> rti.connextdds.InstanceHandle

Informs RTI Connext that the application will be modifying a particular instance.

2. register_instance(self: rti.connextdds.ServiceRequest.DataWriter, key_holder: rti.connextdds.ServiceRequest, timestamp: rti.connextdds.Time) -> rti.connextdds.InstanceHandle

Informs RTI Connext that the application will be modifying a particular instance and specified the timestamp.

3. register_instance(self: rti.connextdds.ServiceRequest.DataWriter, key_holder: rti.connextdds.ServiceRequest, params: rti.connextdds.WriteParams) -> rti.connextdds.InstanceHandle

Registers instance with parameters.

property reliable_reader_activity_changed_status

Get a copy of the reliable reader activity changed status for this writer.

property reliable_writer_cache_changed_status

Get a copy of the reliable cache status for this writer.

property service_request_accepted_status

Get a copy of the service request accepted status for this writer.

set_listener(*self:* rti.connextdds.ServiceRequest.DataWriter, *listener:* rti.connextdds.ServiceRequest.DataWriterListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and event mask for the DataWriter.

property topic

Get the Topic object associated with this DataWriter.

property topic_name

Get the topic name associated with this DataWriter.

property type_name

Get the type name for the topic object associated with this DataWriter.

unregister_instance(**args*, ***kwargs*)

Overloaded function.

1. unregister_instance(self: rti.connextdds.ServiceRequest.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.ServiceRequest.DataWriter

Unregister an instance.

2. unregister_instance(self: rti.connextdds.ServiceRequest.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> rti.connextdds.ServiceRequest.DataWriter

Unregister an instance with timestamp.

3. unregister_instance(self: rti.connextdds.ServiceRequest.DataWriter, params: rti.connextdds.WriteParams) -> None

Unregister an instance with parameters.

unregister_instance_async(**args*, ***kwargs*)

Overloaded function.

1. unregister_instance_async(self: rti.connextdds.ServiceRequest.DataWriter, handle: rti.connextdds.InstanceHandle) -> object

Unregister an instance.

2. unregister_instance_async(self: rti.connextdds.ServiceRequest.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> object

Unregister an instance with timestamp.

3. unregister_instance_async(self: rti.connextdds.ServiceRequest.DataWriter, key_holder: rti.connextdds.ServiceRequest) -> object

Unregister the instance associated with key_holder.

4. unregister_instance_async(self: rti.connextdds.ServiceRequest.DataWriter, key_holder: rti.connextdds.ServiceRequest, timestamp: rti.connextdds.Time) -> object

Unregister the instance associate with key_holder using a timestamp.

5. unregister_instance_async(self: rti.connextdds.ServiceRequest.DataWriter, params: rti.connextdds.WriteParams) -> object

Unregister an instance with parameters.

wait_for_acknowledgments(*self:* rti.connextdds.ServiceRequest.DataWriter, *max_wait:* rti.connextdds.Duration) → None

Blocks the calling thread until all data written by a reliable DataWriter is acknowledged or until the timeout expires.

wait_for_asynchronous_publishing(*self:* rti.connextdds.ServiceRequest.DataWriter, *max_wait:* rti.connextdds.Duration) → None

This operation blocks the calling thread (up to max_wait) until all data written by the asynchronous DataWriter is sent and acknowledged (if reliable) by all matched DataReader entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; a time out indicates that max_wait elapsed before all the data was sent and/or acknowledged.

In other words, this guarantees that sending to best effort DataReader is complete in addition to what DataWriter.wait_for_acknowledgments() provides.

If the DataWriter does not have PublishMode kind set to PublishModeKind.ASYNCHRONOUS the operation will complete immediately

wait_for_asynchronous_publishing_async(*self:* rti.connextdds.ServiceRequest.DataWriter, *max_wait:* rti.connextdds.Duration) → object

This function is awaitable until either a timeout of max_wait or all data written by the asynchronous DataWriter is sent and acknowledged (if reliable) by all matched DataReader entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; a time out indicates that max_wait elapsed before all the data was sent and/or acknowledged.This function works with asyncio.

In other words, this guarantees that sending to best effort DataReader is complete in addition to what DataWriter.wait_for_acknowledgments() provides.

If the DataWriter does not have PublishMode kind set to PublishModeKind.ASYNCHRONOUS the operation will complete immediately

write(**args*, ***kwargs*)

Overloaded function.

1. write(self: rti.connextdds.ServiceRequest.DataWriter, samples: rti.connextdds.ServiceRequestSeq) -> None

Write a sequence of samples.

2. write(self: rti.connextdds.ServiceRequest.DataWriter, samples: rti.connextdds.ServiceRequestSeq, timestamp: rti.connextdds.Time) -> None

Write a sequence of samples with a timestamp.

3. write(self: rti.connextdds.ServiceRequest.DataWriter, sample: rti.connextdds.ServiceRequest) -> None

Write a sample.

4. write(self: rti.connextdds.ServiceRequest.DataWriter, sample: rti.connextdds.ServiceRequest, timestamp: rti.connextdds.Time) -> None

Write a sample with a specified timestamp.

5. write(self: rti.connextdds.ServiceRequest.DataWriter, sample: rti.connextdds.ServiceRequest, handle: rti.connextdds.InstanceHandle) -> None

Write a sample with an instance handle.

6. write(self: rti.connextdds.ServiceRequest.DataWriter, sample: rti.connextdds.ServiceRequest, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> None

Write a sample with an instance handle and specified timestamp.

7. write(self: rti.connextdds.ServiceRequest.DataWriter, sample: rti.connextdds.ServiceRequest, params: rti.connextdds.WriteParams) -> None

Write with advanced parameters.

write_async(**args*, ***kwargs*)

Overloaded function.

1. write_async(self: rti.connextdds.ServiceRequest.DataWriter, sample: rti.connextdds.ServiceRequest) -> object

Write a sample. This method is awaitable and is only for use with asyncio.

2. write_async(self: rti.connextdds.ServiceRequest.DataWriter, sample: rti.connextdds.ServiceRequest, timestamp: rti.connextdds.Time) -> object

Write a sample with a specified timestamp. This methods is awaitable and only for use with asyncio.

3. write_async(self: rti.connextdds.ServiceRequest.DataWriter, sample: rti.connextdds.ServiceRequest, handle: rti.connextdds.InstanceHandle) -> object

Write a sample with an instance handle. This method is awaitable and only for use with asyncio.

4. write_async(self: rti.connextdds.ServiceRequest.DataWriter, sample: rti.connextdds.ServiceRequest, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> object

Write a sample with an instance handle and specified timestamp. This method is awaitable and only for use with asyncio.

5. write_async(self: rti.connextdds.ServiceRequest.DataWriter, samples: rti.connextdds.ServiceRequestSeq) -> object

Write a sequence of samples. This method is awaitable and only for use with asyncio.

6. write_async(self: rti.connextdds.ServiceRequest.DataWriter, samples: rti.connextdds.ServiceRequestSeq, timestamp: rti.connextdds.Time) -> object

Write a sequence of samples with a timestamp. This method is awaitable and only for use with asyncio.

7. write_async(self: rti.connextdds.ServiceRequest.DataWriter, samples: rti.connextdds.ServiceRequestSeq, handles: rti.connextdds.InstanceHandleSeq) -> object

Write a sequence of samples with their instance handles. This method is awaitable and only for use with asyncio.

8. write_async(self: rti.connextdds.ServiceRequest.DataWriter, samples: rti.connextdds.ServiceRequestSeq, handles: rti.connextdds.InstanceHandleSeq, timestamp: rti.connextdds.Time) -> object

Write a sequence of samples with their instance handles and a timestamp. This method is awaitable and only for use with asyncio.

9. write_async(self: rti.connextdds.ServiceRequest.DataWriter, sample: rti.connextdds.ServiceRequest, params: rti.connextdds.WriteParams) -> object

Write with advanced parameters.

class DataWriterListener

Bases: `pybind11_object`

__init__ (*self:* rti.connextdds.ServiceRequest.DataWriterListener) → None

__module__ = 'rti.connextdds'

on_application_acknowledgment (*self:* rti.connextdds.ServiceRequest.DataWriterListener, *arg0:* rti.connextdds.ServiceRequest.DataWriter, *arg1:* rti.connextdds.AcknowledgmentInfo) → None

On application acknowledgment callback

on_instance_replaced (*self:* rti.connextdds.ServiceRequest.DataWriterListener, *arg0:* rti.connextdds.ServiceRequest.DataWriter, *arg1:* rti.connextdds.InstanceHandle) → None

On instance replaced callback.

on_liveliness_lost (*self:* rti.connextdds.ServiceRequest.DataWriterListener, *arg0:* rti.connextdds.ServiceRequest.DataWriter, *arg1:* rti.connextdds.LivelinessLostStatus) → None

Liveliness lost callback.

on_offered_deadline_missed (*self:* rti.connextdds.ServiceRequest.DataWriterListener, *arg0:* rti.connextdds.ServiceRequest.DataWriter, *arg1:* rti.connextdds.OfferedDeadlineMissedStatus) → None

Offered deadline missed callback.

on_offered_incompatible_qos (*self:* rti.connextdds.ServiceRequest.DataWriterListener, *arg0:* rti.connextdds.ServiceRequest.DataWriter, *arg1:* rti.connextdds.OfferedIncompatibleQosStatus) → None

Offered incompatible QoS callback.

on_publication_matched (*self:* rti.connextdds.ServiceRequest.DataWriterListener, *arg0:* rti.connextdds.ServiceRequest.DataWriter, *arg1:* rti.connextdds.PublicationMatchedStatus) → None

Publication matched callback.

on_reliable_reader_activity_changed(*self:* rti.connextdds.ServiceRequest.DataWriterListener, *arg0:* rti.connextdds.ServiceRequest.DataWriter, *arg1:* rti.connextdds.ReliableReaderActivityChangedStatus) → None

Reliable reader activity changed callback.

on_reliable_writer_cache_changed(*self:* rti.connextdds.ServiceRequest.DataWriterListener, *arg0:* rti.connextdds.ServiceRequest.DataWriter, *arg1:* rti.connextdds.ReliableWriterCacheChangedStatus) → None

Reliable writer cache changed callback.

on_service_request_accepted(*self:* rti.connextdds.ServiceRequest.DataWriterListener, *arg0:* rti.connextdds.ServiceRequest.DataWriter, *arg1:* rti.connextdds.ServiceRequestAcceptedStatus) → None

On service request accepted callback.

class DataWriterSeq

Bases: `pybind11_object`

__add__(*self:* rti.connextdds.ServiceRequest.DataWriterSeq, *arg0:* rti.connextdds.ServiceRequest.DataWriterSeq) → *rti.connextdds.ServiceRequest.DataWriterSeq*

__bool__(*self:* rti.connextdds.ServiceRequest.DataWriterSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.ServiceRequest.DataWriterSeq, *x:* rti.connextdds.ServiceRequest.DataWriter) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.ServiceRequest.DataWriterSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.ServiceRequest.DataWriterSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.ServiceRequest.DataWriterSeq, *arg0:* rti.connextdds.ServiceRequest.DataWriterSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.ServiceRequest.DataWriterSeq, s: slice) -> rti.connextdds.ServiceRequest.DataWriterSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.ServiceRequest.DataWriterSeq, arg0: int) -> rti.connextdds.ServiceRequest.DataWriter

__hash__ = None

__iadd__(*self:* rti.connextdds.ServiceRequest.DataWriterSeq, *arg0:* rti.connextdds.ServiceRequest.DataWriterSeq) → *rti.connextdds.ServiceRequest.DataWriterSeq*

__imul__(*self:* rti.connextdds.ServiceRequest.DataWriterSeq, *arg0: int*) → *rti.connextdds.ServiceRequest.DataWriterSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ServiceRequest.DataWriterSeq) -> None
2. __init__(self: rti.connextdds.ServiceRequest.DataWriterSeq, arg0: rti.connextdds.ServiceRequest.DataWriterSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.ServiceRequest.DataWriterSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.ServiceRequest.DataWriterSeq) → Iterator

__len__(*self:* rti.connextdds.ServiceRequest.DataWriterSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.ServiceRequest.DataWriterSeq, *arg0: int*) → *rti.connextdds.ServiceRequest.DataWriterSeq*

__ne__(*self:* rti.connextdds.ServiceRequest.DataWriterSeq, *arg0:* rti.connextdds.ServiceRequest.DataWriterSeq) → bool

__rmul__(*self:* rti.connextdds.ServiceRequest.DataWriterSeq, *arg0: int*) → *rti.connextdds.ServiceRequest.DataWriterSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.ServiceRequest.DataWriterSeq, arg0: int, arg1: rti.connextdds.ServiceRequest.DataWriter) -> None
2. __setitem__(self: rti.connextdds.ServiceRequest.DataWriterSeq, arg0: slice, arg1: rti.connextdds.ServiceRequest.DataWriterSeq) -> None

Assign list elements using a slice object

append(*self:* rti.connextdds.ServiceRequest.DataWriterSeq, *x:* rti.connextdds.ServiceRequest.DataWriter) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.ServiceRequest.DataWriterSeq) → None

Clear the contents

count (*self:* rti.connextdds.ServiceRequest.DataWriterSeq, *x:* rti.connextdds.ServiceRequest.DataWriter) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.ServiceRequest.DataWriterSeq, L: rti.connextdds.ServiceRequest.DataWriterSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.ServiceRequest.DataWriterSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.ServiceRequest.DataWriterSeq, *i: int*, *x:* rti.connextdds.ServiceRequest.DataWriter) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.ServiceRequest.DataWriterSeq) -> rti.connextdds.ServiceRequest.DataWriter

Remove and return the last item

2. pop(self: rti.connextdds.ServiceRequest.DataWriterSeq, i: int) -> rti.connextdds.ServiceRequest.DataWriter

Remove and return the item at index `i`

remove (*self:* rti.connextdds.ServiceRequest.DataWriterSeq, *x:* rti.connextdds.ServiceRequest.DataWriter) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class ITopicDescription

Bases: *IEntity*

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property name

The name of the entity conforming to the ITopicDescription interface.

property participant

The parent DomainParticipant.

property type_name

The name of the associated type.

class LoanedSample

Bases: `pybind11_object`

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ServiceRequest.LoanedSample) -> None

Basic constructor

2. __init__(self: rti.connextdds.ServiceRequest.LoanedSample, data: rti.connextdds.ServiceRequest, info: rti.connextdds.SampleInfo) -> None

Construct LoanedSample with data and info.

__iter__ (*self:* rti.connextdds.ServiceRequest.LoanedSample) → object

__module__ = 'rti.connextdds'

property data

Get the data associated with the sample.

property info

Get the info associated with the sample.

class LoanedSamples

Bases: pybind11_object

__enter__ (*self:* rti.connextdds.ServiceRequest.LoanedSamples) → *rti.connextdds.ServiceRequest.LoanedSamples*

Enter a context for the loaned samples, loan returned on context exit.

__exit__ (*self:* rti.connextdds.ServiceRequest.LoanedSamples, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for the loaned samples, returning the resources.

__getitem__ (*self:* rti.connextdds.ServiceRequest.LoanedSamples, *arg0: int*) → *rti.connextdds.ServiceRequest.LoanedSample*

Access a LoanedSample object in an array-like syntax

__init__ (*self:* rti.connextdds.ServiceRequest.LoanedSamples) → None

Create an empty LoanedSamples object.

__iter__ (*self:* rti.connextdds.ServiceRequest.LoanedSamples) → Iterator

__len__ (*self:* rti.connextdds.ServiceRequest.LoanedSamples) → int

Get the number of samples in the loan.

__module__ = 'rti.connextdds'

property length

Get the number of samples in the loan.

return_loan (*self:* rti.connextdds.ServiceRequest.LoanedSamples) → None

Returns the loan to the DataReader.

class NoOpDataReaderListener

Bases: *DataReaderListener*

__init__ (*self:* rti.connextdds.ServiceRequest.NoOpDataReaderListener) → None

__module__ = 'rti.connextdds'

on_data_available (*self:* rti.connextdds.ServiceRequest.NoOpDataReaderListener, *arg0:* rti.connextdds.ServiceRequest.DataReader) → None

Data available callback.

on_liveliness_changed (*self:* rti.connextdds.ServiceRequest.NoOpDataReaderListener, *arg0:* rti.connextdds.ServiceRequest.DataReader, *arg1:* rti.connextdds.LivelinessChangedStatus) → None

Liveliness changed callback.

on_requested_deadline_missed (*self:* rti.connextdds.ServiceRequest.NoOpDataReaderListener, *arg0:* rti.connextdds.ServiceRequest.DataReader, *arg1:* rti.connextdds.RequestedDeadlineMissedStatus) → None

Requested deadline missed callback.

on_requested_incompatible_qos (*self:* rti.connextdds.ServiceRequest.NoOpDataReaderListener, *arg0:* rti.connextdds.ServiceRequest.DataReader, *arg1:* rti.connextdds.RequestedIncompatibleQosStatus) → None

Requested incompatible QoS callback.

on_sample_lost (*self:* rti.connextdds.ServiceRequest.NoOpDataReaderListener, *arg0:* rti.connextdds.ServiceRequest.DataReader, *arg1:* rti.connextdds.SampleLostStatus) → None

Sample lost callback.

on_sample_rejected (*self:* rti.connextdds.ServiceRequest.NoOpDataReaderListener, *arg0:* rti.connextdds.ServiceRequest.DataReader, *arg1:* rti.connextdds.SampleRejectedStatus) → None

Sample rejected callback.

on_subscription_matched (*self:* rti.connextdds.ServiceRequest.NoOpDataReaderListener, *arg0:* rti.connextdds.ServiceRequest.DataReader, *arg1:* rti.connextdds.SubscriptionMatchedStatus) → None

Subscription matched callback.

class NoOpDataWriterListener

Bases: *DataWriterListener*

__init__ (*self:* rti.connextdds.ServiceRequest.NoOpDataWriterListener) → None

__module__ = 'rti.connextdds'

on_application_acknowledgment(*self:* rti.connextdds.ServiceRequest.NoOpDataWriterListener, *arg0:* rti.connextdds.ServiceRequest.DataWriter, *arg1:* rti.connextdds.AcknowledgmentInfo) → None

On application acknowledgment callback

on_instance_replaced(*self:* rti.connextdds.ServiceRequest.NoOpDataWriterListener, *arg0:* rti.connextdds.ServiceRequest.DataWriter, *arg1:* rti.connextdds.InstanceHandle) → None

On instance replaced callback.

on_liveliness_lost(*self:* rti.connextdds.ServiceRequest.NoOpDataWriterListener, *arg0:* rti.connextdds.ServiceRequest.DataWriter, *arg1:* rti.connextdds.LivelinessLostStatus) → None

Liveliness lost callback.

on_offered_deadline_missed(*self:* rti.connextdds.ServiceRequest.NoOpDataWriterListener, *arg0:* rti.connextdds.ServiceRequest.DataWriter, *arg1:* rti.connextdds.OfferedDeadlineMissedStatus) → None

Offered deadline missed callback.

on_offered_incompatible_qos(*self:* rti.connextdds.ServiceRequest.NoOpDataWriterListener, *arg0:* rti.connextdds.ServiceRequest.DataWriter, *arg1:* rti.connextdds.OfferedIncompatibleQosStatus) → None

Offered incompatible QoS callback.

on_publication_matched(*self:* rti.connextdds.ServiceRequest.NoOpDataWriterListener, *arg0:* rti.connextdds.ServiceRequest.DataWriter, *arg1:* rti.connextdds.PublicationMatchedStatus) → None

Publication matched callback.

on_reliable_reader_activity_changed(*self:* rti.connextdds.ServiceRequest.NoOpDataWriterListener, *arg0:* rti.connextdds.ServiceRequest.DataWriter, *arg1:* rti.connextdds.ReliableReaderActivityChangedStatus) → None

Reliable reader activity changed callback.

on_reliable_writer_cache_changed(*self:* rti.connextdds.ServiceRequest.NoOpDataWriterListener, *arg0:* rti.connextdds.ServiceRequest.DataWriter, *arg1:* rti.connextdds.ReliableWriterCacheChangedStatus) → None

Reliable writer cache changed callback.

on_service_request_accepted(*self:* rti.connextdds.ServiceRequest.NoOpDataWriterListener, *arg0:* rti.connextdds.ServiceRequest.DataWriter, *arg1:* rti.connextdds.ServiceRequestAcceptedStatus) → None

On service request accepted callback.

class NoOpTopicListener

Bases: *TopicListener*

__init__(*self:* rti.connextdds.ServiceRequest.NoOpTopicListener) → None

__module__ = 'rti.connextdds'

on_inconsistent_topic(*self:* rti.connextdds.ServiceRequest.NoOpTopicListener, *arg0:* rti.connextdds.ServiceRequest.Topic, *arg1:* rti.connextdds.InconsistentTopicStatus) → None

Inconsistent topic callback.

class Sample

Bases: pybind11_object

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ServiceRequest.Sample, data: rti.connextdds.ServiceRequest, info: rti.connextdds.SampleInfo) -> None

Construct Sample with data and info.

2. __init__(self: rti.connextdds.ServiceRequest.Sample, sample: rti.connextdds.ServiceRequest.Sample) -> None

Copy constructor.

3. __init__(self: rti.connextdds.ServiceRequest.Sample) -> None

Basic constructor

4. __init__(self: rti.connextdds.ServiceRequest.Sample, loaned_sample: rti.connextdds.ServiceRequest.LoanedSample) -> None

Construct a sample with a loaned sample.

__iter__(*self:* rti.connextdds.ServiceRequest.Sample) → object

__module__ = 'rti.connextdds'

property data

The data associated with the sample.

property info

Get the info associated with the sample.

class SharedSamples

Bases: pybind11_object

__getitem__(*self:* rti.connextdds.ServiceRequest.SharedSamples, *arg0: int*) → *rti.connextdds.ServiceRequest.LoanedSample*

Get the sample at the specified index.

__init__(*self:* rti.connextdds.ServiceRequest.SharedSamples, *loaned_samples:* rti.connextdds.ServiceRequest.LoanedSamples) → None

Constructs an instance of SharedSamples, removing ownership of the loan from the Loaned Samples.

__iter__(*self:* rti.connextdds.ServiceRequest.SharedSamples) → Iterator

Make a sample iterator

__len__(*self:* rti.connextdds.ServiceRequest.SharedSamples) → int

Returns the number of samples.

__module__ = 'rti.connextdds'

class Topic

Bases: *ITopicDescription*, *IAnyTopic*

__eq__(*self:* rti.connextdds.ServiceRequest.Topic, *arg0:* rti.connextdds.ServiceRequest.Topic) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ServiceRequest.Topic, entity: rti.connextdds.IEntity) -> None

Cast an Entity to a Topic.

2. __init__(self: rti.connextdds.ServiceRequest.Topic, topic_description: rti.connextdds.ServiceRequest.ITopicDescription) -> None

Cast an ITopicDescription to a Topic.

3. __init__(self: rti.connextdds.ServiceRequest.Topic, topic: rti.connextdds.IAnyTopic) -> None

Create a typed Topic from an AnyTopic.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.ServiceRequest.Topic, *arg0:* rti.connextdds.ServiceRequest.Topic) → bool

Test for inequality.

static find(*participant:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.ServiceRequest.Topic*]

Look up a Topic by its name in the DomainParticipant.

property inconsistent_topic_status

Get a copy of the current InconsistentTopicStatus for this Topic.

property listener

The listener.

property qos

Get the TopicQos for this Topic.

This property's getter returns a deep copy.

set_listener(*self:* rti.connextdds.ServiceRequest.Topic, *listener:* rti.connextdds.ServiceRequest.TopicListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and event mask.

class TopicDescription

Bases: *ITopicDescription*

__eq__(*self:* rti.connextdds.ServiceRequest.TopicDescription, *arg0:* rti.connextdds.ServiceRequest.TopicDescription) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ServiceRequest.TopicDescription, topic: rti.connextdds.ServiceRequest.ITopicDescription) -> None

Cast a Topic to a TopicDescription.

2. __init__(self: rti.connextdds.ServiceRequest.TopicDescription, entity: rti.connextdds.IEntity) -> None

Cast a Topic to a TopicDescription.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.ServiceRequest.TopicDescription, *arg0:* rti.connextdds.ServiceRequest.TopicDescription) → bool

Test for inequality.

class TopicListener

Bases: pybind11_object

__init__(*self:* rti.connextdds.ServiceRequest.TopicListener) → None

__module__ = 'rti.connextdds'

on_inconsistent_topic(*self:* rti.connextdds.ServiceRequest.TopicListener, *arg0:* rti.connextdds.ServiceRequest.Topic, *arg1:* rti.connextdds.InconsistentTopicStatus) → None

Inconsistent topic callback.

class TopicSeq

Bases: pybind11_object

__add__ (*self:* rti.connextdds.ServiceRequest.TopicSeq, *arg0:* rti.connextdds.ServiceRequest.TopicSeq) → *rti.connextdds.ServiceRequest.TopicSeq*

__bool__ (*self:* rti.connextdds.ServiceRequest.TopicSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.ServiceRequest.TopicSeq, *x:* rti.connextdds.ServiceRequest.Topic) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.ServiceRequest.TopicSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.ServiceRequest.TopicSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.ServiceRequest.TopicSeq, *arg0:* rti.connextdds.ServiceRequest.TopicSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.ServiceRequest.TopicSeq, s: slice) -> rti.connextdds.ServiceRequest.TopicSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.ServiceRequest.TopicSeq, arg0: int) -> rti.connextdds.ServiceRequest.Topic

__hash__ = None

__iadd__ (*self:* rti.connextdds.ServiceRequest.TopicSeq, *arg0:* rti.connextdds.ServiceRequest.TopicSeq) → *rti.connextdds.ServiceRequest.TopicSeq*

__imul__ (*self:* rti.connextdds.ServiceRequest.TopicSeq, *arg0: int*) → *rti.connextdds.ServiceRequest.TopicSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ServiceRequest.TopicSeq) -> None
2. __init__(self: rti.connextdds.ServiceRequest.TopicSeq, arg0: rti.connextdds.ServiceRequest.TopicSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.ServiceRequest.TopicSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.ServiceRequest.TopicSeq) → Iterator

__len__ (*self:* rti.connextdds.ServiceRequest.TopicSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.ServiceRequest.TopicSeq, *arg0: int*) → *rti.connextdds.ServiceRequest.TopicSeq*

__ne__ (*self:* rti.connextdds.ServiceRequest.TopicSeq, *arg0:* rti.connextdds.ServiceRequest.TopicSeq) → bool

__rmul__ (*self:* rti.connextdds.ServiceRequest.TopicSeq, *arg0: int*) → *rti.connextdds.ServiceRequest.TopicSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.ServiceRequest.TopicSeq, arg0: int, arg1: rti.connextdds.ServiceRequest.Topic) -> None
2. __setitem__(self: rti.connextdds.ServiceRequest.TopicSeq, arg0: slice, arg1: rti.connextdds.ServiceRequest.TopicSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.ServiceRequest.TopicSeq, *x:* rti.connextdds.ServiceRequest.Topic) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.ServiceRequest.TopicSeq) → None

Clear the contents

count (*self:* rti.connextdds.ServiceRequest.TopicSeq, *x:* rti.connextdds.ServiceRequest.Topic) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.ServiceRequest.TopicSeq, L: rti.connextdds.ServiceRequest.TopicSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.ServiceRequest.TopicSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.ServiceRequest.TopicSeq, *i: int*, *x:* rti.connextdds.ServiceRequest.Topic) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.ServiceRequest.TopicSeq) -> rti.connextdds.ServiceRequest.Topic

Remove and return the last item

2. pop(self: rti.connextdds.ServiceRequest.TopicSeq, i: int) -> rti.connextdds.ServiceRequest.Topic

Remove and return the item at index `i`

remove (*self:* rti.connextdds.ServiceRequest.TopicSeq, *x:* rti.connextdds.ServiceRequest.Topic) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class ValidLoanedSamples

Bases: pybind11_object

__enter__ (*self:* rti.connextdds.ServiceRequest.ValidLoanedSamples) → *rti.connextdds.ServiceRequest.ValidLoanedSamples*

__exit__ (*self:* rti.connextdds.ServiceRequest.ValidLoanedSamples, *arg0: object*, *arg1: object*, *arg2: object*) → None

__init__ (**args*, ***kwargs*)

__iter__ (*self:* rti.connextdds.ServiceRequest.ValidLoanedSamples) → Iterator

__module__ = 'rti.connextdds'

class WriterContentFilter

Bases: *ContentFilter*

__init__ (*self:* rti.connextdds.ServiceRequest.WriterContentFilter) → None

__module__ = 'rti.connextdds'

writer_attach (*self:* rti.connextdds.ServiceRequest.WriterContentFilter) → Optional[object]

A writer-side filtering API to create some state that can facilitate filtering on the writer side.

writer_compile (*self:* rti.connextdds.ServiceRequest.WriterContentFilter, *writer_filter_data: Optional[object]*, *property:* rti.connextdds.ExpressionProperty, *expression: str*, *parameters:* rti.connextdds.StringSeq, *type_code: Optional[*rti.connextdds.DynamicType*]*, *type_class_name: str*, *cookie:* rti.connextdds.Cookie) → None

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching DataReader.

writer_detach (*self:* rti.connextdds.ServiceRequest.WriterContentFilter, *writer_filter_data: Optional[object]*) → None

A writer-side filtering API to clean up a previously created state using writer_attach.

writer_evaluate (*self:* rti.connextdds.ServiceRequest.WriterContentFilter, *writer_filter_data: Optional[object]*, *sample:* rti.connextdds.ServiceRequest, *meta_data:* rti.connextdds.FilterSampleInfo) → *rti.connextdds.CookieVector*

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching DataReader.

writer_finalize (*self:* rti.connextdds.ServiceRequest.WriterContentFilter, *writer_filter_data: Optional[object]*, *cookie:* rti.connextdds.Cookie) → None

A writer-side filtering API to clean up a previously compiled instance of the content filter.

writer_return_loan (*self:* rti.connextdds.ServiceRequest.WriterContentFilter, *writer_filter_data: Optional[object]*, *cookies:* rti.connextdds.CookieVector) → None

A writer-side filtering API to return the loan on the list of DataReaders returned by writer_evaluate.

class WriterContentFilterHelper

Bases: *WriterContentFilter*

__init__ (*self:* rti.connextdds.ServiceRequest.WriterContentFilterHelper) → None

__module__ = 'rti.connextdds'

add_cookie (*self:* rti.connextdds.ServiceRequest.WriterContentFilterHelper, *cookie:* rti.connextdds.Cookie) → None

A helper function which will add a Cookie to the Cookie sequence that is then returned by the writer_evaluate function.

writer_evaluate_helper (*self:* rti.connextdds.ServiceRequest.WriterContentFilterHelper, *writer_filter_data: Optional[object]*, *sample:* rti.connextdds.ServiceRequest, *meta_data:* rti.connextdds.FilterSampleInfo) → None

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching DataReader.

__eq__ (*self:* rti.connextdds.ServiceRequest, *arg0:* rti.connextdds.ServiceRequest) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ServiceRequest, *arg0:* rti.connextdds.ServiceRequest) → bool

Test for inequality.

property instance_id

Get the instance id of the request.

property request_body

Get the request body of the request.

property service_id

The service id of the request.

topic_name = 'DDSServiceRequest'

class rti.connextdds.**ServiceRequestAcceptedStatus**

Bases: pybind11_object

Information about the service_request_accepted status Currently, the only service that causes the ServiceRequestAcceptedStatus to be triggered is TopicQuery service. A ServiceRequest is accepted when a DataWriter matches with a DataReader that has created a TopicQuery. This status is also changed (and the listener, if any, called) when a ServiceRequest has been cancelled, or deleted. This will happen when a DataReader deletes a TopicQuery using TopicQuery.close().

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property current_count

The current number of ServiceRequests that have been accepted by this DataWriter.

property last_request_handle

A handle to the last ServiceRequest that caused the DataWriter's status to change.

property service_id

ID of the service to which the accepted Request belongs.

property total_count

The total number of ServiceRequests that have been accepted by this DataWriter.

class rti.connextdds.**ServiceRequestId**

Bases: pybind11_object

LOCATOR_REACHABILITY = <ServiceRequestId.LOCATOR_REACHABILITY: 2>

MONITORING_LIBRARY_COMMAND =
<ServiceRequestId.MONITORING_LIBRARY_COMMAND: 4>

MONITORING_LIBRARY_REPLY =
<ServiceRequestId.MONITORING_LIBRARY_REPLY: 5>

class ServiceRequestId

Bases: pybind11_object

Members:

UNKNOWN : An unknown service.

TOPIC_QUERY : The topic query service.

LOCATOR_REACHABILITY : The locator reachability service.

MONITORING_LIBRARY_COMMAND : The Monitoring Library 2.0 command service.

MONITORING_LIBRARY_REPLY : The Monitoring Library 2.0 reply service.

LOCATOR_REACHABILITY = <ServiceRequestId.LOCATOR_REACHABILITY: 2>

MONITORING_LIBRARY_COMMAND = <ServiceRequestId.MONITORING_LIBRARY_COMMAND: 4>

MONITORING_LIBRARY_REPLY = <ServiceRequestId.MONITORING_LIBRARY_REPLY: 5>

TOPIC_QUERY = <ServiceRequestId.TOPIC_QUERY: 1>

UNKNOWN = <ServiceRequestId.UNKNOWN: 0>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.ServiceRequestId.ServiceRequestId) → int

__init__(*self:* rti.connextdds.ServiceRequestId.ServiceRequestId, *value: int*) → None

__int__(*self:* rti.connextdds.ServiceRequestId.ServiceRequestId) → int

__members__ = {'LOCATOR_REACHABILITY': <ServiceRequestId.LOCATOR_REACHABILITY: 2>, 'MONITORING_LIBRARY_COMMAND': <ServiceRequestId.MONITORING_LIBRARY_COMMAND: 4>, 'MONITORING_LIBRARY_REPLY': <ServiceRequestId.MONITORING_LIBRARY_REPLY: 5>, 'TOPIC_QUERY': <ServiceRequestId.TOPIC_QUERY: 1>, 'UNKNOWN': <ServiceRequestId.UNKNOWN: 0>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.ServiceRequestId.ServiceRequestId, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

TOPIC_QUERY = <ServiceRequestId.TOPIC_QUERY: 1>

UNKNOWN = <ServiceRequestId.UNKNOWN: 0>

__eq__ (*self:* rti.connextdds.ServiceRequestId, *arg0:* rti.connextdds.ServiceRequestId) → bool
Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.ServiceRequestId, *arg0:* rti.connextdds.ServiceRequestId) → bool
Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.ServiceRequestId, *arg0:* rti.connextdds.ServiceRequestId) → bool
Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)
Overloaded function.

1. __init__(self: rti.connextdds.ServiceRequestId) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.ServiceRequestId, arg0: rti.connextdds.ServiceRequestId.ServiceRequestId) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.ServiceRequestId) → *rti.connextdds.ServiceRequestId.ServiceRequestId*
Int conversion.

__le__ (*self:* rti.connextdds.ServiceRequestId, *arg0:* rti.connextdds.ServiceRequestId) → bool
Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.ServiceRequestId, *arg0:* rti.connextdds.ServiceRequestId) → bool
Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ServiceRequestId, *arg0:* rti.connextdds.ServiceRequestId) → bool
Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.ServiceRequestId) → str
String conversion.

property underlying
Retrieves the actual enumerated value.

class rti.connextdds.**ServiceRequestSeq**
Bases: pybind11_object

__add__ (*self:* rti.connextdds.ServiceRequestSeq, *arg0:* rti.connextdds.ServiceRequestSeq) → *rti.connextdds.ServiceRequestSeq*

__bool__ (*self:* rti.connextdds.ServiceRequestSeq) → bool
Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.ServiceRequestSeq, *x:* rti.connextdds.ServiceRequest) → bool
Return true the container contains x

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.ServiceRequestSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.ServiceRequestSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.ServiceRequestSeq, *arg0:* rti.connextdds.ServiceRequestSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.ServiceRequestSeq, s: slice) -> rti.connextdds.ServiceRequestSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.ServiceRequestSeq, arg0: int) -> rti.connextdds.ServiceRequest

__hash__ = None

__iadd__ (*self:* rti.connextdds.ServiceRequestSeq, *arg0:* rti.connextdds.ServiceRequestSeq) → *rti.connextdds.ServiceRequestSeq*

__imul__ (*self:* rti.connextdds.ServiceRequestSeq, *arg0: int*) → *rti.connextdds.ServiceRequestSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ServiceRequestSeq) -> None

2. __init__(self: rti.connextdds.ServiceRequestSeq, arg0: rti.connextdds.ServiceRequestSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.ServiceRequestSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.ServiceRequestSeq) → Iterator

__len__ (*self:* rti.connextdds.ServiceRequestSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.ServiceRequestSeq, *arg0: int*) → *rti.connextdds.ServiceRequestSeq*

__ne__ (*self:* rti.connextdds.ServiceRequestSeq, *arg0:* rti.connextdds.ServiceRequestSeq) → bool

__rmul__ (*self:* rti.connextdds.ServiceRequestSeq, *arg0: int*) → *rti.connextdds.ServiceRequestSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.ServiceRequestSeq, arg0: int, arg1: rti.connextdds.ServiceRequest) -> None

2. __setitem__(self: rti.connextdds.ServiceRequestSeq, arg0: slice, arg1: rti.connextdds.ServiceRequestSeq) -> None

Assign list elements using a slice object

append(*self:* rti.connextdds.ServiceRequestSeq, *x:* rti.connextdds.ServiceRequest) → None

Add an item to the end of the list

clear(*self:* rti.connextdds.ServiceRequestSeq) → None

Clear the contents

count(*self:* rti.connextdds.ServiceRequestSeq, *x:* rti.connextdds.ServiceRequest) → int

Return the number of times `x` appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.ServiceRequestSeq, L: rti.connextdds.ServiceRequestSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.ServiceRequestSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert(*self:* rti.connextdds.ServiceRequestSeq, *i: int*, *x:* rti.connextdds.ServiceRequest) → None

Insert an item at a given position.

pop(**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.ServiceRequestSeq) -> rti.connextdds.ServiceRequest

Remove and return the last item

2. pop(self: rti.connextdds.ServiceRequestSeq, i: int) -> rti.connextdds.ServiceRequest

Remove and return the item at index `i`

remove(*self:* rti.connextdds.ServiceRequestSeq, *x:* rti.connextdds.ServiceRequest) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**ServiceRequestTimestampedSeq**

Bases: `pybind11_object`

__add__(*self:* rti.connextdds.ServiceRequestTimestampedSeq, *arg0:* rti.connextdds.ServiceRequestTimestampedSeq) → *rti.connextdds.ServiceRequestTimestampedSeq*

__bool__ (*self:* rti.connextdds.ServiceRequestTimestampedSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.ServiceRequestTimestampedSeq, *x:* *Tuple[*rti.connextdds.ServiceRequest, rti.connextdds.Time*]*) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.ServiceRequestTimestampedSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.ServiceRequestTimestampedSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.ServiceRequestTimestampedSeq, *arg0:* rti.connextdds.ServiceRequestTimestampedSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.ServiceRequestTimestampedSeq, s: slice) -> rti.connextdds.ServiceRequestTimestampedSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.ServiceRequestTimestampedSeq, arg0: int) -> Tuple[rti.connextdds.ServiceRequest, rti.connextdds.Time]

__hash__ = None

__iadd__ (*self:* rti.connextdds.ServiceRequestTimestampedSeq, *arg0:* rti.connextdds.ServiceRequestTimestampedSeq) → *rti.connextdds.ServiceRequestTimestampedSeq*

__imul__ (*self:* rti.connextdds.ServiceRequestTimestampedSeq, *arg0: int*) → *rti.connextdds.ServiceRequestTimestampedSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ServiceRequestTimestampedSeq) -> None

2. __init__(self: rti.connextdds.ServiceRequestTimestampedSeq, arg0: rti.connextdds.ServiceRequestTimestampedSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.ServiceRequestTimestampedSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.ServiceRequestTimestampedSeq) → Iterator

__len__ (*self:* rti.connextdds.ServiceRequestTimestampedSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.ServiceRequestTimestampedSeq, *arg0: int*) → *rti.connextdds.ServiceRequestTimestampedSeq*

__ne__ (*self:* rti.connextdds.ServiceRequestTimestampedSeq, *arg0:* rti.connextdds.ServiceRequestTimestampedSeq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__rmul__ (*self:* rti.connextdds.ServiceRequestTimestampedSeq, *arg0: int*) → *rti.connextdds.ServiceRequestTimestampedSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.ServiceRequestTimestampedSeq, arg0: int, arg1: Tuple[rti.connextdds.ServiceRequest, rti.connextdds.Time]) -> None

2. __setitem__(self: rti.connextdds.ServiceRequestTimestampedSeq, arg0: slice, arg1: rti.connextdds.ServiceRequestTimestampedSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.ServiceRequestTimestampedSeq, *x:* *Tuple[*rti.connextdds.ServiceRequest, rti.connextdds.Time*]*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.ServiceRequestTimestampedSeq) → None

Clear the contents

count (*self:* rti.connextdds.ServiceRequestTimestampedSeq, *x:* *Tuple[*rti.connextdds.ServiceRequest, rti.connextdds.Time*]*) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.ServiceRequestTimestampedSeq, L: rti.connextdds.ServiceRequestTimestampedSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.ServiceRequestTimestampedSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.ServiceRequestTimestampedSeq, *i: int*, *x:* *Tuple[*rti.connextdds.ServiceRequest, rti.connextdds.Time*]*) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.ServiceRequestTimestampedSeq) -> Tuple[rti.connextdds.ServiceRequest, rti.connextdds.Time]

Remove and return the last item

2. pop(self: rti.connextdds.ServiceRequestTimestampedSeq, i: int) -> Tuple[rti.connextdds.ServiceRequest, rti.connextdds.Time]

Remove and return the item at index `i`

remove (*self:* rti.connextdds.ServiceRequestTimestampedSeq, *x:* *Tuple[*rti.connextdds.ServiceRequest*,* rti.connextdds.Time*]*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

`rti.connextdds.`**ShortSeq**

alias of *Int16Seq*

`rti.connextdds.`**ShortType**

alias of *Int16Type*

class `rti.connextdds.`**StatusCondition**

Bases: *ICondition*

__eq__ (*self:* rti.connextdds.StatusCondition, *arg0:* rti.connextdds.StatusCondition) → bool

Compare StatusCondition objects for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.StatusCondition, entity: rti.connextdds.IEntity) -> None

Obtain a reference to an entity's StatusCondition object

2. __init__(self: rti.connextdds.StatusCondition, condition: rti.connextdds.ICondition) -> None

Downcast a Condition to a StatusCondition.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.StatusCondition, *arg0:* rti.connextdds.StatusCondition) → bool

Compare StatusCondition objects for inequality.

dispatch (*self:* rti.connextdds.StatusCondition) → None

Dispatches the functions registered with the condition.

property enabled_statuses

Get/set the enabled statuses for this condition.

property entity

Get the Entity associated with this StatusCondition.

reset_handler(*self:* rti.connextdds.StatusCondition) → None

Resets the handler for this StatusCondition.

set_handler(*self:* rti.connextdds.StatusCondition, *func:* *Callable[[*rti.connextdds.StatusCondition*], None]*) → None

Set a handler function for this StatusCondition.

property trigger_value

The trigger value of the condition.

class rti.connextdds.**StatusMask**

Bases: pybind11_object

A set of statuses.

ALL = 11111111111111111111111111111111

DATAREADER_CACHE = 00010000000000000000000000000000

DATAREADER_PROTOCOL = 00100000000000000000000000000000

DATAWRITER_APPLICATION_ACKNOWLEDGMENT = 00000000010000000000000000000000

DATAWRITER_CACHE = 00000100000000000000000000000000

DATAWRITER_INSTANCE_REPLACED = 00000000100000000000000000000000

DATAWRITER_PROTOCOL = 00001000000000000000000000000000

DATA_AVAILABLE = 00000000000000000000010000000000

DATA_ON_READERS = 00000000000000000000001000000000

DESTINATION_UNREACHABLE = 01000000000000000000000000000000

INCONSISTENT_TOPIC = 00000000000000000000000000000001

INVALID_LOCAL_IDENTITY_ADVANCE_NOTICE = 00000000001000000000000000000000

LIVELINESS_CHANGED = 00000000000000000001000000000000

LIVELINESS_LOST = 00000000000000000000100000000000

NONE = 00000000000000000000000000000000

OFFERED_DEADLINE_MISSED = 00000000000000000000000000000010

OFFERED_INCOMPATIBLE_QOS = 00000000000000000000000000100000

PUBLICATION_MATCHED = 00000000000000000010000000000000

RELIABLE_READER_ACTIVITY_CHANGED = 00000010000000000000000000000000

RELIABLE_WRITER_CACHE_CHANGED = 00000001000000000000000000000000

REQUESTED_DEADLINE_MISSED = 00000000000000000000000000000100

REQUESTED_INCOMPATIBLE_QOS = 00000000000000000000000001000000

SAMPLE_LOST = 00000000000000000000000010000000

SAMPLE_REJECTED = 00000000000000000000000100000000

SAMPLE_REMOVED = 10000000000000000000000000000000

SERVICE_REQUEST_ACCEPTED = 00000000001000000000000000000000

SUBSCRIPTION_MATCHED = 00000000000000000100000000000000

__and__ (*self:* rti.connextdds.StatusMask, *arg0:* rti.connextdds.StatusMask) → *rti.connextdds.StatusMask*

Bitwise logical AND of masks.

__bool__ (*self:* rti.connextdds.StatusMask) → bool

Test if any bits are set.

__contains__ (*self:* rti.connextdds.StatusMask, *arg0:* rti.connextdds.StatusMask) → bool

__eq__ (*self:* rti.connextdds.StatusMask, *arg0:* rti.connextdds.StatusMask) → bool

Compare masks for equality.

__getitem__ (*self:* rti.connextdds.StatusMask, *arg0: int*) → bool

Get individual mask bit.

__hash__ = None

__iand__ (*self:* rti.connextdds.StatusMask, *arg0:* rti.connextdds.StatusMask) → *rti.connextdds.StatusMask*

Set mask to logical AND with another mask.

__ilshift__ (*self:* rti.connextdds.StatusMask, *arg0: int*) → *rti.connextdds.StatusMask*

Left shift bits in mask.

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.StatusMask) -> None

Create a StatusMask equivalent to StatusMask.NONE

2. __init__(self: rti.connextdds.StatusMask, value: int) -> None

Creates a mask from the bits in an integer.

__int__ (*self:* rti.connextdds.StatusMask) → int

Convert mask to int.

__ior__ (*self:* rti.connextdds.StatusMask, *arg0:* rti.connextdds.StatusMask) → *rti.connextdds.StatusMask*

Set mask to logical OR with another mask.

__irshift__ (*self:* rti.connextdds.StatusMask, *arg0: int*) → *rti.connextdds.StatusMask*

Right shift bits in mask.

__ixor__ (*self:* rti.connextdds.StatusMask, *arg0:* rti.connextdds.StatusMask) → *rti.connextdds.StatusMask*

Set mask to logical XOR with another mask.

__lshift__ (*self:* rti.connextdds.StatusMask, *arg0: int*) → *rti.connextdds.StatusMask*

Left shift bits in mask.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.StatusMask, *arg0:* rti.connextdds.StatusMask) → bool

Compare masks for inequality.

__or__ (*self:* rti.connextdds.StatusMask, *arg0:* rti.connextdds.StatusMask) → *rti.connextdds.StatusMask*

Bitwise logical OR of masks.

__repr__ (*self:* rti.connextdds.StatusMask) → str

Convert mask to string.

__rshift__ (*self:* rti.connextdds.StatusMask, *arg0: int*) → *rti.connextdds.StatusMask*

Right shift bits in mask.

__setitem__ (*self:* rti.connextdds.StatusMask, *arg0: int*, *arg1: bool*) → None

Set individual mask bit

__str__ (*self:* rti.connextdds.StatusMask) → str

Convert mask to string.

__xor__ (*self:* rti.connextdds.StatusMask, *arg0:* rti.connextdds.StatusMask) → *rti.connextdds.StatusMask*

Bitwise logical XOR of masks.

property count

Returns the number of bits set in the mask.

flip (**args*, ***kwargs*)

Overloaded function.

1. flip(self: rti.connextdds.StatusMask) -> rti.connextdds.StatusMask

Flip all bits in the mask.

2. flip(self: rti.connextdds.StatusMask, pos: int) -> rti.connextdds.StatusMask

Flip the mask bit at the specified position.

reset (**args*, ***kwargs*)

Overloaded function.

1. reset(self: rti.connextdds.StatusMask) -> rti.connextdds.StatusMask

Clear all bits in the mask.

2. reset(self: rti.connextdds.StatusMask, pos: int) -> rti.connextdds.StatusMask

Clear the mask bit at the specified position.

set (**args*, ***kwargs*)

Overloaded function.

1. set(self: rti.connextdds.StatusMask) -> rti.connextdds.StatusMask

Set all bits in the mask.

2. set(self: rti.connextdds.StatusMask, pos: int, value: bool = True) -> rti.connextdds.StatusMask

Set the mask bit at the specified position to the provided value (default: true).

property size

Returns the number of bits in the mask type.

test (*self:* rti.connextdds.StatusMask, *pos: int*) → bool

Test whether the mask bit at position "pos" is set.

test_all (*self:* rti.connextdds.StatusMask) → bool

Test if all bits are set.

test_any (*self:* rti.connextdds.StatusMask) → bool

Test if any bits are set.

test_none (*self:* rti.connextdds.StatusMask) → bool

Test if none of the bits are set.

class rti.connextdds.**StreamKind**

Bases: pybind11_object

ANY = <rti.connextdds.StreamKind object>

LIVE = <rti.connextdds.StreamKind object>

TOPIC_QUERY = <rti.connextdds.StreamKind object>

__and__ (*self:* rti.connextdds.StreamKind, *arg0:* rti.connextdds.StreamKind) → *rti.connextdds.StreamKind*

Bitwise logical AND of masks.

__bool__ (*self:* rti.connextdds.StreamKind) → *rti.connextdds.StreamKind*

Test if any bits are set.

__contains__ (*self:* rti.connextdds.StreamKind, *arg0:* rti.connextdds.StreamKind) → bool

__eq__ (*self:* rti.connextdds.StreamKind, *arg0:* rti.connextdds.StreamKind) → bool

Compare masks for equality.

__getitem__ (*self:* rti.connextdds.StreamKind, *arg0: int*) → bool

Get individual mask bit.

__hash__ = None

__iand__ (*self:* rti.connextdds.StreamKind, *arg0:* rti.connextdds.StreamKind) → *rti.connextdds.StreamKind*

Set mask to logical AND with another mask.

__ilshift__ (*self:* rti.connextdds.StreamKind, *arg0: int*) → *rti.connextdds.StreamKind*

Left shift bits in mask.

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.StreamKind) -> None

Create a StreamKind with nothing enabled.

2. __init__(self: rti.connextdds.StreamKind, value: int) -> None

Creates a mask from the bits in an integer.

__int__ (*self:* rti.connextdds.StreamKind) → int

Convert mask to int.

__ior__ (*self:* rti.connextdds.StreamKind, *arg0:* rti.connextdds.StreamKind) → *rti.connextdds.StreamKind*

Set mask to logical OR with another mask.

__irshift__ (*self:* rti.connextdds.StreamKind, *arg0: int*) → *rti.connextdds.StreamKind*

Right shift bits in mask.

__ixor__ (*self:* rti.connextdds.StreamKind, *arg0:* rti.connextdds.StreamKind) → *rti.connextdds.StreamKind*

Set mask to logical XOR with another mask.

__lshift__ (*self:* rti.connextdds.StreamKind, *arg0: int*) → *rti.connextdds.StreamKind*

Left shift bits in mask.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.StreamKind, *arg0:* rti.connextdds.StreamKind) → bool

Compare masks for inequality.

__or__ (*self:* rti.connextdds.StreamKind, *arg0:* rti.connextdds.StreamKind) → *rti.connextdds.StreamKind*

Bitwise logical OR of masks.

__rshift__ (*self:* rti.connextdds.StreamKind, *arg0: int*) → *rti.connextdds.StreamKind*

Right shift bits in mask.

__setitem__ (*self:* rti.connextdds.StreamKind, *arg0: int*, *arg1: bool*) → None

Set individual mask bit

__str__ (*self:* rti.connextdds.StreamKind) → str

__xor__ (*self:* rti.connextdds.StreamKind, *arg0:* rti.connextdds.StreamKind) → *rti.connextdds.StreamKind*

Bitwise logical XOR of masks.

property count

Returns the number of bits set in the mask.

flip (**args*, ***kwargs*)

Overloaded function.

1. flip(self: rti.connextdds.StreamKind) -> rti.connextdds.StreamKind

Flip all bits in the mask.

2. flip(self: rti.connextdds.StreamKind, pos: int) -> rti.connextdds.StreamKind

Flip the mask bit at the specified position.

reset (**args*, ***kwargs*)

Overloaded function.

1. reset(self: rti.connextdds.StreamKind) -> rti.connextdds.StreamKind

Clear all bits in the mask.

2. reset(self: rti.connextdds.StreamKind, pos: int) -> rti.connextdds.StreamKind

Clear the mask bit at the specified position.

set (**args*, ***kwargs*)

Overloaded function.

1. set(self: rti.connextdds.StreamKind) -> rti.connextdds.StreamKind

Set all bits in the mask.

2. set(self: rti.connextdds.StreamKind, pos: int, value: bool = True) -> rti.connextdds.StreamKind

Set the mask bit at the specified position to the provided value (default: true).

property size

Returns the number of bits in the mask type.

test (*self:* rti.connextdds.StreamKind, *pos: int*) → bool

Test whether the mask bit at position “pos” is set.

test_all (*self:* rti.connextdds.StreamKind) → bool

Test if all bits are set.

test_any (*self:* rti.connextdds.StreamKind) → bool

Test if any bits are set.

test_none (*self:* rti.connextdds.StreamKind) → bool

Test if none of the bits are set.

class rti.connextdds.**StringMap**

Bases: pybind11_object

__bool__ (*self:* rti.connextdds.StringMap) → bool

Check whether the map is nonempty

__contains__ (**args*, ***kwargs*)

Overloaded function.

1. __contains__(self: rti.connextdds.StringMap, arg0: str) -> bool
2. __contains__(self: rti.connextdds.StringMap, arg0: object) -> bool

__delitem__ (*self:* rti.connextdds.StringMap, *arg0: str*) → None

__getitem__ (*self:* rti.connextdds.StringMap, *arg0: str*) → str

__init__ (*self:* rti.connextdds.StringMap) → None

__iter__ (*self:* rti.connextdds.StringMap) → Iterator

__len__ (*self:* rti.connextdds.StringMap) → int

__module__ = 'rti.connextdds'

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__ (*self:* rti.connextdds.StringMap) → str

Return the canonical string representation of this map.

__setitem__ (*self:* rti.connextdds.StringMap, *arg0: str*, *arg1: str*) → None

items (*self:* rti.connextdds.StringMap) → rti.connextdds.ItemsView[*StringMap*]

keys (*self:* rti.connextdds.StringMap) → rti.connextdds.KeysView[*StringMap*]

values (*self:* rti.connextdds.StringMap) → rti.connextdds.ValuesView[*StringMap*]

class rti.connextdds.**StringPairSeq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.StringPairSeq, *arg0:* rti.connextdds.StringPairSeq) → *rti.connextdds.StringPairSeq*

__bool__ (*self:* rti.connextdds.StringPairSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.StringPairSeq, *x: Tuple[str, str]*) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.StringPairSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.StringPairSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.StringPairSeq, *arg0:* rti.connextdds.StringPairSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.StringPairSeq, s: slice) -> rti.connextdds.StringPairSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.StringPairSeq, arg0: int) -> Tuple[str, str]

__hash__ = None

__iadd__ (*self:* rti.connextdds.StringPairSeq, *arg0:* rti.connextdds.StringPairSeq) → *rti.connextdds.StringPairSeq*

__imul__ (*self:* rti.connextdds.StringPairSeq, *arg0: int*) → *rti.connextdds.StringPairSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.StringPairSeq) -> None

2. __init__(self: rti.connextdds.StringPairSeq, arg0: rti.connextdds.StringPairSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.StringPairSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.StringPairSeq) → Iterator

__len__ (*self:* rti.connextdds.StringPairSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.StringPairSeq, *arg0: int*) → *rti.connextdds.StringPairSeq*

__ne__ (*self:* rti.connextdds.StringPairSeq, *arg0:* rti.connextdds.StringPairSeq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__rmul__ (*self:* rti.connextdds.StringPairSeq, *arg0: int*) → *rti.connextdds.StringPairSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.StringPairSeq, arg0: int, arg1: Tuple[str, str]) -> None
2. __setitem__(self: rti.connextdds.StringPairSeq, arg0: slice, arg1: rti.connextdds.StringPairSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.StringPairSeq, *x: Tuple[str, str]*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.StringPairSeq) → None

Clear the contents

count (*self:* rti.connextdds.StringPairSeq, *x: Tuple[str, str]*) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.StringPairSeq, L: rti.connextdds.StringPairSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.StringPairSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.StringPairSeq, *i: int*, *x: Tuple[str, str]*) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.StringPairSeq) -> Tuple[str, str]

Remove and return the last item

2. pop(self: rti.connextdds.StringPairSeq, i: int) -> Tuple[str, str]

Remove and return the item at index `i`

remove (*self:* rti.connextdds.StringPairSeq, *x: Tuple[str, str]*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**StringSeq**

Bases: `pybind11_object`

__add__ (*self:* rti.connextdds.StringSeq, *arg0:* rti.connextdds.StringSeq) → *rti.connextdds.StringSeq*

__bool__ (*self:* rti.connextdds.StringSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.StringSeq, *x: str*) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.StringSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.StringSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.StringSeq, *arg0:* rti.connextdds.StringSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.StringSeq, s: slice) -> rti.connextdds.StringSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.StringSeq, arg0: int) -> str

__hash__ = None

__iadd__ (*self:* rti.connextdds.StringSeq, *arg0:* rti.connextdds.StringSeq) → *rti.connextdds.StringSeq*

__imul__ (*self:* rti.connextdds.StringSeq, *arg0: int*) → *rti.connextdds.StringSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.StringSeq) -> None

2. __init__(self: rti.connextdds.StringSeq, arg0: rti.connextdds.StringSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.StringSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.StringSeq) → Iterator

__len__ (*self:* rti.connextdds.StringSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.StringSeq, *arg0: int*) → *rti.connextdds.StringSeq*

__ne__ (*self:* rti.connextdds.StringSeq, *arg0:* rti.connextdds.StringSeq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__ (*self:* rti.connextdds.StringSeq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.StringSeq, *arg0: int*) → *rti.connextdds.StringSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.StringSeq, arg0: int, arg1: str) -> None
2. __setitem__(self: rti.connextdds.StringSeq, arg0: slice, arg1: rti.connextdds.StringSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.StringSeq, *x: str*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.StringSeq) → None

Clear the contents

count (*self:* rti.connextdds.StringSeq, *x: str*) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.StringSeq, L: rti.connextdds.StringSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.StringSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.StringSeq, *i: int*, *x: str*) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.StringSeq) -> str

Remove and return the last item

2. pop(self: rti.connextdds.StringSeq, i: int) -> str

Remove and return the item at index `i`

remove (*self:* rti.connextdds.StringSeq, *x: str*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**StringType**

Bases: *UnidimensionalCollectionType*

__eq__(*self:* rti.connextdds.StringType, *arg0:* rti.connextdds.StringType) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.StringType, bounds: int) -> None

Creates a bounded string.

2. __init__(self: rti.connextdds.StringType) -> None

Creates an unbounded string.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.StringType, *arg0:* rti.connextdds.StringType) → bool

Test for inequality.

class rti.connextdds.**StructType**

Bases: *ACTMember*

__eq__(*self:* rti.connextdds.StructType, *arg0:* rti.connextdds.StructType) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.StructType, name: str) -> None

Creates an empty StructType.

2. __init__(self: rti.connextdds.StructType, name: str, extensibility: rti.connextdds.ExtensibilityKind) -> None

Creates an empty StructType with a specified extensibility.

3. __init__(self: rti.connextdds.StructType, name: str, parent: rti.connextdds.StructType) -> None

Create an empty StructType with a base type.

4. __init__(self: rti.connextdds.StructType, name: str, parent: rti.connextdds.StructType, extensibility: rti.connextdds.ExtensibilityKind) -> None

Creates an empty StructType with a base type and specified extensibility.

5. __init__(self: rti.connextdds.StructType, name: str, members: rti.connextdds.MemberSeq) -> None

Create a StructType with the provided members.

6. __init__(self: rti.connextdds.StructType, name: str, members: rti.connextdds.MemberSeq, extensibility: rti.connextdds.ExtensibilityKind) -> None

Creates a StructType with the provided members and specified extensibility.

7. __init__(self: rti.connextdds.StructType, name: str, parent: rti.connextdds.StructType, members: rti.connextdds.MemberSeq) -> None

Create a StructType with a base type and the provided members.

8. __init__(self: rti.connextdds.StructType, name: str, parent: rti.connextdds.StructType, members: rti.connextdds.MemberSeq, extensibility: rti.connextdds.ExtensibilityKind) -> None

Creates a StructType with a base types, the provided members, and the specified extensibility.

9. __init__(self: rti.connextdds.StructType, type: rti.connextdds.DynamicType) -> None

Cast a DynamicType to a StructType

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.StructType, *arg0:* rti.connextdds.StructType) → bool

Test for inequality.

add_member(*self:* rti.connextdds.StructType, *member:* rti.connextdds.Member) → *rti.connextdds.StructType*

Adds a member at the end.

add_members(*self:* rti.connextdds.StructType, *members:* rti.connextdds.MemberSeq) → *rti.connextdds.StructType*

Adds all members of the sequence to the end.

property extensibility_kind

Struct's extensibility kind.

find_member_by_id(*self:* rti.connextdds.StructType, *member_id: int*) → int

Get the index of the member with a specfic ID.

property has_parent

Indicates if this type has a base type.

property parent

Retrieve the base type.

class rti.connextdds.**Subscriber**

Bases: *IEntity*

__eq__(*self:* rti.connextdds.Subscriber, *arg0:* rti.connextdds.Subscriber) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Subscriber, participant: rti.connextdds.DomainParticipant) -> None

Create a subscriber under a DomainParticipant.

2. __init__(self: rti.connextdds.Subscriber, participant: rti.connextdds.DomainParticipant, qos: rti.connextdds.SubscriberQos, listener: rti.connextdds.SubscriberListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a Subscriber under a DomainParticipant with a listener.

3. __init__(self: rti.connextdds.Subscriber, entity: rti.connextdds.IEntity) -> None

Cast an Entity to a Subscriber.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Subscriber, *arg0:* rti.connextdds.Subscriber) → bool

Test for inequality.

property default_datareader_qos

The default DataReaderQos.

This property's getter returns a deep copy.

find_datareader (*self:* rti.connextdds.Subscriber, *name: str*) → Optional[*rti.connextdds.AnyDataReader*]

Find a DataReader in this Subscriber by its name.

find_datareader_by_topic_name (*self:* rti.connextdds.Subscriber, *topic_name: str*) → Optional[*rti.connextdds.AnyDataReader*]

Find a DataReader in this Subscriber by its topic name. If more than one exists for this Topic, the first one found is returned.

find_datareaders (**args*, ***kwargs*)

Overloaded function.

1. find_datareaders(self: rti.connextdds.Subscriber) -> rti.connextdds.AnyDataReaderSeq

Find all DataReaders in the Subscriber.

2. find_datareaders(self: rti.connextdds.Subscriber, arg0: rti.connextdds.DataState) -> rti.connextdds.AnyDataReaderSeq

Find all DataReaders that contain samples of the given DataState in the Subscriber.

3. find_datareaders(self: rti.connextdds.Subscriber, arg0: str) -> rti.connextdds.AnyDataReaderSeq

Find all DataReaders for a given topic name

property listener

Get the listener.

notify_datareaders (*self:* rti.connextdds.Subscriber) → None

This operation invokes the operation on_data_available on the DataReaderListener objects attached to contained DataReader entities with a DATA_AVAILABLE status that is considered changed

property participant

Get the parent DomainParticipant for this Subscriber.

property qos

The SubscriberQos for this Subscriber.

This property's getter returns a deep copy.

set_listener (*self:* rti.connextdds.Subscriber, *listener:* rti.connextdds.SubscriberListener, *event_mask:* rti.connextdds.StatusMask) → None

Bind the listener and event mask to the Subscriber.

class rti.connextdds.**SubscriberListener**

Bases: *AnyDataReaderListener*

__init__ (*self:* rti.connextdds.SubscriberListener) → None

__module__ = 'rti.connextdds'

on_data_on_readers (*self:* rti.connextdds.SubscriberListener, *arg0:* rti.connextdds.Subscriber) → None

Data on datareaders callback.

class rti.connextdds.**SubscriberQos**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.SubscriberQos, *arg0:* rti.connextdds.SubscriberQos) → bool

Test for equality

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SubscriberQos) -> None

Create a SubscriberQos with the default value for each policy.

2. __init__(self: rti.connextdds.SubscriberQos, subscriber: rti.connextdds.Subscriber) -> None

Create a SubscriberQos with settings equivalent to those of the provided Subscriber object.

3. __init__(self: rti.connextdds.SubscriberQos, other: rti.connextdds.SubscriberQos) -> None

Create a copy of a SubscriberQos object.

__lshift__(**args*, ***kwargs*)

Overloaded function.

1. __lshift__(self: rti.connextdds.SubscriberQos, arg0: rti.connextdds.Presentation) -> rti.connextdds.SubscriberQos

Set the PresentationQoS.

2. __lshift__(self: rti.connextdds.SubscriberQos, arg0: rti.connextdds.Partition) -> rti.connextdds.SubscriberQos

Set the PartitionQoS.

3. __lshift__(self: rti.connextdds.SubscriberQos, arg0: rti.connextdds.GroupData) -> rti.connextdds.SubscriberQos

Set the GroupDataQoS.

4. __lshift__(self: rti.connextdds.SubscriberQos, arg0: rti.connextdds.EntityFactory) -> rti.connextdds.SubscriberQos

Set the EntityFactoryQoS.

5. __lshift__(self: rti.connextdds.SubscriberQos, arg0: rti.connextdds.ExclusiveArea) -> rti.connextdds.SubscriberQos

Set the ExclusiveAreaQoS.

6. __lshift__(self: rti.connextdds.SubscriberQos, arg0: rti.connextdds.EntityName) -> rti.connextdds.SubscriberQos

Set the EntityNameQoS.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.SubscriberQos, *arg0:* rti.connextdds.SubscriberQos) → bool

Test for inequality.

__rshift__(**args*, ***kwargs*)

Overloaded function.

1. __rshift__(self: rti.connextdds.SubscriberQos, arg0: rti.connextdds.Presentation) -> rti.connextdds.SubscriberQos

Get the PresentationQoS.

2. __rshift__(self: rti.connextdds.SubscriberQos, arg0: rti.connextdds.Partition) -> rti.connextdds.SubscriberQos

Get the PartitionQoS.

3. __rshift__(self: rti.connextdds.SubscriberQos, arg0: rti.connextdds.GroupData) -> rti.connextdds.SubscriberQos

Get the GroupDataQoS.

4. __rshift__(self: rti.connextdds.SubscriberQos, arg0: rti.connextdds.EntityFactory) -> rti.connextdds.SubscriberQos

Get the EntityFactoryQoS.

5. __rshift__(self: rti.connextdds.SubscriberQos, arg0: rti.connextdds.ExclusiveArea) -> rti.connextdds.SubscriberQos

Get the ExclusiveAreaQoS.

6. __rshift__(self: rti.connextdds.SubscriberQos, arg0: rti.connextdds.EntityName) -> rti.connextdds.SubscriberQos

Get the EntityNameQoS.

__str__ (*self:* rti.connextdds.SubscriberQos) → str

property entity_factory

Get/set EntityFactory QoS.

property entity_name

Get/set EntityName QoS.

property exclusive_area

Get/set ExclusiveArea QoS.

property group_data

Get/set GroupData QoS.

property partition

Get/set Partition QoS.

property presentation

Get/set Presentation QoS.

to_string (*self:* rti.connextdds.SubscriberQos, *format:* rti.connextdds.QosPrintFormat = *QosPrintFormat()*, *base: Optional[*rti.connextdds.SubscriberQos*]* = *None*, *print_all: bool* = *False*) → str

Convert QoS to string based on params.

class rti.connextdds.**SubscriberSeq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.SubscriberSeq, *arg0:* rti.connextdds.SubscriberSeq) → *rti.connextdds.SubscriberSeq*

__bool__ (*self:* rti.connextdds.SubscriberSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.SubscriberSeq, *x:* rti.connextdds.Subscriber) → bool

Return true the container contains x

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.SubscriberSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.SubscriberSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.SubscriberSeq, *arg0:* rti.connextdds.SubscriberSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.SubscriberSeq, s: slice) -> rti.connextdds.SubscriberSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.SubscriberSeq, arg0: int) -> rti.connextdds.Subscriber

__hash__ = None

__iadd__ (*self:* rti.connextdds.SubscriberSeq, *arg0:* rti.connextdds.SubscriberSeq) → *rti.connextdds.SubscriberSeq*

__imul__ (*self:* rti.connextdds.SubscriberSeq, *arg0: int*) → *rti.connextdds.SubscriberSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SubscriberSeq) -> None

2. __init__(self: rti.connextdds.SubscriberSeq, arg0: rti.connextdds.SubscriberSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.SubscriberSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.SubscriberSeq) → Iterator

__len__ (*self:* rti.connextdds.SubscriberSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.SubscriberSeq, *arg0: int*) → *rti.connextdds.SubscriberSeq*

__ne__ (*self:* rti.connextdds.SubscriberSeq, *arg0:* rti.connextdds.SubscriberSeq) → bool

__rmul__ (*self:* rti.connextdds.SubscriberSeq, *arg0: int*) → *rti.connextdds.SubscriberSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.SubscriberSeq, arg0: int, arg1: rti.connextdds.Subscriber) -> None

2. __setitem__(self: rti.connextdds.SubscriberSeq, arg0: slice, arg1: rti.connextdds.SubscriberSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.SubscriberSeq, *x:* rti.connextdds.Subscriber) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.SubscriberSeq) → None

Clear the contents

count (*self:* rti.connextdds.SubscriberSeq, *x:* rti.connextdds.Subscriber) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.SubscriberSeq, L: rti.connextdds.SubscriberSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.SubscriberSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.SubscriberSeq, *i: int*, *x:* rti.connextdds.Subscriber) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.SubscriberSeq) -> rti.connextdds.Subscriber

Remove and return the last item

2. pop(self: rti.connextdds.SubscriberSeq, i: int) -> rti.connextdds.Subscriber

Remove and return the item at index `i`

remove (*self:* rti.connextdds.SubscriberSeq, *x:* rti.connextdds.Subscriber) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**SubscriptionBuiltinTopicData**

Bases: `pybind11_object`

class ContentFilter

Bases: *ContentFilterBase*

__init__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilter) → None

__module__ = 'rti.connextdds'

compile (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilter, *expression: str*, *parameters:* rti.connextdds.StringSeq, *type_code: Optional[*rti.connextdds.DynamicType*]*, *type_class_name: str*, *old_compile_data: Optional[object]*) → Optional[object]

Compile an instance of the content filter according to the filter expression and parameters of the given data type.

evaluate (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilter, *compile_data: Optional[object]*, *sample:* rti.connextdds.SubscriptionBuiltinTopicData, *meta_data:* rti.connextdds.FilterSampleInfo) → bool

Evaluate whether the sample is passing the filter or not according to the sample content.

finalize (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilter, *compile_data: Optional[object]*) → None

A previously compiled instance of the content filter is no longer in use and resources can now be cleaned up.

class ContentFilteredTopic

Bases: *ITopicDescription*, *IAnyTopic*

__eq__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic, topic: rti.connextdds.SubscriptionBuiltinTopicData.Topic, name: str, contentfilter: rti.connextdds.Filter) -> None

Create a ContentFilteredTopic with a name and Filter.

2. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic, topic_description: rti.connextdds.SubscriptionBuiltinTopicData.ITopicDescription) -> None

Cast a TopicDescription to a ContentFilteredTopic.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic) → bool

Test for inequality.

append_to_expression_parameter (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic, *index: int*, *extension: str*) → None

Append the extension to the end of parameter at the provided index, separated by a comma.

property filter_expression

Get the filter expression

property filter_parameters

Get/set the filter parameters.

static find (*participant:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic*]

Look up a ContentFilteredTopic by its name in the DomainParticipant.

remove_from_expression_parameter (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic, *index: int*, *remove_term: str*) → None

Removes the specified term from the parameter at the provided index.

set_filter (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic, *arg0:* rti.connextdds.Filter) → None

Set the filter.

property topic

Get the underlying Topic.

class ContentFilteredTopicSeq

Bases: `pybind11_object`

__add__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq) → *rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq*

__bool__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, s: slice) -> rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, arg0: int) -> rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic

__hash__ = None

__iadd__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq) → *rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq*

__imul__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* *int*) → *rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq) -> None
2. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, arg0: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq) → Iterator

__len__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* *int*) → *rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq*

__ne__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq) → bool

__rmul__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* *int*) → *rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, arg0: int, arg1: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic) -> None
2. __setitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, arg0: slice, arg1: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq) → None

Clear the contents

count (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, L: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, *i: int*, *x:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq) -> rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic

Remove and return the last item

2. pop(self: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, i: int) -> rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic

Remove and return the item at index `i`

remove (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopicSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class DataReader

Bases: *IDataReader*

class Selector

Bases: `pybind11_object`

__init__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector, *datareader:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → None

Create a Selector for a DataReader to read/take based on selected conditions

__module__ = 'rti.connextdds'

condition (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector, *condition:* rti.connextdds.IReadCondition) → *rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector*

Select samples based on a ReadCondition.

content (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector, *query:* rti.connextdds.Query) → *rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector*

Select samples based on a Query.

instance (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector*

Select a specific instance to read/take.

max_samples (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector, *max: int*) → *rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector*

Limit the number of samples read/taken by the Select.

next_instance (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector, *previous:* rti.connextdds.InstanceHandle) → *rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector*

Select the instance after the specified instance to read/take.

read (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector) → list

Read copies of available samples (data and info) based on the Selector settings.

read_data (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector) → list

Read copies of available valid data based on the Selector settings.

read_loaned (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector) → *rti.connextdds.SubscriptionBuiltinTopicData.LoanedSamples*

Take available samples (data and info) based on the Selector settings and return them in a loaned container.

state (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector, *state:* rti.connextdds.DataState) → *rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector*

Select samples with a specified data state.

take (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector) → list

Take copies of available samples (data and info) based on the Selector settings.

take_data (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector) → list

Take copies of available valid data based on the Selector settings.

take_loaned (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader.Selector) → *rti.connextdds.SubscriptionBuiltinTopicData.LoanedSamples*

Read available samples (data and info) based on the Selector settings and return them in a loaned container.

__enter__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → *rti.connextdds.SubscriptionBuiltinTopicData.DataReader*

Enter a context for this DataReader, to be cleaned up on exiting context

__eq__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → bool

Test for equality.

__exit__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for this DataReader, cleaning up resources.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReader, topic: rti.connextdds.SubscriptionBuiltinTopicData.Topic) -> None

Create a DataReader in the implicit subscriber with default QoS.

2. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReader, topic: rti.connextdds.SubscriptionBuiltinTopicData.Topic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader in the implicit subscriber with specific QoS and a listener.

3. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReader, cft: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic) -> None

Create a DataReader with a ContentFilteredTopic in the implicit subscriber with default QoS.

4. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReader, cft: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader with a ContentFilteredTopic in the implicit subscriber with specific QoS.

5. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReader, subscriber: rti.connextdds.Subscriber, topic: rti.connextdds.SubscriptionBuiltinTopicData.Topic) -> None

Create a DataReader.

6. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReader, subscriber: rti.connextdds.Subscriber, topic: rti.connextdds.SubscriptionBuiltinTopicData.Topic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader in a subscriber with specific QoS and a listener.

7. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReader, subscriber: rti.connextdds.Subscriber, cft: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic) -> None

Create a DataReader with a ContentFilteredTopic in a subscriber with default QoS.

8. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReader, subscriber: rti.connextdds.Subscriber, cft: rti.connextdds.SubscriptionBuiltinTopicData.ContentFilteredTopic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.Subscrip-

tionBuiltinTopicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader with a ContentFilteredTopic in a subscriber with specific QoS.

9. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReader, reader: rti.connextdds.IAnyDataReader) -> None

Get a typed DataReader from an AnyDataReader.

10. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReader, entity: rti.connextdds.IEntity) -> None

Get a typed DataReader from an Entity.

__lshift__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *arg0:* rti.connextdds.DataReaderQos) → *rti.connextdds.SubscriptionBuiltinTopicData.DataReader*

Set the DataReaderQos for this DataReader.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → bool

Test for inequality.

__rshift__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *arg0:* rti.connextdds.DataReaderQos) → *rti.connextdds.SubscriptionBuiltinTopicData.DataReader*

Get the DataReaderQos from this DataReader

acknowledge_all(**args*, ***kwargs*)

Overloaded function.

1. acknowledge_all(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReader) -> None

Acknowledge all previously accessed samples.

2. acknowledge_all(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReader, arg0: rti.connextdds.AckResponseData) -> None

Acknowledge all previously accessed samples.

acknowledge_sample(**args*, ***kwargs*)

Overloaded function.

1. acknowledge_sample(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReader, sample_info: rti.connextdds.SampleInfo) -> None

Acknowledge a single sample.

2. acknowledge_sample(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReader, sample_info: rti.connextdds.SampleInfo, ack_response_data: rti.connextdds.AckResponseData) -> None

Acknowledge a single sample with ack response data.

close(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → None

Close this DataReader.

property datareader_cache_status

Get the DataReaderCacheStatus for the DataReader.

property datareader_protocol_status

Get the DataReaderProtocolStatus for the DataReader.

property default_filter_state

Returns the filter state for the read/take operations.

static find_all_by_topic(*subscriber:* rti.connextdds.Subscriber, *topic_name: str*) → *rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq*

Retrieve all DataReaders for the given topic name in the subscriber.

static find_by_name(**args*, ***kwargs*)

Overloaded function.

1. find_by_name(participant: rti.connextdds.DomainParticipant, name: str) -> Optional[rti.connextdds.SubscriptionBuiltinTopicData.DataReader]

Find DataReader in DomainParticipant with the DataReader's name, returning the first found.

2. find_by_name(subscriber: rti.connextdds.Subscriber, name: str) -> Optional[rti.connextdds.SubscriptionBuiltinTopicData.DataReader]

Find DataReader in Subscriber with the DataReader's name, returning the first found.

static find_by_topic(*subscriber:* rti.connextdds.Subscriber, *name: str*) → Optional[*rti.connextdds.SubscriptionBuiltinTopicData.DataReader*]

Find DataReader in Subscriber with a topic name, returning the first found.

is_matched_publication_alive(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *arg0:* rti.connextdds.InstanceHandle) → bool

Check if a matched publication is alive.

key_value(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.SubscriptionBuiltinTopicData*

Retrieve the instance key that corresponds to an instance handle.

property listener

Gets or sets the listener with StatusMask.ALL

property liveliness_changed_status

Get the LivelinessChangedStatus for this DataReader.

lookup_instance(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *key_holder:* rti.connextdds.SubscriptionBuiltinTopicData) → *rti.connextdds.InstanceHandle*

Retrieve the instance handle that corresponds to an instance key_holder

matched_publication_data(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.PublicationBuiltinTopicData*

Get the PublicationBuiltinTopicData for a publication matched to this DataReader.

matched_publication_datareader_protocol_status (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *publication_handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.DataReaderProtocolStatus*

Get the DataReaderProtocolStatus for the DataReader based on the matched publication handle.

matched_publication_participant_data (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ParticipantBuiltinTopicData*

Get the ParticipantBuiltinTopicData for a publication matched to this DataReader.

property matched_publications

Get a copy of the list of the currently matched publication handles.

property qos

The DataReaderQos for this DataReader.

This property's getter returns a deep copy.

read (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → list

Read copies of all available samples (data and info).

read_data (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → list

Read copies of all available valid data.

read_loaned (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → *rti.connextdds.SubscriptionBuiltinTopicData.LoanedSamples*

Read all available samples (data and info) and return them in a loaned container.

property requested_deadline_missed_status

Get the RequestedDeadlineMissed status for the DataReader

property requested_incompatible_qos_status

Get the RequestedIncompatibleQosStatus for the DataReader.

property sample_lost_status

Get the SampleLostStatus for the DataReader.

property sample_rejected_status

Get the SampleRejectedStatus for the DataReader.

select (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → dds::sub::DataReader<dds::topic::TSubscriptionBuiltinTopicData<rti::topic::SubscriptionBuiltinTopicDataImpl>, rti::sub::DataReaderImpl>::Selector

Get a Selector to perform complex data selections, such as per-instance selection, content, and status filtering.

set_listener (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *listener:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and associated event mask.

property subscriber

Returns the parent Subscriber of the DataReader.

property subscription_matched_status

Get the SubscriptionMatchedStatus for the DataReader.

take (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → list

Take copies of all available samples (data and info).

take_data (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → list

Take copies of all available valid data.

take_loaned (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → *rti.connextdds.SubscriptionBuiltinTopicData.LoanedSamples*

Take all available samples (data and info) and return them in a loaned container.

property topic_description

Returns the TopicDescription associated with the DataReader.

property topic_name

Get the topic name associated with this DataReader.

property type_name

Get the type name associated with this DataReader.

wait_for_historical_data (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *max_wait:* rti.connextdds.Duration) → None

Waits until all "historical" data is received for DataReaders that have a non-VOLATILE Durability Qos kind.

wait_for_historical_data_async (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *max_wait:* rti.connextdds.Duration) → object

Waits until all "historical" data is received for DataReaders that have a non-VOLATILE Durability Qos kind. This call is awaitable and only for use with asyncio.

class DataReaderListener

Bases: `pybind11_object`

__init__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderListener) → None

__module__ = 'rti.connextdds'

on_data_available(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → None

Data available callback.

on_liveliness_changed(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *arg1:* rti.connextdds.LivelinessChangedStatus) → None

Liveliness changed callback.

on_requested_deadline_missed(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *arg1:* rti.connextdds.RequestedDeadlineMissedStatus) → None

Requested deadline missed callback.

on_requested_incompatible_qos(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *arg1:* rti.connextdds.RequestedIncompatibleQosStatus) → None

Requested incompatible QoS callback.

on_sample_lost(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SampleLostStatus) → None

Sample lost callback.

on_sample_rejected(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SampleRejectedStatus) → None

Sample rejected callback.

on_subscription_matched(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SubscriptionMatchedStatus) → None

Subscription matched callback.

class DataReaderSeq

Bases: pybind11_object

__add__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq) → *rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq*

__bool__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, s: slice) -> rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, arg0: int) -> rti.connextdds.SubscriptionBuiltinTopicData.DataReader

__hash__ = None

__iadd__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq) → *rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq*

__imul__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, *arg0: int*) → *rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq) -> None
2. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, arg0: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq) → Iterator

__len__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, *arg0: int*) → *rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq*

__ne__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq) → bool

__rmul__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, *arg0: int*) → *rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, arg0: int, arg1: rti.connextdds.SubscriptionBuiltinTopicData.DataReader) -> None
2. __setitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, arg0: slice, arg1: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq) → None

Clear the contents

count (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, L: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, *i: int*, *x:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq) -> rti.connextdds.SubscriptionBuiltinTopicData.DataReader

Remove and return the last item

2. pop(self: rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, i: int) -> rti.connextdds.SubscriptionBuiltinTopicData.DataReader

Remove and return the item at index `i`

remove(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataReaderSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class DataWriter

Bases: *IEntity*, *IAnyDataWriter*

__enter__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter) → *rti.connextdds.SubscriptionBuiltinTopicData.DataWriter*

Enter a context for this DataWriter, to be cleaned up on exiting context

__eq__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter) → bool

Test for equality.

__exit__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for this DataWriter, cleaning up resources.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, topic: rti.connextdds.SubscriptionBuiltinTopicData.Topic) -> None

Creates a DataWriter in the implicit publisher with default QoS.

2. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, topic: rti.connextdds.SubscriptionBuiltinTopicData.Topic, qos: rti.connextdds.DataWriterQos, listener: rti.connextdds.SubscriptionBuiltinTopicData.DataWriterListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Creates a DataWriter in the implicit publisher with specific QoS and optionally a listener.

3. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, pub: rti.connextdds.Publisher, topic: rti.connextdds.SubscriptionBuiltinTopicData.Topic) -> None

Creates a DataWriter in a publisher with default QoS.

4. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, pub: rti.connextdds.Publisher, topic: rti.connextdds.SubscriptionBuiltinTopicData.Topic, qos: rti.connextdds.DataWriterQos, listener: rti.connextdds.SubscriptionBuiltinTopicData.DataWriterListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Creates a DataWriter in a publisher with specific QoS and optionally a listener.

5. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, writer: rti.connextdds.IAnyDataWriter) -> None

Create a typed DataWriter from an AnyDataWriter.

6. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, entity: rti.connextdds.IEntity) -> None

Create a typed DataWriter from an Entity.

__lshift__(**args*, ***kwargs*)

Overloaded function.

1. __lshift__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, arg0: rti.connextdds.DataWriterQos) -> rti.connextdds.SubscriptionBuiltinTopicData.DataWriter

Sets the DataWriterQos.

2. __lshift__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, arg0: Tuple[rti.connextdds.SubscriptionBuiltinTopicData, rti.connextdds.Time]) -> rti.connextdds.SubscriptionBuiltinTopicData.DataWriter

Writes a paired sample with a timestamp.

3. __lshift__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, arg0: Tuple[rti.connextdds.SubscriptionBuiltinTopicData, rti.connextdds.InstanceHandle]) -> rti.connextdds.SubscriptionBuiltinTopicData.DataWriter

Writes a paired sample with an instance handle.

4. __lshift__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, arg0: rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq) -> rti.connextdds.SubscriptionBuiltinTopicData.DataWriter

Writes a sequence of pairs of samples with timestamps.

5. __lshift__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, arg0: rti.connextdds.SubscriptionBuiltinTopicDataSeq) -> rti.connextdds.SubscriptionBuiltinTopicData.DataWriter

Writes a sequence of samples.

6. __lshift__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, arg0: rti.connextdds.SubscriptionBuiltinTopicData) -> rti.connextdds.SubscriptionBuiltinTopicData.DataWriter

Writes a sample.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter) → bool

Test for inequality.

__rshift__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg0:* rti.connextdds.DataWriterQos) → *rti.connextdds.SubscriptionBuiltinTopicData.DataWriter*

Get the DataWriterQos.

assert_liveliness(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter) → None

Manually asserts the liveliness of the DataWriter.

close(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter) → None

Close this DataWriter.

create_data(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter) → *rti.connextdds.SubscriptionBuiltinTopicData*

Create data of the writer's associated type and initialize it.

property datawriter_cache_status

Get a copy of the cache status for this writer.

property datawriter_protocol_status

Get a copy of the protocol status for this writer.

dispose_instance(**args*, ***kwargs*)

Overloaded function.

1. dispose_instance(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.SubscriptionBuiltinTopicData.DataWriter

Dispose an instance.

2. dispose_instance(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> rti.connextdds.SubscriptionBuiltinTopicData.DataWriter

Dispose an instance with a timestamp.

3. dispose_instance(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, params: rti.connextdds.WriteParams) -> None

Dispose an instance with params.

dispose_instance_async(**args*, ***kwargs*)

Overloaded function.

1. dispose_instance_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> object

Dispose an instance.

2. dispose_instance_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> object

Dispose an instance with a timestamp.

3. dispose_instance_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, key_holder: rti.connextdds.SubscriptionBuiltinTopicData) -> object

Dispose the instance associated with key_holder.

4. dispose_instance_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, key_holder: rti.connextdds.SubscriptionBuiltinTopicData, timestamp: rti.connextdds.Time) -> object

Dispose the instance associated with key_holder using a timestamp

5. dispose_instance_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, params: rti.connextdds.WriteParams) -> object

Dispose an instance with params.

static find_all_by_topic(*publisher:* rti.connextdds.Publisher, *topic_name: str*) → *rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq*

Retrieve all DataWriters for the given topic name in the publisher.

static find_by_name(**args*, ***kwargs*)

Overloaded function.

1. find_by_name(participant: rti.connextdds.DomainParticipant, name: str) -> Optional[rti.connextdds.SubscriptionBuiltinTopicData.DataWriter]

Find DataWriter in DomainParticipant with the provided name, returning the first found.

2. find_by_name(publisher: rti.connextdds.Publisher, name: str) -> Optional[rti.connextdds.SubscriptionBuiltinTopicData.DataWriter]

Find DataWriter in Publisher with the DataReader's name, returning the first found.

static find_by_topic(*publisher:* rti.connextdds.Publisher, *name: str*) → Optional[*rti.connextdds.SubscriptionBuiltinTopicData.DataWriter*]

Find DataWriter in publisher with a topic name, returning the first found.

flush(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter) → None

Flushes the batch in progress in the context of thecalling thread.

is_matched_subscription_active(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg0:* rti.connextdds.InstanceHandle) → bool

A boolean indicating whether or not the matched subscription is active.

is_sample_app_acknowledged(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *sample_id:* rti.connextdds.SampleIdentity) → bool

Indicates if a sample is considered application-acknowledged.

key_value(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.SubscriptionBuiltinTopicData*

Retrieve the instance key that corresponds to an instance handle.

property listener

Get the listener associated with the DataWriter or set the listener.

property liveliness_lost_status

Get a copy of the LivelinessLostStatus.

lookup_instance(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *key_holder:* rti.connextdds.SubscriptionBuiltinTopicData) → *rti.connextdds.InstanceHandle*

Retrieve the instance handle that corresponds to an instance key_holder

matched_subscription_data(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.SubscriptionBuiltinTopicData*

Get the SubscriptionBuiltinTopicData for a subscription matched to this DataWriter.

matched_subscription_datawriter_protocol_status(**args*, ***kwargs*)

Overloaded function.

1. matched_subscription_datawriter_protocol_status(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.DataWriterProtocolStatus

Get a copy of the protocol status for this writer per a matched subscription handle.

2. matched_subscription_datawriter_protocol_status(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, locator: rti.connextdds.Locator) -> rti.connextdds.DataWriterProtocolStatus

Get a copy of the protocol status for this writer per a matched subscription locator.

matched_subscription_participant_data (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ParticipantBuiltinTopicData*

Get the ParticipantBuiltinTopicData for a subscription matched to this DataWriter.

property matched_subscriptions

Get a copy of the list of the currently matched subscription handles.

property matched_subscriptions_locators

The locators used to communicate with matched DataReaders.

property offered_deadline_missed_status

Get a copy of the OfferedDeadlineMissedStatus.

property offered_incompatible_qos_status

Get a copy of the OfferedIncompatibleQosStatus

property publication_matched_status

Get a copy of the PublicationMatchedStatus

property publisher

Get the Publisher that owns this DataWriter.

property qos

The DataWriterQos for this DataWriter.This property's getter returns a deep copy.

register_instance (**args*, ***kwargs*)

Overloaded function.

1. register_instance(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, key_holder: rti.connextdds.SubscriptionBuiltinTopicData) -> rti.connextdds.InstanceHandle

Informs RTI Connext that the application will be modifying a particular instance.

2. register_instance(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, key_holder: rti.connextdds.SubscriptionBuiltinTopicData, timestamp: rti.connextdds.Time) -> rti.connextdds.InstanceHandle

Informs RTI Connext that the application will be modifying a particular instance and specified the timestamp.

3. register_instance(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, key_holder: rti.connextdds.SubscriptionBuiltinTopicData, params: rti.connextdds.WriteParams) -> rti.connextdds.InstanceHandle

Registers instance with parameters.

property reliable_reader_activity_changed_status

Get a copy of the reliable reader activity changed status for this writer.

property reliable_writer_cache_changed_status

Get a copy of the reliable cache status for this writer.

property service_request_accepted_status

Get a copy of the service request accepted status for this writer.

set_listener(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *listener:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and event mask for the DataWriter.

property topic

Get the Topic object associated with this DataWriter.

property topic_name

Get the topic name associated with this DataWriter.

property type_name

Get the type name for the topic object associated with this DataWriter.

unregister_instance(**args*, ***kwargs*)

Overloaded function.

1. unregister_instance(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.SubscriptionBuiltinTopicData.DataWriter

Unregister an instance.

2. unregister_instance(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> rti.connextdds.SubscriptionBuiltinTopicData.DataWriter

Unregister an instance with timestamp.

3. unregister_instance(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, params: rti.connextdds.WriteParams) -> None

Unregister an instance with parameters.

unregister_instance_async(**args*, ***kwargs*)

Overloaded function.

1. unregister_instance_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> object

Unregister an instance.

2. unregister_instance_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> object

Unregister an instance with timestamp.

3. unregister_instance_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, key_holder: rti.connextdds.SubscriptionBuiltinTopicData) -> object

Unregister the instance associated with key_holder.

4. unregister_instance_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, key_holder: rti.connextdds.SubscriptionBuiltinTopicData, timestamp: rti.connextdds.Time) -> object

Unregister the instance associate with key_holder using a timestamp.

5. unregister_instance_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, params: rti.connextdds.WriteParams) -> object

Unregister an instance with parameters.

wait_for_acknowledgments (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *max_wait:* rti.connextdds.Duration) → None

Blocks the calling thread until all data written by a reliable DataWriter is acknowledged or until the timeout expires.

wait_for_asynchronous_publishing (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *max_wait:* rti.connextdds.Duration) → None

This operation blocks the calling thread (up to max_wait) until all data written by the asynchronous DataWriter is sent and acknowledged (if reliable) by all matched DataReader entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; a time out indicates that max_wait elapsed before all the data was sent and/or acknowledged.

In other words, this guarantees that sending to best effort DataReader is complete in addition to what DataWriter.wait_for_acknowledgments() provides.

If the DataWriter does not have PublishMode kind set to PublishModeKind.ASYNCHRONOUS the operation will complete immediately

wait_for_asynchronous_publishing_async (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *max_wait:* rti.connextdds.Duration) → object

This function is awaitable until either a timeout of max_wait or all data written by the asynchronous DataWriter is sent and acknowledged (if reliable) by all matched DataReader entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; a time out indicates that max_wait elapsed before all the data was sent and/or acknowledged.This function works with asyncio.

In other words, this guarantees that sending to best effort DataReader is complete in addition to what DataWriter.wait_for_acknowledgments() provides.

If the DataWriter does not have PublishMode kind set to PublishModeKind.ASYNCHRONOUS the operation will complete immediately

write (**args*, ***kwargs*)

Overloaded function.

1. write(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, samples: rti.connextdds.SubscriptionBuiltinTopicDataSeq) -> None

Write a sequence of samples.

2. write(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, samples: rti.connextdds.SubscriptionBuiltinTopicDataSeq, timestamp: rti.connextdds.Time) -> None

Write a sequence of samples with a timestamp.

3. write(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, sample: rti.connextdds.SubscriptionBuiltinTopicData) -> None

Write a sample.

4. write(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, sample: rti.connextdds.SubscriptionBuiltinTopicData, timestamp: rti.connextdds.Time) -> None

Write a sample with a specified timestamp.

5. write(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, sample: rti.connextdds.SubscriptionBuiltinTopicData, handle: rti.connextdds.InstanceHandle) -> None

Write a sample with an instance handle.

6. write(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, sample: rti.connextdds.SubscriptionBuiltinTopicData, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> None

Write a sample with an instance handle and specified timestamp.

7. write(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, sample: rti.connextdds.SubscriptionBuiltinTopicData, params: rti.connextdds.WriteParams) -> None

Write with advanced parameters.

write_async(**args*, ***kwargs*)

Overloaded function.

1. write_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, sample: rti.connextdds.SubscriptionBuiltinTopicData) -> object

Write a sample. This method is awaitable and is only for use with asyncio.

2. write_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, sample: rti.connextdds.SubscriptionBuiltinTopicData, timestamp: rti.connextdds.Time) -> object

Write a sample with a specified timestamp. This methods is awaitable and only for use with asyncio.

3. write_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, sample: rti.connextdds.SubscriptionBuiltinTopicData, handle: rti.connextdds.InstanceHandle) -> object

Write a sample with an instance handle. This method is awaitable and only for use with asyncio.

4. write_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, sample: rti.connextdds.SubscriptionBuiltinTopicData, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> object

Write a sample with an instance handle and specified timestamp. This method is awaitable and only for use with asyncio.

5. write_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, samples: rti.connextdds.SubscriptionBuiltinTopicDataSeq) -> object

Write a sequence of samples. This method is awaitable and only for use with asyncio.

6. write_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, samples: rti.connextdds.SubscriptionBuiltinTopicDataSeq, timestamp: rti.connextdds.Time) -> object

Write a sequence of samples with a timestamp. This method is awaitable and only for use with asyncio.

7. write_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, samples: rti.connextdds.SubscriptionBuiltinTopicDataSeq, handles: rti.connextdds.InstanceHandleSeq) -> object

Write a sequence of samples with their instance handles. This method is awaitable and only for use with asyncio.

8. write_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, samples: rti.connextdds.SubscriptionBuiltinTopicDataSeq, handles: rti.connextdds.InstanceHandleSeq, timestamp: rti.connextdds.Time) -> object

Write a sequence of samples with their instance handles and a timestamp. This method is awaitable and only for use with asyncio.

9. write_async(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, sample: rti.connextdds.SubscriptionBuiltinTopicData, params: rti.connextdds.WriteParams) -> object

Write with advanced parameters.

class DataWriterListener

Bases: pybind11_object

__init__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterListener) → None

__module__ = 'rti.connextdds'

on_application_acknowledgment (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.AcknowledgmentInfo) → None

On application acknowledgment callback

on_instance_replaced (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.InstanceHandle) → None

On instance replaced callback.

on_liveliness_lost (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.LivelinessLostStatus) → None

Liveliness lost callback.

on_offered_deadline_missed (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.OfferedDeadlineMissedStatus) → None

Offered deadline missed callback.

on_offered_incompatible_qos (*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.OfferedIncompatibleQosStatus) → None

Offered incompatible QoS callback.

on_publication_matched(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.PublicationMatchedStatus) → None

Publication matched callback.

on_reliable_reader_activity_changed(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ReliableReaderActivityChangedStatus) → None

Reliable reader activity changed callback.

on_reliable_writer_cache_changed(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ReliableWriterCacheChangedStatus) → None

Reliable writer cache changed callback.

on_service_request_accepted(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ServiceRequestAcceptedStatus) → None

On service request accepted callback.

class DataWriterSeq

Bases: `pybind11_object`

__add__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq) → *rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq*

__bool__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, s: slice) -> rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, arg0: int) -> rti.connextdds.SubscriptionBuiltinTopicData.DataWriter

__hash__ = None

__iadd__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq) → *rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq*

__imul__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, *arg0: int*) → *rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq) -> None
2. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, arg0: rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq) → Iterator

__len__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, *arg0: int*) → *rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq*

__ne__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq) → bool

__rmul__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, *arg0: int*) → *rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, arg0: int, arg1: rti.connextdds.SubscriptionBuiltinTopicData.DataWriter) -> None
2. __setitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, arg0: slice, arg1: rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq) -> None

Assign list elements using a slice object

append(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter) → None

Add an item to the end of the list

clear(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq) → None

Clear the contents

count(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter) → int

Return the number of times `x` appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, L: rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, *i: int*, *x:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter) → None

Insert an item at a given position.

pop(**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq) -> rti.connextdds.SubscriptionBuiltinTopicData.DataWriter

Remove and return the last item

2. pop(self: rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, i: int) -> rti.connextdds.SubscriptionBuiltinTopicData.DataWriter

Remove and return the item at index `i`

remove(*self:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriterSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class ITopicDescription

Bases: *IEntity*

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property name

The name of the entity conforming to the ITopicDescription interface.

property participant

The parent DomainParticipant.

property type_name

The name of the associated type.

class LoanedSample

Bases: pybind11_object

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.LoanedSample) -> None

Basic constructor

2. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.LoanedSample, data: rti.connextdds.SubscriptionBuiltinTopicData, info: rti.connextdds.SampleInfo) -> None

Construct LoanedSample with data and info.

__iter__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.LoanedSample) → object

__module__ = 'rti.connextdds'

property data

Get the data associated with the sample.

property info

Get the info associated with the sample.

class LoanedSamples

Bases: pybind11_object

__enter__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.LoanedSamples) → *rti.connextdds.SubscriptionBuiltinTopicData.LoanedSamples*

Enter a context for the loaned samples, loan returned on context exit.

__exit__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.LoanedSamples, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for the loaned samples, returning the resources.

__getitem__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.LoanedSamples, *arg0: int*) → *rti.connextdds.SubscriptionBuiltinTopicData.LoanedSample*

Access a LoanedSample object in an array-like syntax

__init__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.LoanedSamples) → None

Create an empty LoanedSamples object.

__iter__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.LoanedSamples) → Iterator

__len__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.LoanedSamples) → int

Get the number of samples in the loan.

__module__ = 'rti.connextdds'

property length

Get the number of samples in the loan.

return_loan(*self:* rti.connextdds.SubscriptionBuiltinTopicData.LoanedSamples) → None

Returns the loan to the DataReader.

class NoOpDataReaderListener

Bases: *DataReaderListener*

__init__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpDataReaderListener) → None

__module__ = 'rti.connextdds'

on_data_available(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader) → None

Data available callback.

on_liveliness_changed(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *arg1:* rti.connextdds.LivelinessChangedStatus) → None

Liveliness changed callback.

on_requested_deadline_missed(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *arg1:* rti.connextdds.RequestedDeadlineMissedStatus) → None

Requested deadline missed callback.

on_requested_incompatible_qos(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *arg1:* rti.connextdds.RequestedIncompatibleQosStatus) → None

Requested incompatible QoS callback.

on_sample_lost(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SampleLostStatus) → None

Sample lost callback.

on_sample_rejected(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SampleRejectedStatus) → None

Sample rejected callback.

on_subscription_matched(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SubscriptionMatchedStatus) → None

Subscription matched callback.

class NoOpDataWriterListener

Bases: *DataWriterListener*

__init__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpDataWriterListener) → None

__module__ = 'rti.connextdds'

on_application_acknowledgment(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.AcknowledgmentInfo) → None

On application acknowledgment callback

on_instance_replaced(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.InstanceHandle) → None

On instance replaced callback.

on_liveliness_lost(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.LivelinessLostStatus) → None

Liveliness lost callback.

on_offered_deadline_missed(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.OfferedDeadlineMissedStatus) → None

Offered deadline missed callback.

on_offered_incompatible_qos(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.OfferedIncompatibleQosStatus) → None

Offered incompatible QoS callback.

on_publication_matched(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.PublicationMatchedStatus) → None

Publication matched callback.

on_reliable_reader_activity_changed(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ReliableReaderActivityChangedStatus) → None

Reliable reader activity changed callback.

on_reliable_writer_cache_changed(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ReliableWriterCacheChangedStatus) → None

Reliable writer cache changed callback.

on_service_request_accepted(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ServiceRequestAcceptedStatus) → None

On service request accepted callback.

class NoOpTopicListener

Bases: *TopicListener*

__init__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpTopicListener) → None

__module__ = 'rti.connextdds'

on_inconsistent_topic(*self:* rti.connextdds.SubscriptionBuiltinTopicData.NoOpTopicListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.Topic, *arg1:* rti.connextdds.InconsistentTopicStatus) → None

Inconsistent topic callback.

class Sample

Bases: pybind11_object

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.Sample, data: rti.connextdds.SubscriptionBuiltinTopicData, info: rti.connextdds.SampleInfo) -> None

Construct Sample with data and info.

2. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.Sample, sample: rti.connextdds.SubscriptionBuiltinTopicData.Sample) -> None

Copy constructor.

3. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.Sample) -> None

Basic constructor

4. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.Sample, loaned_sample: rti.connextdds.SubscriptionBuiltinTopicData.LoanedSample) -> None

Construct a sample with a loaned sample.

__iter__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.Sample) → object

__module__ = 'rti.connextdds'

property data

The data associated with the sample.

property info

Get the info associated with the sample.

class SharedSamples

Bases: pybind11_object

__getitem__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.SharedSamples, *arg0: int*) → *rti.connextdds.SubscriptionBuiltinTopicData.LoanedSample*

Get the sample at the specified index.

__init__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.SharedSamples, *loaned_samples:* rti.connextdds.SubscriptionBuiltinTopicData.LoanedSamples) → None

Constructs an instance of SharedSamples, removing ownership of the loan from the Loaned Samples.

__iter__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.SharedSamples) → Iterator

Make a sample iterator

__len__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.SharedSamples) → int

Returns the number of samples.

__module__ = 'rti.connextdds'

class Topic

Bases: *ITopicDescription*, *IAnyTopic*

__eq__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.Topic, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.Topic) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.Topic, entity: rti.connextdds.IEntity) -> None

Cast an Entity to a Topic.

2. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.Topic, topic_description: rti.connextdds.SubscriptionBuiltinTopicData.ITopicDescription) -> None

Cast an ITopicDescription to a Topic.

3. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.Topic, topic: rti.connextdds.IAnyTopic) -> None

Create a typed Topic from an AnyTopic.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.Topic, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.Topic) → bool

Test for inequality.

static find(*participant:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.SubscriptionBuiltinTopicData.Topic*]

Look up a Topic by its name in the DomainParticipant.

property inconsistent_topic_status

Get a copy of the current InconsistentTopicStatus for this Topic.

property listener

The listener.

property qos

Get the TopicQos for this Topic.

This property's getter returns a deep copy.

set_listener(*self:* rti.connextdds.SubscriptionBuiltinTopicData.Topic, *listener:* rti.connextdds.SubscriptionBuiltinTopicData.TopicListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and event mask.

class TopicDescription

Bases: *ITopicDescription*

__eq__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicDescription, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.TopicDescription) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.TopicDescription, topic: rti.connextdds.SubscriptionBuiltinTopicData.ITopicDescription) -> None

Cast a Topic to a TopicDescription.

2. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.TopicDescription, entity: rti.connextdds.IEntity) -> None

Cast a Topic to a TopicDescription.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicDescription, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.TopicDescription) → bool

Test for inequality.

class TopicListener

Bases: pybind11_object

__init__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicListener) → None

__module__ = 'rti.connextdds'

on_inconsistent_topic(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicListener, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.Topic, *arg1:* rti.connextdds.InconsistentTopicStatus) → None

Inconsistent topic callback.

class TopicSeq

Bases: pybind11_object

__add__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq) → *rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq*

__bool__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData.Topic) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, s: slice) -> rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, arg0: int) -> rti.connextdds.SubscriptionBuiltinTopicData.Topic

__hash__ = None

__iadd__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq) → *rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq*

__imul__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, *arg0: int*) → *rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq) -> None
2. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, arg0: rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq) → Iterator

__len__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, *arg0: int*) → *rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq*

__ne__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq) → bool

__rmul__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, *arg0: int*) → *rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, arg0: int, arg1: rti.connextdds.SubscriptionBuiltinTopicData.Topic) -> None
2. __setitem__(self: rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, arg0: slice, arg1: rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq) -> None

Assign list elements using a slice object

append(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData.Topic) → None

Add an item to the end of the list

clear(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq) → None

Clear the contents

count(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData.Topic) → int

Return the number of times `x` appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, L: rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, *i: int*, *x:* rti.connextdds.SubscriptionBuiltinTopicData.Topic) → None

Insert an item at a given position.

pop(**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq) -> rti.connextdds.SubscriptionBuiltinTopicData.Topic

Remove and return the last item

2. pop(self: rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, i: int) -> rti.connextdds.SubscriptionBuiltinTopicData.Topic

Remove and return the item at index `i`

remove(*self:* rti.connextdds.SubscriptionBuiltinTopicData.TopicSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData.Topic) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class ValidLoanedSamples

Bases: `pybind11_object`

__enter__(*self:* rti.connextdds.SubscriptionBuiltinTopicData.ValidLoanedSamples) → *rti.connextdds.SubscriptionBuiltinTopicData.ValidLoanedSamples*

__exit__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ValidLoanedSamples, *arg0: object*, *arg1: object*, *arg2: object*) → None

__init__ (**args*, ***kwargs*)

__iter__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.ValidLoanedSamples) → Iterator

__module__ = 'rti.connextdds'

class WriterContentFilter

Bases: *ContentFilter*

__init__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.WriterContentFilter) → None

__module__ = 'rti.connextdds'

writer_attach (*self:* rti.connextdds.SubscriptionBuiltinTopicData.WriterContentFilter) → Optional[object]

A writer-side filtering API to create some state that can facilitate filtering on the writer side.

writer_compile (*self:* rti.connextdds.SubscriptionBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *property:* rti.connextdds.ExpressionProperty, *expression: str*, *parameters:* rti.connextdds.StringSeq, *type_code: Optional[*rti.connextdds.DynamicType*]*, *type_class_name: str*, *cookie:* rti.connextdds.Cookie) → None

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching DataReader.

writer_detach (*self:* rti.connextdds.SubscriptionBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*) → None

A writer-side filtering API to clean up a previously created state using writer_attach.

writer_evaluate (*self:* rti.connextdds.SubscriptionBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *sample:* rti.connextdds.SubscriptionBuiltinTopicData, *meta_data:* rti.connextdds.FilterSampleInfo) → *rti.connextdds.CookieVector*

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching DataReader.

writer_finalize (*self:* rti.connextdds.SubscriptionBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *cookie:* rti.connextdds.Cookie) → None

A writer-side filtering API to clean up a previously compiled instance of the content filter.

writer_return_loan (*self:* rti.connextdds.SubscriptionBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *cookies:* rti.connextdds.CookieVector) → None

A writer-side filtering API to return the loan on the list of DataReaders returned by writer_evaluate.

class WriterContentFilterHelper

Bases: *WriterContentFilter*

__init__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData.WriterContentFilterHelper) → None

__module__ = 'rti.connextdds'

add_cookie (*self:* rti.connextdds.SubscriptionBuiltinTopicData.WriterContentFilterHelper, *cookie:* rti.connextdds.Cookie) → None

A helper function which will add a Cookie to the Cookie sequence that is then returned by the writer_evaluate function.

writer_evaluate_helper (*self:* rti.connextdds.SubscriptionBuiltinTopicData.WriterContentFilterHelper, *writer_filter_data: Optional[object]*, *sample:* rti.connextdds.SubscriptionBuiltinTopicData, *meta_data:* rti.connextdds.FilterSampleInfo) → None

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching DataReader.

__eq__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData) → None

Create a default SubscriptionBuiltinTopicData.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.SubscriptionBuiltinTopicData, *arg0:* rti.connextdds.SubscriptionBuiltinTopicData) → bool

Test for inequality.

builtin_topic_name = 'DCPSSubscription'

property content_filter_property

The custom multicast locators that the endpoint can specify.

property data_tag

The DataTag policy of the corresponding DataReader.

property deadline

The Deadline policy of the corresponding DataReader.

property destination_order

The DestinationOrder policy of the corresponding DataReader.

property disable_positive_acks

See whether or not a matching DataReader will send positive acknowledgments for reliability.

property durability

The Durability policy of the corresponding DataReader.

property group_data

The GroupData policy of the corresponding DataReader's Subscriber.

property key

The DCPS key to distinguish entries.

property latency_budget

The LatencyBudget policy of the corresponding DataReader.

property liveliness

The Liveliness policy of the corresponding DataReader.

property multicast_locators

The custom multicast locators that the endpoint can specify.

property ownership

The Ownership policy of the corresponding DataReader.

property participant_key

The DCPS key of the DomainParticipant to which the DataReader belongs.

property partition

The Partition policy of the corresponding DataReader's Subscriber.

property presentation

The Presentation policy of the corresponding DataReader's Subscriber.

property product_version

The current version for RTI Connext.

property property

The propagated name-value properties of the corresponding DataReader.

property reliability

The Reliability policy of the corresponding DataReader.

property representation

The Representation policy of the corresponding DataReader.

property rtps_protocol_version

The version number of the RTPS wire protocol used.

property rtps_vendor_id

The ID of the vendor implementing the RTPS wire protocol.

property service

The Service policy

property subscriber_key

The DCPS key of the Publisher to which the DataReader belongs.

property subscription_name

The subscription name and role name.

property time_based_filter

The TimeBasedFilter policy of the corresponding DataReader.

property topic_data

The TopicData policy of the corresponding DataReader's Topic.

property topic_name

The name of the related Topic.

property type

The type.

property type_name

The name of the type attached to the Topic.

property unicast_locators

The custom unicast locators that the endpoint can specify.

property user_data

The UserData policy of the corresponding DataReader.

property virtual_guid

The virtual Guid associated to the DataReader.

class rti.connextdds.**SubscriptionBuiltinTopicDataSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.SubscriptionBuiltinTopicDataSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicDataSeq) → *rti.connextdds.SubscriptionBuiltinTopicDataSeq*

__bool__(*self:* rti.connextdds.SubscriptionBuiltinTopicDataSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.SubscriptionBuiltinTopicDataSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.SubscriptionBuiltinTopicDataSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.SubscriptionBuiltinTopicDataSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicDataSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.SubscriptionBuiltinTopicDataSeq, s: slice) -> rti.connextdds.SubscriptionBuiltinTopicDataSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.SubscriptionBuiltinTopicDataSeq, arg0: int) -> rti.connextdds.SubscriptionBuiltinTopicData

__hash__ = None

__iadd__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicDataSeq) → *rti.connextdds.SubscriptionBuiltinTopicDataSeq*

__imul__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataSeq, *arg0: int*) → *rti.connextdds.SubscriptionBuiltinTopicDataSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SubscriptionBuiltinTopicDataSeq) -> None

2. __init__(self: rti.connextdds.SubscriptionBuiltinTopicDataSeq, arg0: rti.connextdds.SubscriptionBuiltinTopicDataSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.SubscriptionBuiltinTopicDataSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataSeq) → Iterator

__len__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataSeq, *arg0: int*) → *rti.connextdds.SubscriptionBuiltinTopicDataSeq*

__ne__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicDataSeq) → bool

__rmul__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataSeq, *arg0: int*) → *rti.connextdds.SubscriptionBuiltinTopicDataSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.SubscriptionBuiltinTopicDataSeq, arg0: int, arg1: rti.connextdds.SubscriptionBuiltinTopicData) -> None

2. __setitem__(self: rti.connextdds.SubscriptionBuiltinTopicDataSeq, arg0: slice, arg1: rti.connextdds.SubscriptionBuiltinTopicDataSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.SubscriptionBuiltinTopicDataSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.SubscriptionBuiltinTopicDataSeq) → None

Clear the contents

count (*self:* rti.connextdds.SubscriptionBuiltinTopicDataSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.SubscriptionBuiltinTopicDataSeq, L: rti.connextdds.SubscriptionBuiltinTopicDataSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.SubscriptionBuiltinTopicDataSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.SubscriptionBuiltinTopicDataSeq, *i: int*, *x:* rti.connextdds.SubscriptionBuiltinTopicData) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.SubscriptionBuiltinTopicDataSeq) -> rti.connextdds.SubscriptionBuiltinTopicData

Remove and return the last item

2. pop(self: rti.connextdds.SubscriptionBuiltinTopicDataSeq, i: int) -> rti.connextdds.SubscriptionBuiltinTopicData

Remove and return the item at index i

remove (*self:* rti.connextdds.SubscriptionBuiltinTopicDataSeq, *x:* rti.connextdds.SubscriptionBuiltinTopicData) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**SubscriptionBuiltinTopicDataTimestampedSeq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq) → *rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq*

__bool__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, *x:* *Tuple[*rti.connextdds.SubscriptionBuiltinTopicData*,* rti.connextdds.Time*]*) → bool

Return true the container contains x

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, arg0: int) -> None

Delete the list elements at index i

2. __delitem__(self: rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, s: slice) -> rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, arg0: int) -> Tuple[rti.connextdds.SubscriptionBuiltinTopicData, rti.connextdds.Time]

__hash__ = None

__iadd__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq) → *rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq*

__imul__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, *arg0: int*) → *rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq) -> None
2. __init__(self: rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, arg0: rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq) → Iterator

__len__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, *arg0: int*) → *rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq*

__ne__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, *arg0:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__rmul__ (*self:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, *arg0: int*) → *rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, arg0: int, arg1: Tuple[rti.connextdds.SubscriptionBuiltinTopicData, rti.connextdds.Time]) -> None
2. __setitem__(self: rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, arg0: slice, arg1: rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, *x: Tuple[*rti.connextdds.SubscriptionBuiltinTopicData*,* rti.connextdds.Time*]*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq) → None

Clear the contents

count (*self:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, *x:* *Tuple[*rti.connextdds.SubscriptionBuiltinTopicData, rti.connextdds.Time*]*) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, L: rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, *i: int*, *x:* *Tuple[*rti.connextdds.SubscriptionBuiltinTopicData, rti.connextdds.Time*]*) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq) -> Tuple[rti.connextdds.SubscriptionBuiltinTopicData, rti.connextdds.Time]

Remove and return the last item

2. pop(self: rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, i: int) -> Tuple[rti.connextdds.SubscriptionBuiltinTopicData, rti.connextdds.Time]

Remove and return the item at index `i`

remove (*self:* rti.connextdds.SubscriptionBuiltinTopicDataTimestampedSeq, *x:* *Tuple[*rti.connextdds.SubscriptionBuiltinTopicData, rti.connextdds.Time*]*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**SubscriptionMatchedStatus**

Bases: `pybind11_object`

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property current_count

The number of DataWriters that are currently matched with this DataReader.

property current_count_change

The delta for the current count since the last time the listener fired or the status was read.

property current_count_peak

The highest value that the current count has reached.

property last_publication_handle

A handle to the DataWriter that caused the most recent change to to this status.

property total_count

Total count of times the DataReader matched with a DataWriter.

property total_count_change

The delta in the total count since the last time the listener fired or the status was read.

class rti.connextdds.**SuspendedPublication**

Bases: pybind11_object

__enter__(*self:* rti.connextdds.SuspendedPublication) → *rti.connextdds.SuspendedPublication*

Context manage the SuspendedPublication.

__exit__(*self:* rti.connextdds.SuspendedPublication, *arg0: object*, *arg1: object*, *arg2: object*) → None

__init__(*self:* rti.connextdds.SuspendedPublication, *publisher:* rti.connextdds.Publisher) → None

Suspends the publications of the publisher.

__module__ = 'rti.connextdds'

resume(*self:* rti.connextdds.SuspendedPublication) → None

Indicates that the application has completed these changes.

class rti.connextdds.**SyslogVerbosity**

Bases: pybind11_object

Members:

SILENT : Silent verbosity level

EMERGENCY : Emergency verbosity level

ALERT : Alert verbosity level

CRITICAL : Critical verbosity level

ERROR : Error verbosity level

WARNING : Warning verbosity level

NOTICE : Notice verbosity level

INFORMATIONAL : Informational verbosity level

DEBUG : Debug verbosity level

ALERT = <SyslogVerbosity.ALERT: 3>

CRITICAL = <SyslogVerbosity.CRITICAL: 7>

DEBUG = <SyslogVerbosity.DEBUG: 255>

EMERGENCY = <SyslogVerbosity.EMERGENCY: 1>

ERROR = <SyslogVerbosity.ERROR: 15>

INFORMATIONAL = <SyslogVerbosity.INFORMATIONAL: 127>

NOTICE = <SyslogVerbosity.NOTICE: 63>

SILENT = <SyslogVerbosity.SILENT: 0>

WARNING = <SyslogVerbosity.WARNING: 31>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.SyslogVerbosity) → int

__init__(*self:* rti.connextdds.SyslogVerbosity, *value: int*) → None

__int__(*self:* rti.connextdds.SyslogVerbosity) → int

__members__ = {'ALERT': <SyslogVerbosity.ALERT: 3>, 'CRITICAL': <SyslogVerbosity.CRITICAL: 7>, 'DEBUG': <SyslogVerbosity.DEBUG: 255>, 'EMERGENCY': <SyslogVerbosity.EMERGENCY: 1>, 'ERROR': <SyslogVerbosity.ERROR: 15>, 'INFORMATIONAL': <SyslogVerbosity.INFORMATIONAL: 127>, 'NOTICE': <SyslogVerbosity.NOTICE: 63>, 'SILENT': <SyslogVerbosity.SILENT: 0>, 'WARNING': <SyslogVerbosity.WARNING: 31>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.SyslogVerbosity, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

class rti.connextdds.**SystemResourceLimits**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.SystemResourceLimits, *arg0:* rti.connextdds.SystemResourceLimits) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.SystemResourceLimits) -> None

Creates a SystemResourceLimits qos policy with default values.

2. __init__(self: rti.connextdds.SystemResourceLimits, max_objects_per_thread: int) -> None

Creates a SystemResourceLimits qos policy with the provided max_objects_per_thread.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.SystemResourceLimits, *arg0:* rti.connextdds.SystemResourceLimits) → bool

Test for inequality.

property initial_objects_per_thread

The number of objects per thread for a DomainParticipantFactory for which infrastructure will initially be allocated.

property max_objects_per_thread

The maximum number of objects that can be stored per thread.

class rti.connextdds.**ThreadContext**

Bases: pybind11_object

__enter__(*self:* rti.connextdds.ThreadContext) → *rti.connextdds.ThreadContext*

__exit__(*self:* rti.connextdds.ThreadContext, *arg0: object*, *arg1: object*, *arg2: object*) → None

__init__(*self:* rti.connextdds.ThreadContext) → None

Creates a thread context that will unregister the thread on exit.

__module__ = 'rti.connextdds'

class rti.connextdds.**ThreadSettings**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.ThreadSettings, *arg0:* rti.connextdds.ThreadSettings) → bool

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ThreadSettings) -> None

Create a ThreadSettings object with default settings.

2. __init__(self: rti.connextdds.ThreadSettings, mask: rti.connextdds.ThreadSettingsKindMask, priority: int, stack_size: int, cpu_list: rti.connextdds.Int32Vector, cpu_rotation: rti.connextdds.ThreadSettingsCpuRotationKind.ThreadSettingsCpuRotationKind) -> None

Create a ThreadSettings object with the specified parameters.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.ThreadSettings, *arg0:* rti.connextdds.ThreadSettings) → bool

property cpu_list

Get/set a copy of the list of CPUs available to the thread.

property cpu_rotation

Get/set a copy of the thread settings mask.

property mask

Get/set a copy of the thread settings mask.

property priority

Get/set a copy of the thread priority.

property stack_size

Get/set a copy of the thread stack size.

class rti.connextdds.**ThreadSettingsCpuRotationKind**

Bases: pybind11_object

NO_ROTATION = <ThreadSettingsCpuRotationKind.NO_ROTATION: 0>

ROUND_ROBIN = <ThreadSettingsCpuRotationKind.ROUND_ROBIN: 1>

class ThreadSettingsCpuRotationKind

Bases: pybind11_object

Members:

NO_ROTATION : Any thread controlled by this QoS can run on any listed processor, as determined by OS scheduling.

ROUND_ROBIN : Threads controlled by this QoS will be assigned one processor from the list in round-robin order.

NO_ROTATION = <ThreadSettingsCpuRotationKind.NO_ROTATION: 0>

ROUND_ROBIN = <ThreadSettingsCpuRotationKind.ROUND_ROBIN: 1>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.ThreadSettingsCpuRotationKind.ThreadSettingsCpuRotationKind) → int

__init__ (*self:* rti.connextdds.ThreadSettingsCpuRotationKind.ThreadSettingsCpuRotationKind, *value: int*) → None

__int__ (*self:* rti.connextdds.ThreadSettingsCpuRotationKind.ThreadSettingsCpuRotationKind) → int

__members__ = {'NO_ROTATION': <ThreadSettingsCpuRotationKind.NO_ROTATION: 0>, 'ROUND_ROBIN': <ThreadSettingsCpuRotationKind.ROUND_ROBIN: 1>}

__module__ = 'rti.connextdds'

__ne__ (*self: object*, *other: object*) → bool

__repr__ (*self: object*) → str

__setstate__ (*self:* rti.connextdds.ThreadSettingsCpuRotationKind.ThreadSettingsCpuRotationKind, *state: int*) → None

__str__ ()

name(self: handle) -> str

property name

property value

__eq__ (*self:* rti.connextdds.ThreadSettingsCpuRotationKind, *arg0:* rti.connextdds.ThreadSettingsCpuRotationKind) → bool

Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.ThreadSettingsCpuRotationKind, *arg0:* rti.connextdds.ThreadSettingsCpuRotationKind) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.ThreadSettingsCpuRotationKind, *arg0:* rti.connextdds.ThreadSettingsCpuRotationKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ThreadSettingsCpuRotationKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.ThreadSettingsCpuRotationKind, arg0: rti.connextdds.ThreadSettingsCpuRotationKind.ThreadSettingsCpuRotationKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.ThreadSettingsCpuRotationKind) → *rti.connextdds.ThreadSettingsCpuRotationKind.ThreadSettingsCpuRotationKind*

Int conversion.

__le__ (*self:* rti.connextdds.ThreadSettingsCpuRotationKind, *arg0:* rti.connextdds.ThreadSettingsCpuRotationKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.ThreadSettingsCpuRotationKind, *arg0:* rti.connextdds.ThreadSettingsCpuRotationKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ThreadSettingsCpuRotationKind, *arg0:* rti.connextdds.ThreadSettingsCpuRotationKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.ThreadSettingsCpuRotationKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**ThreadSettingsKindMask**

Bases: pybind11_object

CANCEL_ASYNCHRONOUS = 0000

FLOATING_POINT = 0001

PRIORITY_ENFORCE = 0000

REALTIME_PRIORITY = 1000

STDIO = 0010

__and__ (*self:* rti.connextdds.ThreadSettingsKindMask, *arg0:* rti.connextdds.ThreadSettingsKindMask) → *rti.connextdds.ThreadSettingsKindMask*

Bitwise logical AND of masks.

__bool__ (*self:* rti.connextdds.ThreadSettingsKindMask) → bool

Test if any bits are set.

__contains__ (*self:* rti.connextdds.ThreadSettingsKindMask, *arg0:* rti.connextdds.ThreadSettingsKindMask) → bool

__eq__ (*self:* rti.connextdds.ThreadSettingsKindMask, *arg0:* rti.connextdds.ThreadSettingsKindMask) → bool

Compare masks for equality.

__getitem__ (*self:* rti.connextdds.ThreadSettingsKindMask, *arg0: int*) → bool

Get individual mask bit.

__hash__ = None

__iand__ (*self:* rti.connextdds.ThreadSettingsKindMask, *arg0:* rti.connextdds.ThreadSettingsKindMask) → *rti.connextdds.ThreadSettingsKindMask*

Set mask to logical AND with another mask.

__ilshift__ (*self:* rti.connextdds.ThreadSettingsKindMask, *arg0: int*) → *rti.connextdds.ThreadSettingsKindMask*

Left shift bits in mask.

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ThreadSettingsKindMask) -> None

Create a ThreadSettingsKindMask with default thread settings.

2. __init__(self: rti.connextdds.ThreadSettingsKindMask, value: int) -> None

Creates a mask from the bits in an integer.

__int__ (*self:* rti.connextdds.ThreadSettingsKindMask) → int

Convert mask to int.

__ior__ (*self:* rti.connextdds.ThreadSettingsKindMask, *arg0:* rti.connextdds.ThreadSettingsKindMask) → *rti.connextdds.ThreadSettingsKindMask*

Set mask to logical OR with another mask.

__irshift__ (*self:* rti.connextdds.ThreadSettingsKindMask, *arg0: int*) → *rti.connextdds.ThreadSettingsKindMask*

Right shift bits in mask.

__ixor__ (*self:* rti.connextdds.ThreadSettingsKindMask, *arg0:* rti.connextdds.ThreadSettingsKindMask) → *rti.connextdds.ThreadSettingsKindMask*

Set mask to logical XOR with another mask.

__lshift__ (*self:* rti.connextdds.ThreadSettingsKindMask, *arg0: int*) → *rti.connextdds.ThreadSettingsKindMask*

Left shift bits in mask.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ThreadSettingsKindMask, *arg0:* rti.connextdds.ThreadSettingsKindMask) → bool

Compare masks for inequality.

__or__ (*self:* rti.connextdds.ThreadSettingsKindMask, *arg0:* rti.connextdds.ThreadSettingsKindMask) → *rti.connextdds.ThreadSettingsKindMask*

Bitwise logical OR of masks.

__repr__ (*self:* rti.connextdds.ThreadSettingsKindMask) → str

Convert mask to string.

__rshift__ (*self:* rti.connextdds.ThreadSettingsKindMask, *arg0: int*) → *rti.connextdds.ThreadSettingsKindMask*

Right shift bits in mask.

__setitem__ (*self:* rti.connextdds.ThreadSettingsKindMask, *arg0: int*, *arg1: bool*) → None

Set individual mask bit

__str__ (*self:* rti.connextdds.ThreadSettingsKindMask) → str

Convert mask to string.

__xor__ (*self:* rti.connextdds.ThreadSettingsKindMask, *arg0:* rti.connextdds.ThreadSettingsKindMask) → *rti.connextdds.ThreadSettingsKindMask*

Bitwise logical XOR of masks.

property count

Returns the number of bits set in the mask.

flip (**args*, ***kwargs*)

Overloaded function.

1. flip(self: rti.connextdds.ThreadSettingsKindMask) -> rti.connextdds.ThreadSettingsKindMask

Flip all bits in the mask.

2. flip(self: rti.connextdds.ThreadSettingsKindMask, pos: int) -> rti.connextdds.ThreadSettingsKindMask

Flip the mask bit at the specified position.

reset (**args*, ***kwargs*)

Overloaded function.

1. reset(self: rti.connextdds.ThreadSettingsKindMask) -> rti.connextdds.ThreadSettingsKindMask

Clear all bits in the mask.

2. reset(self: rti.connextdds.ThreadSettingsKindMask, pos: int) -> rti.connextdds.ThreadSettingsKindMask

Clear the mask bit at the specified position.

set (**args*, ***kwargs*)

Overloaded function.

1. set(self: rti.connextdds.ThreadSettingsKindMask) -> rti.connextdds.ThreadSettingsKindMask

Set all bits in the mask.

2. set(self: rti.connextdds.ThreadSettingsKindMask, pos: int, value: bool = True) -> rti.connextdds.ThreadSettingsKindMask

Set the mask bit at the specified position to the provided value (default: true).

property size

Returns the number of bits in the mask type.

test (*self:* rti.connextdds.ThreadSettingsKindMask, *pos: int*) → bool

Test whether the mask bit at position "pos" is set.

test_all (*self:* rti.connextdds.ThreadSettingsKindMask) → bool

Test if all bits are set.

test_any (*self:* rti.connextdds.ThreadSettingsKindMask) → bool

Test if any bits are set.

test_none (*self:* rti.connextdds.ThreadSettingsKindMask) → bool

Test if none of the bits are set.

class rti.connextdds.**Time**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.Time, *arg0:* rti.connextdds.Duration) → *rti.connextdds.Time*

Add a Time and Duration together.

__eq__ (*self:* rti.connextdds.Time, *arg0:* rti.connextdds.Time) → bool

Check if this Time is equal to another.

__ge__ (*self:* rti.connextdds.Time, *arg0:* rti.connextdds.Time) → bool

Check if this Time is greater than or equal another.

__gt__ (*self:* rti.connextdds.Time, *arg0:* rti.connextdds.Time) → bool

Check if this Time is greater than another.

__hash__ = None

__iadd__ (*self:* rti.connextdds.Time, *arg0:* rti.connextdds.Duration) → *rti.connextdds.Time*

Add a Duration to this Time object.

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Time) -> None

Create a default Time object. The constructed Time object will represent 0 seconds and 0 nanoseconds.

2. __init__(self: rti.connextdds.Time, sec: int, nanosec: int = 0) -> None

Create a Time object. The constructed Time object will represent the given amount of time.

3. __init__(self: rti.connextdds.Time, other: rti.connextdds.Time) -> None

Create a copy of a Time.

4. __init__(self: rti.connextdds.Time, arg0: datetime.datetime) -> None

Create a Time object from a timestamp.

__isub__ (*self:* rti.connextdds.Time, *arg0:* rti.connextdds.Duration) → *rti.connextdds.Time*

Subtract a Duration from this Time object.

__le__ (*self:* rti.connextdds.Time, *arg0:* rti.connextdds.Time) → bool

Check if this Time is less than or equal another.

__lt__ (*self:* rti.connextdds.Time, *arg0:* rti.connextdds.Time) → bool

Check if this Time is less than another.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Time, *arg0:* rti.connextdds.Time) → bool

Check if this Time is not equal to another.

__radd__ (*self:* rti.connextdds.Time, *arg0:* rti.connextdds.Duration) → *rti.connextdds.Time*

Add a Time and Duration together.

__sub__ (**args*, ***kwargs*)

Overloaded function.

1. __sub__(self: rti.connextdds.Time, arg0: rti.connextdds.Duration) -> rti.connextdds.Time

Subtract a Duration from a Time.

2. __sub__(self: rti.connextdds.Time, arg0: rti.connextdds.Time) -> rti.connextdds.Duration

Calculate the duration between two times.

compare (*self:* rti.connextdds.Time, *other:* rti.connextdds.Time) → int

Compare two Time objects.

static from_microseconds (*microseconds: int*) → *rti.connextdds.Time*

Create a Time object from microseconds.

static from_milliseconds (*milliseconds: int*) → *rti.connextdds.Time*

Create a Time object from millisecs.

static from_seconds (*seconds: float*) → *rti.connextdds.Time*

Create a Time object from seconds.

invalid = <rti.connextdds.Time object>

maximum = <rti.connextdds.Time object>

property nanosec

The number of nanoseconds that are represented by this Time object.

property sec

The number of seconds that are represented by this Time object.

to_microseconds (*self:* rti.connextdds.Time) → int

Convert this Time to microseconds.

to_milliseconds (*self:* rti.connextdds.Time) → int

Convert this Time to milliseconds.

to_seconds (*self:* rti.connextdds.Time) → float

Convert this Time to seconds.

zero = <rti.connextdds.Time object>

class rti.connextdds.**TimeBasedFilter**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.TimeBasedFilter, *arg0:* rti.connextdds.TimeBasedFilter) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TimeBasedFilter) -> None

Creates the default time-based filter.

2. __init__(self: rti.connextdds.TimeBasedFilter, min_separation: rti.connextdds.Duration) -> None

Creates a policy with the specified minimum separation.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TimeBasedFilter, *arg0:* rti.connextdds.TimeBasedFilter) → bool

Test for inequality.

property minimum_separation

The minimum separation between subsequent samples.

exception rti.connextdds.**TimeoutError**

Bases: *Exception*

__module__ = 'rti.connextdds'

class rti.connextdds.**Topic**

Bases: *ITopicDescription*, *IAnyTopic*

__eq__ (*self:* rti.connextdds.Topic, *arg0:* rti.connextdds.Topic) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Topic, entity: rti.connextdds.IEntity) -> None

Cast an Entity to a Topic.

2. __init__(self: rti.connextdds.Topic, topic_description: rti.connextdds.ITopicDescription) -> None

Cast an ITopicDescription to a Topic.

3. __init__(self: rti.connextdds.Topic, topic: rti.connextdds.IAnyTopic) -> None

Create a typed Topic from an AnyTopic.

4. __init__(self: rti.connextdds.Topic, participant: rti.connextdds.DomainParticipant, topic_name: str, type: object, qos: rti.connextdds.TopicQos = None, listener: rti.connextdds.TopicListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL, type_name: str = None) -> None

Create a Topic for an @idl.struct or @idl.union type.

The participant, topic_name and type arguments are required. The type_name, qos, listener, mask arguments are optional. The type_name argument can be used to register the type with a name different from the class name.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Topic, *arg0:* rti.connextdds.Topic) → bool

Test for inequality.

static find (*participant:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.Topic*]

Look up a Topic by its name in the DomainParticipant.

property inconsistent_topic_status

Get a copy of the current InconsistentTopicStatus for this Topic.

property listener

The listener.

property qos

Get the TopicQos for this Topic.

This property's getter returns a deep copy.

set_listener (*self:* rti.connextdds.Topic, *listener:* rti.connextdds.TopicListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and event mask.

property type

Get the type associated with the topic.

property type_support

Get the type_support object associated with the topic.

class rti.connextdds.**TopicBuiltinTopicData**

Bases: pybind11_object

class ContentFilter

Bases: *ContentFilterBase*

__init__(*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilter) → None

__module__ = 'rti.connextdds'

compile(*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilter, *expression: str*, *parameters:* rti.connextdds.StringSeq, *type_code: Optional[*rti.connextdds.DynamicType*]*, *type_class_name: str*, *old_compile_data: Optional[object]*) → Optional[object]

Compile an instance of the content filter according to the filter expression and parameters of the given data type.

evaluate(*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilter, *compile_data: Optional[object]*, *sample:* rti.connextdds.TopicBuiltinTopicData, *meta_data:* rti.connextdds.FilterSampleInfo) → bool

Evaluate whether the sample is passing the filter or not according to the sample content.

finalize(*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilter, *compile_data: Optional[object]*) → None

A previously compiled instance of the content filter is no longer in use and resources can now be cleaned up.

class ContentFilteredTopic

Bases: *ITopicDescription*, *IAnyTopic*

__eq__(*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic, *arg0:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic, topic: rti.connextdds.TopicBuiltinTopicData.Topic, name: str, contentfilter: rti.connextdds.Filter) -> None

Create a ContentFilteredTopic with a name and Filter.

2. __init__(self: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic, topic_description: rti.connextdds.TopicBuiltinTopicData.ITopicDescription) -> None

Cast a TopicDescription to a ContentFilteredTopic.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic, *arg0:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic) → bool

Test for inequality.

append_to_expression_parameter (*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic, *index: int*, *extension: str*) → None

Append the extension to the end of parameter at the provided index, separated by a comma.

property filter_expression

Get the filter expression

property filter_parameters

Get/set the filter parameters.

static find (*participant:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic*]

Look up a ContentFilteredTopic by its name in the DomainParticipant.

remove_from_expression_parameter (*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic, *index: int*, *remove_term: str*) → None

Removes the specified term from the parameter at the provided index.

set_filter (*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic, *arg0:* rti.connextdds.Filter) → None

Set the filter.

property topic

Get the underlying Topic.

class ContentFilteredTopicSeq

Bases: `pybind11_object`

__add__ (*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq) → *rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq*

__bool__ (*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, *x:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, s: slice) -> rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, arg0: int) -> rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic

__hash__ = None

__iadd__(*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq) → *rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq*

__imul__(*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq) -> None
2. __init__(self: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, arg0: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq) → Iterator

__len__(*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq*

__ne__(*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, *arg0:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq) → bool

__rmul__(*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, arg0: int, arg1: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic) -> None
2. __setitem__(self: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, arg0: slice, arg1: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq) -> None

Assign list elements using a slice object

append(*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, *x:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic) → None

Add an item to the end of the list

clear(*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq) → None

Clear the contents

count(*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, *x:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic) → int

Return the number of times `x` appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, L: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert(*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, *i: int*, *x:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic) → None

Insert an item at a given position.

pop(**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq) -> rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic

Remove and return the last item

2. pop(self: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, i: int) -> rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic

Remove and return the item at index `i`

remove(*self:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopicSeq, *x:* rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class DataReader

Bases: *IDataReader*

class Selector

Bases: `pybind11_object`

__init__ (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader.Selector, *datareader:* rti.connextdds.TopicBuiltinTopicData.DataReader) → None

Create a Selector for a DataReader to read/take based on selected conditions

__module__ = 'rti.connextdds'

condition (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader.Selector, *condition:* rti.connextdds.IReadCondition) → *rti.connextdds.TopicBuiltinTopicData.DataReader.Selector*

Select samples based on a ReadCondition.

content (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader.Selector, *query:* rti.connextdds.Query) → *rti.connextdds.TopicBuiltinTopicData.DataReader.Selector*

Select samples based on a Query.

instance (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader.Selector, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.TopicBuiltinTopicData.DataReader.Selector*

Select a specific instance to read/take.

max_samples (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader.Selector, *max: int*) → *rti.connextdds.TopicBuiltinTopicData.DataReader.Selector*

Limit the number of samples read/taken by the Select.

next_instance (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader.Selector, *previous:* rti.connextdds.InstanceHandle) → *rti.connextdds.TopicBuiltinTopicData.DataReader.Selector*

Select the instance after the specified instance to read/take.

read (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader.Selector) → list

Read copies of available samples (data and info) based on the Selector settings.

read_data (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader.Selector) → list

Read copies of available valid data based on the Selector settings.

read_loaned (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader.Selector) → *rti.connextdds.TopicBuiltinTopicData.LoanedSamples*

Take available samples (data and info) based on the Selector settings and return them in a loaned container.

state (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader.Selector, *state:* rti.connextdds.DataState) → *rti.connextdds.TopicBuiltinTopicData.DataReader.Selector*

Select samples with a specified data state.

take (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader.Selector) → list

Take copies of available samples (data and info) based on the Selector settings.

take_data (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader.Selector) → list

Take copies of available valid data based on the Selector settings.

take_loaned (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader.Selector) → *rti.connextdds.TopicBuiltinTopicData.LoanedSamples*

Read available samples (data and info) based on the Selector settings and return them in a loaned container.

__enter__ (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader) → *rti.connextdds.TopicBuiltinTopicData.DataReader*

Enter a context for this DataReader, to be cleaned up on exiting context

__eq__ (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReader) → bool

Test for equality.

__exit__ (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for this DataReader, cleaning up resources.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataReader, topic: rti.connextdds.TopicBuiltinTopicData.Topic) -> None

Create a DataReader in the implicit subscriber with default QoS.

2. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataReader, topic: rti.connextdds.TopicBuiltinTopicData.Topic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.TopicBuiltinTopicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader in the implicit subscriber with specific QoS and a listener.

3. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataReader, cft: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic) -> None

Create a DataReader with a ContentFilteredTopic in the implicit subscriber with default QoS.

4. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataReader, cft: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.TopicBuiltinTopicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader with a ContentFilteredTopic in the implicit subscriber with specific QoS.

5. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataReader, subscriber: rti.connextdds.Subscriber, topic: rti.connextdds.TopicBuiltinTopicData.Topic) -> None

Create a DataReader.

6. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataReader, subscriber: rti.connextdds.Subscriber, topic: rti.connextdds.TopicBuiltinTopicData.Topic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.TopicBuiltinTopicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader in a subscriber with specific QoS and a listener.

7. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataReader, subscriber: rti.connextdds.Subscriber, cft: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic) -> None

Create a DataReader with a ContentFilteredTopic in a subscriber with default QoS.

8. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataReader, subscriber: rti.connextdds.Subscriber, cft: rti.connextdds.TopicBuiltinTopicData.ContentFilteredTopic, qos: rti.connextdds.DataReaderQos, listener: rti.connextdds.TopicBuiltinTopicData.DataReaderListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Create a DataReader with a ContentFilteredTopic in a subscriber with specific QoS.

9. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataReader, reader: rti.connextdds.IAnyDataReader) -> None

Get a typed DataReader from an AnyDataReader.

10. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataReader, entity: rti.connextdds.IEntity) -> None

Get a typed DataReader from an Entity.

__lshift__(*self:* rti.connextdds.TopicBuiltinTopicData.DataReader, *arg0:* rti.connextdds.DataReaderQos) → *rti.connextdds.TopicBuiltinTopicData.DataReader*

Set the DataReaderQos for this DataReader.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.TopicBuiltinTopicData.DataReader, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReader) → bool

Test for inequality.

__rshift__(*self:* rti.connextdds.TopicBuiltinTopicData.DataReader, *arg0:* rti.connextdds.DataReaderQos) → *rti.connextdds.TopicBuiltinTopicData.DataReader*

Get the DataReaderQos from this DataReader

acknowledge_all(**args*, ***kwargs*)

Overloaded function.

1. acknowledge_all(self: rti.connextdds.TopicBuiltinTopicData.DataReader) -> None

Acknowledge all previously accessed samples.

2. acknowledge_all(self: rti.connextdds.TopicBuiltinTopicData.DataReader, arg0: rti.connextdds.AckResponseData) -> None

Acknowledge all previously accessed samples.

acknowledge_sample(**args*, ***kwargs*)

Overloaded function.

1. acknowledge_sample(self: rti.connextdds.TopicBuiltinTopicData.DataReader, sample_info: rti.connextdds.SampleInfo) -> None

Acknowledge a single sample.

2. acknowledge_sample(self: rti.connextdds.TopicBuiltinTopicData.DataReader, sample_info: rti.connextdds.SampleInfo, ack_response_data: rti.connextdds.AckResponseData) -> None

Acknowledge a single sample with ack response data.

close (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader) → None

Close this DataReader.

property datareader_cache_status

Get the DataReaderCacheStatus for the DataReader.

property datareader_protocol_status

Get the DataReaderProtocolStatus for the DataReader.

property default_filter_state

Returns the filter state for the read/take operations.

static find_all_by_topic (*subscriber:* rti.connextdds.Subscriber, *topic_name: str*) → *rti.connextdds.TopicBuiltinTopicData.DataReaderSeq*

Retrieve all DataReaders for the given topic name in the subscriber.

static find_by_name (**args*, ***kwargs*)

Overloaded function.

1. find_by_name(participant: rti.connextdds.DomainParticipant, name: str) -> Optional[rti.connextdds.TopicBuiltinTopicData.DataReader]

Find DataReader in DomainParticipant with the DataReader's name, returning the first found.

2. find_by_name(subscriber: rti.connextdds.Subscriber, name: str) -> Optional[rti.connextdds.TopicBuiltinTopicData.DataReader]

Find DataReader in Subscriber with the DataReader's name, returning the first found.

static find_by_topic (*subscriber:* rti.connextdds.Subscriber, *name: str*) → Optional[*rti.connextdds.TopicBuiltinTopicData.DataReader*]

Find DataReader in Subscriber with a topic name, returning the first found.

is_matched_publication_alive (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader, *arg0:* rti.connextdds.InstanceHandle) → bool

Check if a matched publication is alive.

key_value (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.TopicBuiltinTopicData*

Retrieve the instance key that corresponds to an instance handle.

property listener

Gets or sets the listener with StatusMask.ALL

property liveliness_changed_status

Get the LivelinessChangedStatus for this DataReader.

lookup_instance (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader, *key_holder:* rti.connextdds.TopicBuiltinTopicData) → *rti.connextdds.InstanceHandle*

Retrieve the instance handle that corresponds to an instance key_holder

matched_publication_data (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.PublicationBuiltinTopicData*

Get the PublicationBuiltinTopicData for a publication matched to this DataReader.

matched_publication_datareader_protocol_status (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader, *publication_handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.DataReaderProtocolStatus*

Get the DataReaderProtocolStatus for the DataReader based on the matched publication handle.

matched_publication_participant_data (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ParticipantBuiltinTopicData*

Get the ParticipantBuiltinTopicData for a publication matched to this DataReader.

property matched_publications

Get a copy of the list of the currently matched publication handles.

property qos

The DataReaderQos for this DataReader.

This property's getter returns a deep copy.

read (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader) → list

Read copies of all available samples (data and info).

read_data (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader) → list

Read copies of all available valid data.

read_loaned (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader) → *rti.connextdds.TopicBuiltinTopicData.LoanedSamples*

Read all available samples (data and info) and return them in a loaned container.

property requested_deadline_missed_status

Get the RequestedDeadlineMissed status for the DataReader

property requested_incompatible_qos_status

Get the RequestedIncompatibleQosStatus for the DataReader.

property sample_lost_status

Get the SampleLostStatus for the DataReader.

property sample_rejected_status

Get the SampleRejectedStatus for the DataReader.

select (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader) → dds::sub::DataReader<dds::topic::TTopicBuiltinTopicData<rti::topic::TopicBuiltinTopicDataImpl>, rti::sub::DataReaderImpl>::Selector

Get a Selector to perform complex data selections, such as per-instance selection, content, and status filtering.

set_listener (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader, *listener:* rti.connextdds.TopicBuiltinTopicData.DataReaderListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and associated event mask.

property subscriber

Returns the parent Subscriber of the DataReader.

property subscription_matched_status

Get the SubscriptionMatchedStatus for the DataReader.

take (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader) → list

Take copies of all available samples (data and info).

take_data (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader) → list

Take copies of all available valid data.

take_loaned (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader) → *rti.connextdds.TopicBuiltinTopicData.LoanedSamples*

Take all available samples (data and info) and return them in a loaned container.

property topic_description

Returns the TopicDescription associated with the DataReader.

property topic_name

Get the topic name associated with this DataReader.

property type_name

Get the type name associated with this DataReader.

wait_for_historical_data (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader, *max_wait:* rti.connextdds.Duration) → None

Waits until all "historical" data is received for DataReaders that have a non-VOLATILE Durability Qos kind.

wait_for_historical_data_async (*self:* rti.connextdds.TopicBuiltinTopicData.DataReader, *max_wait:* rti.connextdds.Duration) → object

Waits until all "historical" data is received for DataReaders that have a non-VOLATILE Durability Qos kind. This call is awaitable and only for use with asyncio.

class DataReaderListener

Bases: pybind11_object

__init__ (*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderListener) → None

__module__ = 'rti.connextdds'

on_data_available (*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReader) → None

Data available callback.

on_liveliness_changed (*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReader, *arg1:* rti.connextdds.LivelinessChangedStatus) → None

Liveliness changed callback.

on_requested_deadline_missed (*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReader, *arg1:* rti.connextdds.RequestedDeadlineMissedStatus) → None

Requested deadline missed callback.

on_requested_incompatible_qos (*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReader, *arg1:* rti.connextdds.RequestedIncompatibleQosStatus) → None

Requested incompatible QoS callback.

on_sample_lost (*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SampleLostStatus) → None

Sample lost callback.

on_sample_rejected (*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SampleRejectedStatus) → None

Sample rejected callback.

on_subscription_matched (*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SubscriptionMatchedStatus) → None

Subscription matched callback.

class DataReaderSeq

Bases: pybind11_object

__add__(*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq) → *rti.connextdds.TopicBuiltinTopicData.DataReaderSeq*

__bool__(*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, *x:* rti.connextdds.TopicBuiltinTopicData.DataReader) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, s: slice) -> rti.connextdds.TopicBuiltinTopicData.DataReaderSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, arg0: int) -> rti.connextdds.TopicBuiltinTopicData.DataReader

__hash__ = None

__iadd__(*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq) → *rti.connextdds.TopicBuiltinTopicData.DataReaderSeq*

__imul__(*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicData.DataReaderSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataReaderSeq) -> None
2. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, arg0: rti.connextdds.TopicBuiltinTopicData.DataReaderSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq) → Iterator

__len__ (*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicData.DataReaderSeq*

__ne__ (*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq) → bool

__rmul__ (*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicData.DataReaderSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, arg0: int, arg1: rti.connextdds.TopicBuiltinTopicData.DataReader) -> None
2. __setitem__(self: rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, arg0: slice, arg1: rti.connextdds.TopicBuiltinTopicData.DataReaderSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, *x:* rti.connextdds.TopicBuiltinTopicData.DataReader) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq) → None

Clear the contents

count (*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, *x:* rti.connextdds.TopicBuiltinTopicData.DataReader) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, L: rti.connextdds.TopicBuiltinTopicData.DataReaderSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, *i: int*, *x:* rti.connextdds.TopicBuiltinTopicData.DataReader) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.TopicBuiltinTopicData.DataReaderSeq) -> rti.connextdds.TopicBuiltinTopicData.DataReader

Remove and return the last item

2. pop(self: rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, i: int) -> rti.connextdds.TopicBuiltinTopicData.DataReader

Remove and return the item at index `i`

remove(*self:* rti.connextdds.TopicBuiltinTopicData.DataReaderSeq, *x:* rti.connextdds.TopicBuiltinTopicData.DataReader) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class DataWriter

Bases: *IEntity*, *IAnyDataWriter*

__enter__(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter) → *rti.connextdds.TopicBuiltinTopicData.DataWriter*

Enter a context for this DataWriter, to be cleaned up on exiting context

__eq__(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter) → bool

Test for equality.

__exit__(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for this DataWriter, cleaning up resources.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, topic: rti.connextdds.TopicBuiltinTopicData.Topic) -> None

Creates a DataWriter in the implicit publisher with default QoS.

2. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, topic: rti.connextdds.TopicBuiltinTopicData.Topic, qos: rti.connextdds.DataWriterQos, listener: rti.connextdds.TopicBuiltinTopicData.DataWriterListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Creates a DataWriter in the implicit publisher with specific QoS and optionally a listener.

3. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, pub: rti.connextdds.Publisher, topic: rti.connextdds.TopicBuiltinTopicData.Topic) -> None

Creates a DataWriter in a publisher with default QoS.

4. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, pub: rti.connextdds.Publisher, topic: rti.connextdds.TopicBuiltinTopicData.Topic, qos: rti.connextdds.DataWriterQos, listener: rti.connextdds.TopicBuiltinTopicData.DataWriterListener = None, mask: rti.connextdds.StatusMask = StatusMask.ALL) -> None

Creates a DataWriter in a publisher with specific QoS and optionally a listener.

5. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, writer: rti.connextdds.IAnyDataWriter) -> None

Create a typed DataWriter from an AnyDataWriter.

6. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, entity: rti.connextdds.IEntity) -> None

Create a typed DataWriter from an Entity.

__lshift__ (**args*, ***kwargs*)

Overloaded function.

1. __lshift__(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, arg0: rti.connextdds.DataWriterQos) -> rti.connextdds.TopicBuiltinTopicData.DataWriter

Sets the DataWriterQos.

2. __lshift__(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, arg0: Tuple[rti.connextdds.TopicBuiltinTopicData, rti.connextdds.Time]) -> rti.connextdds.TopicBuiltinTopicData.DataWriter

Writes a paired sample with a timestamp.

3. __lshift__(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, arg0: Tuple[rti.connextdds.TopicBuiltinTopicData, rti.connextdds.InstanceHandle]) -> rti.connextdds.TopicBuiltinTopicData.DataWriter

Writes a paired sample with an instance handle.

4. __lshift__(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, arg0: rti.connextdds.TopicBuiltinTopicDataTimestampedSeq) -> rti.connextdds.TopicBuiltinTopicData.DataWriter

Writes a sequence of pairs of samples with timestamps.

5. __lshift__(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, arg0: rti.connextdds.TopicBuiltinTopicDataSeq) -> rti.connextdds.TopicBuiltinTopicData.DataWriter

Writes a sequence of samples.

6. __lshift__(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, arg0: rti.connextdds.TopicBuiltinTopicData) -> rti.connextdds.TopicBuiltinTopicData.DataWriter

Writes a sample.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter) → bool

Test for inequality.

__rshift__ (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg0:* rti.connextdds.DataWriterQos) → *rti.connextdds.TopicBuiltinTopicData.DataWriter*

Get the DataWriterQos.

assert_liveliness (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter) → None

Manually asserts the liveliness of the DataWriter.

close (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter) → None

Close this DataWriter.

create_data (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter) → *rti.connextdds.TopicBuiltinTopicData*

Create data of the writer's associated type and initialize it.

property datawriter_cache_status

Get a copy of the cache status for this writer.

property datawriter_protocol_status

Get a copy of the protocol status for this writer.

dispose_instance(**args*, ***kwargs*)

Overloaded function.

1. dispose_instance(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.TopicBuiltinTopicData.DataWriter

Dispose an instance.

2. dispose_instance(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> rti.connextdds.TopicBuiltinTopicData.DataWriter

Dispose an instance with a timestamp.

3. dispose_instance(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, params: rti.connextdds.WriteParams) -> None

Dispose an instance with params.

dispose_instance_async(**args*, ***kwargs*)

Overloaded function.

1. dispose_instance_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> object

Dispose an instance.

2. dispose_instance_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> object

Dispose an instance with a timestamp.

3. dispose_instance_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, key_holder: rti.connextdds.TopicBuiltinTopicData) -> object

Dispose the instance associated with key_holder.

4. dispose_instance_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, key_holder: rti.connextdds.TopicBuiltinTopicData, timestamp: rti.connextdds.Time) -> object

Dispose the instance associated with key_holder using a timestamp

5. dispose_instance_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, params: rti.connextdds.WriteParams) -> object

Dispose an instance with params.

static find_all_by_topic(*publisher:* rti.connextdds.Publisher, *topic_name: str*) → *rti.connextdds.TopicBuiltinTopicData.DataWriterSeq*

Retrieve all DataWriters for the given topic name in the publisher.

static find_by_name(**args*, ***kwargs*)

Overloaded function.

1. find_by_name(participant: rti.connextdds.DomainParticipant, name: str) -> Optional[rti.connextdds.TopicBuiltinTopicData.DataWriter]

Find DataWriter in DomainParticipant with the provided name, returning the first found.

2. find_by_name(publisher: rti.connextdds.Publisher, name: str) -> Optional[rti.connextdds.TopicBuiltinTopicData.DataWriter]

Find DataWriter in Publisher with the DataReader's name, returning the first found.

static find_by_topic (*publisher:* rti.connextdds.Publisher, *name: str*) → Optional[*rti.connextdds.TopicBuiltinTopicData.DataWriter*]

Find DataWriter in publisher with a topic name, returning the first found.

flush (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter) → None

Flushes the batch in progress in the context of thecalling thread.

is_matched_subscription_active (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg0:* rti.connextdds.InstanceHandle) → bool

A boolean indicating whether or not the matched subscription is active.

is_sample_app_acknowledged (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *sample_id:* rti.connextdds.SampleIdentity) → bool

Indicates if a sample is considered application-acknowledged.

key_value (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.TopicBuiltinTopicData*

Retrieve the instance key that corresponds to an instance handle.

property listener

Get the listener associated with the DataWriter or set the listener.

property liveliness_lost_status

Get a copy of the LivelinessLostStatus.

lookup_instance (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *key_holder:* rti.connextdds.TopicBuiltinTopicData) → *rti.connextdds.InstanceHandle*

Retrieve the instance handle that corresponds to an instance key_holder

matched_subscription_data (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.SubscriptionBuiltinTopicData*

Get the SubscriptionBuiltinTopicData for a subscription matched to this DataWriter.

matched_subscription_datawriter_protocol_status (**args*, ***kwargs*)

Overloaded function.

1. matched_subscription_datawriter_protocol_status(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.DataWriterProtocolStatus

Get a copy of the protocol status for this writer per a matched subscription handle.

2. matched_subscription_datawriter_protocol_status(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, locator: rti.connextdds.Locator) -> rti.connextdds.DataWriterProtocolStatus

Get a copy of the protocol status for this writer per a matched subscription locator.

matched_subscription_participant_data (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *handle:* rti.connextdds.InstanceHandle) → *rti.connextdds.ParticipantBuiltinTopicData*

Get the ParticipantBuiltinTopicData for a subscription matched to this DataWriter.

property matched_subscriptions

Get a copy of the list of the currently matched subscription handles.

property matched_subscriptions_locators

The locators used to communicate with matched DataReaders.

property offered_deadline_missed_status

Get a copy of the OfferedDeadlineMissedStatus.

property offered_incompatible_qos_status

Get a copy of the OfferedIncompatibleQosStatus

property publication_matched_status

Get a copy of the PublicationMatchedStatus

property publisher

Get the Publisher that owns this DataWriter.

property qos

The DataWriterQos for this DataWriter.This property's getter returns a deep copy.

register_instance (**args*, ***kwargs*)

Overloaded function.

1. register_instance(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, key_holder: rti.connextdds.TopicBuiltinTopicData) -> rti.connextdds.InstanceHandle

Informs RTI Connext that the application will be modifying a particular instance.

2. register_instance(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, key_holder: rti.connextdds.TopicBuiltinTopicData, timestamp: rti.connextdds.Time) -> rti.connextdds.InstanceHandle

Informs RTI Connext that the application will be modifying a particular instance and specified the timestamp.

3. register_instance(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, key_holder: rti.connextdds.TopicBuiltinTopicData, params: rti.connextdds.WriteParams) -> rti.connextdds.InstanceHandle

Registers instance with parameters.

property reliable_reader_activity_changed_status

Get a copy of the reliable reader activity changed status for this writer.

property reliable_writer_cache_changed_status

Get a copy of the reliable cache status for this writer.

property service_request_accepted_status

Get a copy of the service request accepted status for this writer.

set_listener(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *listener:* rti.connextdds.TopicBuiltinTopicData.DataWriterListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and event mask for the DataWriter.

property topic

Get the Topic object associated with this DataWriter.

property topic_name

Get the topic name associated with this DataWriter.

property type_name

Get the type name for the topic object associated with this DataWriter.

unregister_instance(**args*, ***kwargs*)

Overloaded function.

1. unregister_instance(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> rti.connextdds.TopicBuiltinTopicData.DataWriter

Unregister an instance.

2. unregister_instance(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> rti.connextdds.TopicBuiltinTopicData.DataWriter

Unregister an instance with timestamp.

3. unregister_instance(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, params: rti.connextdds.WriteParams) -> None

Unregister an instance with parameters.

unregister_instance_async(**args*, ***kwargs*)

Overloaded function.

1. unregister_instance_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle) -> object

Unregister an instance.

2. unregister_instance_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> object

Unregister an instance with timestamp.

3. unregister_instance_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, key_holder: rti.connextdds.TopicBuiltinTopicData) -> object

Unregister the instance associated with key_holder.

4. unregister_instance_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, key_holder: rti.connextdds.TopicBuiltinTopicData, timestamp: rti.connextdds.Time) -> object

Unregister the instance associate with key_holder using a timestamp.

5. unregister_instance_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, params: rti.connextdds.WriteParams) -> object

Unregister an instance with parameters.

wait_for_acknowledgments (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *max_wait:* rti.connextdds.Duration) → None

Blocks the calling thread until all data written by a reliable DataWriter is acknowledged or until the timeout expires.

wait_for_asynchronous_publishing (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *max_wait:* rti.connextdds.Duration) → None

This operation blocks the calling thread (up to max_wait) until all data written by the asynchronous DataWriter is sent and acknowledged (if reliable) by all matched DataReader entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; a time out indicates that max_wait elapsed before all the data was sent and/or acknowledged.

In other words, this guarantees that sending to best effort DataReader is complete in addition to what DataWriter.wait_for_acknowledgments() provides.

If the DataWriter does not have PublishMode kind set to PublishModeKind.ASYNCHRONOUS the operation will complete immediately

wait_for_asynchronous_publishing_async (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *max_wait:* rti.connextdds.Duration) → object

This function is awaitable until either a timeout of max_wait or all data written by the asynchronous DataWriter is sent and acknowledged (if reliable) by all matched DataReader entities. A successful completion indicates that all the samples written have been sent and acknowledged where applicable; a time out indicates that max_wait elapsed before all the data was sent and/or acknowledged.This function works with asyncio.

In other words, this guarantees that sending to best effort DataReader is complete in addition to what DataWriter.wait_for_acknowledgments() provides.

If the DataWriter does not have PublishMode kind set to PublishModeKind.ASYNCHRONOUS the operation will complete immediately

write (**args*, ***kwargs*)

Overloaded function.

1. write(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, samples: rti.connextdds.TopicBuiltinTopicDataSeq) -> None

Write a sequence of samples.

2. write(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, samples: rti.connextdds.TopicBuiltinTopicDataSeq, timestamp: rti.connextdds.Time) -> None

Write a sequence of samples with a timestamp.

3. write(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, sample: rti.connextdds.TopicBuiltinTopicData) -> None

Write a sample.

4. write(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, sample: rti.connextdds.TopicBuiltinTopicData, timestamp: rti.connextdds.Time) -> None

Write a sample with a specified timestamp.

5. write(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, sample: rti.connextdds.TopicBuiltinTopicData, handle: rti.connextdds.InstanceHandle) -> None

Write a sample with an instance handle.

6. write(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, sample: rti.connextdds.TopicBuiltinTopicData, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> None

Write a sample with an instance handle and specified timestamp.

7. write(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, sample: rti.connextdds.TopicBuiltinTopicData, params: rti.connextdds.WriteParams) -> None

Write with advanced parameters.

write_async(**args*, ***kwargs*)

Overloaded function.

1. write_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, sample: rti.connextdds.TopicBuiltinTopicData) -> object

Write a sample. This method is awaitable and is only for use with asyncio.

2. write_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, sample: rti.connextdds.TopicBuiltinTopicData, timestamp: rti.connextdds.Time) -> object

Write a sample with a specified timestamp. This methods is awaitable and only for use with asyncio.

3. write_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, sample: rti.connextdds.TopicBuiltinTopicData, handle: rti.connextdds.InstanceHandle) -> object

Write a sample with an instance handle. This method is awaitable and only for use with asyncio.

4. write_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, sample: rti.connextdds.TopicBuiltinTopicData, handle: rti.connextdds.InstanceHandle, timestamp: rti.connextdds.Time) -> object

Write a sample with an instance handle and specified timestamp. This method is awaitable and only for use with asyncio.

5. write_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, samples: rti.connextdds.TopicBuiltinTopicDataSeq) -> object

Write a sequence of samples. This method is awaitable and only for use with asyncio.

6. write_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, samples: rti.connextdds.TopicBuiltinTopicDataSeq, timestamp: rti.connextdds.Time) -> object

Write a sequence of samples with a timestamp. This method is awaitable and only for use with asyncio.

7. write_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, samples: rti.connextdds.TopicBuiltinTopicDataSeq, handles: rti.connextdds.InstanceHandleSeq) -> object

Write a sequence of samples with their instance handles. This method is awaitable and only for use with asyncio.

8. write_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, samples: rti.connextdds.TopicBuiltinTopicDataSeq, handles: rti.connextdds.InstanceHandleSeq, timestamp: rti.connextdds.Time) -> object

Write a sequence of samples with their instance handles and a timestamp. This method is awaitable and only for use with asyncio.

9. write_async(self: rti.connextdds.TopicBuiltinTopicData.DataWriter, sample: rti.connextdds.TopicBuiltinTopicData, params: rti.connextdds.WriteParams) -> object

Write with advanced parameters.

class DataWriterListener

Bases: pybind11_object

__init__ (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterListener) → None

__module__ = 'rti.connextdds'

on_application_acknowledgment (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.AcknowledgmentInfo) → None

On application acknowledgment callback

on_instance_replaced (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.InstanceHandle) → None

On instance replaced callback.

on_liveliness_lost (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.LivelinessLostStatus) → None

Liveliness lost callback.

on_offered_deadline_missed (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.OfferedDeadlineMissedStatus) → None

Offered deadline missed callback.

on_offered_incompatible_qos (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.OfferedIncompatibleQosStatus) → None

Offered incompatible QoS callback.

on_publication_matched (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.PublicationMatchedStatus) → None

Publication matched callback.

on_reliable_reader_activity_changed (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ReliableReaderActivityChangedStatus) → None

Reliable reader activity changed callback.

on_reliable_writer_cache_changed(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ReliableWriterCacheChangedStatus) → None

Reliable writer cache changed callback.

on_service_request_accepted(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ServiceRequestAcceptedStatus) → None

On service request accepted callback.

class DataWriterSeq

Bases: `pybind11_object`

__add__(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq) → *rti.connextdds.TopicBuiltinTopicData.DataWriterSeq*

__bool__(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, *x:* rti.connextdds.TopicBuiltinTopicData.DataWriter) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, s: slice) -> rti.connextdds.TopicBuiltinTopicData.DataWriterSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, arg0: int) -> rti.connextdds.TopicBuiltinTopicData.DataWriter

__hash__ = None

__iadd__(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq) → *rti.connextdds.TopicBuiltinTopicData.DataWriterSeq*

__imul__(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicData.DataWriterSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataWriterSeq) -> None
2. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, arg0: rti.connextdds.TopicBuiltinTopicData.DataWriterSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq) → Iterator

__len__(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicData.DataWriterSeq*

__ne__(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq) → bool

__rmul__(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicData.DataWriterSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, arg0: int, arg1: rti.connextdds.TopicBuiltinTopicData.DataWriter) -> None
2. __setitem__(self: rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, arg0: slice, arg1: rti.connextdds.TopicBuiltinTopicData.DataWriterSeq) -> None

Assign list elements using a slice object

append(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, *x:* rti.connextdds.TopicBuiltinTopicData.DataWriter) → None

Add an item to the end of the list

clear(*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq) → None

Clear the contents

count (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, *x:* rti.connextdds.TopicBuiltinTopicData.DataWriter) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, L: rti.connextdds.TopicBuiltinTopicData.DataWriterSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, *i: int*, *x:* rti.connextdds.TopicBuiltinTopicData.DataWriter) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.TopicBuiltinTopicData.DataWriterSeq) -> rti.connextdds.TopicBuiltinTopicData.DataWriter

Remove and return the last item

2. pop(self: rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, i: int) -> rti.connextdds.TopicBuiltinTopicData.DataWriter

Remove and return the item at index `i`

remove (*self:* rti.connextdds.TopicBuiltinTopicData.DataWriterSeq, *x:* rti.connextdds.TopicBuiltinTopicData.DataWriter) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class ITopicDescription

Bases: *IEntity*

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

property name

The name of the entity conforming to the ITopicDescription interface.

property participant

The parent DomainParticipant.

property type_name

The name of the associated type.

class LoanedSample

Bases: `pybind11_object`

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicBuiltinTopicData.LoanedSample) -> None

Basic constructor

2. __init__(self: rti.connextdds.TopicBuiltinTopicData.LoanedSample, data: rti.connextdds.TopicBuiltinTopicData, info: rti.connextdds.SampleInfo) -> None

Construct LoanedSample with data and info.

__iter__ (*self:* rti.connextdds.TopicBuiltinTopicData.LoanedSample) → object

__module__ = 'rti.connextdds'

property data

Get the data associated with the sample.

property info

Get the info associated with the sample.

class LoanedSamples

Bases: pybind11_object

__enter__ (*self:* rti.connextdds.TopicBuiltinTopicData.LoanedSamples) → *rti.connextdds.TopicBuiltinTopicData.LoanedSamples*

Enter a context for the loaned samples, loan returned on context exit.

__exit__ (*self:* rti.connextdds.TopicBuiltinTopicData.LoanedSamples, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit the context for the loaned samples, returning the resources.

__getitem__ (*self:* rti.connextdds.TopicBuiltinTopicData.LoanedSamples, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicData.LoanedSample*

Access a LoanedSample object in an array-like syntax

__init__ (*self:* rti.connextdds.TopicBuiltinTopicData.LoanedSamples) → None

Create an empty LoanedSamples object.

__iter__ (*self:* rti.connextdds.TopicBuiltinTopicData.LoanedSamples) → Iterator

__len__ (*self:* rti.connextdds.TopicBuiltinTopicData.LoanedSamples) → int

Get the number of samples in the loan.

__module__ = 'rti.connextdds'

property length

Get the number of samples in the loan.

return_loan (*self:* rti.connextdds.TopicBuiltinTopicData.LoanedSamples) → None

Returns the loan to the DataReader.

class NoOpDataReaderListener

Bases: *DataReaderListener*

__init__ (*self:* rti.connextdds.TopicBuiltinTopicData.NoOpDataReaderListener) → None

__module__ = 'rti.connextdds'

on_data_available(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReader) → None

Data available callback.

on_liveliness_changed(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReader, *arg1:* rti.connextdds.LivelinessChangedStatus) → None

Liveliness changed callback.

on_requested_deadline_missed(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReader, *arg1:* rti.connextdds.RequestedDeadlineMissedStatus) → None

Requested deadline missed callback.

on_requested_incompatible_qos(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReader, *arg1:* rti.connextdds.RequestedIncompatibleQosStatus) → None

Requested incompatible QoS callback.

on_sample_lost(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SampleLostStatus) → None

Sample lost callback.

on_sample_rejected(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SampleRejectedStatus) → None

Sample rejected callback.

on_subscription_matched(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpDataReaderListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataReader, *arg1:* rti.connextdds.SubscriptionMatchedStatus) → None

Subscription matched callback.

class NoOpDataWriterListener

Bases: *DataWriterListener*

__init__(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpDataWriterListener) → None

__module__ = 'rti.connextdds'

on_application_acknowledgment(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.AcknowledgmentInfo) → None

On application acknowledgment callback

on_instance_replaced(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.InstanceHandle) → None

On instance replaced callback.

on_liveliness_lost(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.LivelinessLostStatus) → None

Liveliness lost callback.

on_offered_deadline_missed(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.OfferedDeadlineMissedStatus) → None

Offered deadline missed callback.

on_offered_incompatible_qos(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.OfferedIncompatibleQosStatus) → None

Offered incompatible QoS callback.

on_publication_matched(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.PublicationMatchedStatus) → None

Publication matched callback.

on_reliable_reader_activity_changed(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ReliableReaderActivityChangedStatus) → None

Reliable reader activity changed callback.

on_reliable_writer_cache_changed(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ReliableWriterCacheChangedStatus) → None

Reliable writer cache changed callback.

on_service_request_accepted(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpDataWriterListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.DataWriter, *arg1:* rti.connextdds.ServiceRequestAcceptedStatus) → None

On service request accepted callback.

class NoOpTopicListener

Bases: *TopicListener*

__init__(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpTopicListener) → None

__module__ = 'rti.connextdds'

on_inconsistent_topic(*self:* rti.connextdds.TopicBuiltinTopicData.NoOpTopicListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.Topic, *arg1:* rti.connextdds.InconsistentTopicStatus) → None

Inconsistent topic callback.

class Sample

Bases: pybind11_object

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicBuiltinTopicData.Sample, data: rti.connextdds.TopicBuiltinTopicData, info: rti.connextdds.SampleInfo) -> None

Construct Sample with data and info.

2. __init__(self: rti.connextdds.TopicBuiltinTopicData.Sample, sample: rti.connextdds.TopicBuiltinTopicData.Sample) -> None

Copy constructor.

3. __init__(self: rti.connextdds.TopicBuiltinTopicData.Sample) -> None

Basic constructor

4. __init__(self: rti.connextdds.TopicBuiltinTopicData.Sample, loaned_sample: rti.connextdds.TopicBuiltinTopicData.LoanedSample) -> None

Construct a sample with a loaned sample.

__iter__(*self:* rti.connextdds.TopicBuiltinTopicData.Sample) → object

__module__ = 'rti.connextdds'

property data

The data associated with the sample.

property info

Get the info associated with the sample.

class SharedSamples

Bases: pybind11_object

__getitem__(*self:* rti.connextdds.TopicBuiltinTopicData.SharedSamples, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicData.LoanedSample*

Get the sample at the specified index.

__init__(*self:* rti.connextdds.TopicBuiltinTopicData.SharedSamples, *loaned_samples:* rti.connextdds.TopicBuiltinTopicData.LoanedSamples) → None

Constructs an instance of SharedSamples, removing ownership of the loan from the Loaned Samples.

__iter__(*self:* rti.connextdds.TopicBuiltinTopicData.SharedSamples) → Iterator

Make a sample iterator

__len__(*self:* rti.connextdds.TopicBuiltinTopicData.SharedSamples) → int

Returns the number of samples.

__module__ = 'rti.connextdds'

class Topic

Bases: *ITopicDescription*, *IAnyTopic*

__eq__(*self:* rti.connextdds.TopicBuiltinTopicData.Topic, *arg0:* rti.connextdds.TopicBuiltinTopicData.Topic) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicBuiltinTopicData.Topic, entity: rti.connextdds.IEntity) -> None

Cast an Entity to a Topic.

2. __init__(self: rti.connextdds.TopicBuiltinTopicData.Topic, topic_description: rti.connextdds.TopicBuiltinTopicData.ITopicDescription) -> None

Cast an ITopicDescription to a Topic.

3. __init__(self: rti.connextdds.TopicBuiltinTopicData.Topic, topic: rti.connextdds.IAnyTopic) -> None

Create a typed Topic from an AnyTopic.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TopicBuiltinTopicData.Topic, *arg0:* rti.connextdds.TopicBuiltinTopicData.Topic) → bool

Test for inequality.

static find (*participant:* rti.connextdds.DomainParticipant, *name: str*) → Optional[*rti.connextdds.TopicBuiltinTopicData.Topic*]

Look up a Topic by its name in the DomainParticipant.

property inconsistent_topic_status

Get a copy of the current InconsistentTopicStatus for this Topic.

property listener

The listener.

property qos

Get the TopicQos for this Topic.

This property's getter returns a deep copy.

set_listener (*self:* rti.connextdds.TopicBuiltinTopicData.Topic, *listener:* rti.connextdds.TopicBuiltinTopicData.TopicListener, *event_mask:* rti.connextdds.StatusMask) → None

Set the listener and event mask.

class TopicDescription

Bases: *ITopicDescription*

__eq__ (*self:* rti.connextdds.TopicBuiltinTopicData.TopicDescription, *arg0:* rti.connextdds.TopicBuiltinTopicData.TopicDescription) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicBuiltinTopicData.TopicDescription, topic: rti.connextdds.TopicBuiltinTopicData.ITopicDescription) -> None

Cast a Topic to a TopicDescription.

2. __init__(self: rti.connextdds.TopicBuiltinTopicData.TopicDescription, entity: rti.connextdds.IEntity) -> None

Cast a Topic to a TopicDescription.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TopicBuiltinTopicData.TopicDescription, *arg0:* rti.connextdds.TopicBuiltinTopicData.TopicDescription) → bool

Test for inequality.

class TopicListener

Bases: pybind11_object

__init__ (*self:* rti.connextdds.TopicBuiltinTopicData.TopicListener) → None

__module__ = 'rti.connextdds'

on_inconsistent_topic (*self:* rti.connextdds.TopicBuiltinTopicData.TopicListener, *arg0:* rti.connextdds.TopicBuiltinTopicData.Topic, *arg1:* rti.connextdds.InconsistentTopicStatus) → None

Inconsistent topic callback.

class TopicSeq

Bases: pybind11_object

__add__ (*self:* rti.connextdds.TopicBuiltinTopicData.TopicSeq, *arg0:* rti.connextdds.TopicBuiltinTopicData.TopicSeq) → *rti.connextdds.TopicBuiltinTopicData.TopicSeq*

__bool__ (*self:* rti.connextdds.TopicBuiltinTopicData.TopicSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.TopicBuiltinTopicData.TopicSeq, *x:* rti.connextdds.TopicBuiltinTopicData.Topic) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.TopicBuiltinTopicData.TopicSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.TopicBuiltinTopicData.TopicSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.TopicBuiltinTopicData.TopicSeq, *arg0:* rti.connextdds.TopicBuiltinTopicData.TopicSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.TopicBuiltinTopicData.TopicSeq, s: slice) -> rti.connextdds.TopicBuiltinTopicData.TopicSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.TopicBuiltinTopicData.TopicSeq, arg0: int) -> rti.connextdds.TopicBuiltinTopicData.Topic

__hash__ = None

__iadd__ (*self:* rti.connextdds.TopicBuiltinTopicData.TopicSeq, *arg0:* rti.connextdds.TopicBuiltinTopicData.TopicSeq) → *rti.connextdds.TopicBuiltinTopicData.TopicSeq*

__imul__ (*self:* rti.connextdds.TopicBuiltinTopicData.TopicSeq, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicData.TopicSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicBuiltinTopicData.TopicSeq) -> None
2. __init__(self: rti.connextdds.TopicBuiltinTopicData.TopicSeq, arg0: rti.connextdds.TopicBuiltinTopicData.TopicSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.TopicBuiltinTopicData.TopicSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.TopicBuiltinTopicData.TopicSeq) → Iterator

__len__(*self:* rti.connextdds.TopicBuiltinTopicData.TopicSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.TopicBuiltinTopicData.TopicSeq, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicData.TopicSeq*

__ne__(*self:* rti.connextdds.TopicBuiltinTopicData.TopicSeq, *arg0:* rti.connextdds.TopicBuiltinTopicData.TopicSeq) → bool

__rmul__(*self:* rti.connextdds.TopicBuiltinTopicData.TopicSeq, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicData.TopicSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.TopicBuiltinTopicData.TopicSeq, arg0: int, arg1: rti.connextdds.TopicBuiltinTopicData.Topic) -> None
2. __setitem__(self: rti.connextdds.TopicBuiltinTopicData.TopicSeq, arg0: slice, arg1: rti.connextdds.TopicBuiltinTopicData.TopicSeq) -> None

Assign list elements using a slice object

append(*self:* rti.connextdds.TopicBuiltinTopicData.TopicSeq, *x:* rti.connextdds.TopicBuiltinTopicData.Topic) → None

Add an item to the end of the list

clear(*self:* rti.connextdds.TopicBuiltinTopicData.TopicSeq) → None

Clear the contents

count(*self:* rti.connextdds.TopicBuiltinTopicData.TopicSeq, *x:* rti.connextdds.TopicBuiltinTopicData.Topic) → int

Return the number of times x appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.TopicBuiltinTopicData.TopicSeq, L: rti.connextdds.TopicBuiltinTopicData.TopicSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.TopicBuiltinTopicData.TopicSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.TopicBuiltinTopicData.TopicSeq, *i: int*, *x:* rti.connextdds.TopicBuiltinTopicData.Topic) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.TopicBuiltinTopicData.TopicSeq) -> rti.connextdds.TopicBuiltinTopicData.Topic

Remove and return the last item

2. pop(self: rti.connextdds.TopicBuiltinTopicData.TopicSeq, i: int) -> rti.connextdds.TopicBuiltinTopicData.Topic

Remove and return the item at index `i`

remove (*self:* rti.connextdds.TopicBuiltinTopicData.TopicSeq, *x:* rti.connextdds.TopicBuiltinTopicData.Topic) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class ValidLoanedSamples

Bases: `pybind11_object`

__enter__ (*self:* rti.connextdds.TopicBuiltinTopicData.ValidLoanedSamples) → *rti.connextdds.TopicBuiltinTopicData.ValidLoanedSamples*

__exit__ (*self:* rti.connextdds.TopicBuiltinTopicData.ValidLoanedSamples, *arg0: object*, *arg1: object*, *arg2: object*) → None

__init__ (**args*, ***kwargs*)

__iter__ (*self:* rti.connextdds.TopicBuiltinTopicData.ValidLoanedSamples) → Iterator

__module__ = 'rti.connextdds'

class WriterContentFilter

Bases: *ContentFilter*

__init__ (*self:* rti.connextdds.TopicBuiltinTopicData.WriterContentFilter) → None

__module__ = 'rti.connextdds'

writer_attach (*self:* rti.connextdds.TopicBuiltinTopicData.WriterContentFilter) → Optional[object]

A writer-side filtering API to create some state that can facilitate filtering on the writer side.

writer_compile (*self:* rti.connextdds.TopicBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *property:* rti.connextdds.ExpressionProperty, *expression: str*, *parameters:* rti.connextdds.StringSeq, *type_code: Optional[*rti.connextdds.DynamicType*]*, *type_class_name: str*, *cookie:* rti.connextdds.Cookie) → None

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching DataReader.

writer_detach (*self:* rti.connextdds.TopicBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*) → None

A writer-side filtering API to clean up a previously created state using writer_attach.

writer_evaluate (*self:* rti.connextdds.TopicBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *sample:* rti.connextdds.TopicBuiltinTopicData, *meta_data:* rti.connextdds.FilterSampleInfo) → *rti.connextdds.CookieVector*

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching DataReader.

writer_finalize (*self:* rti.connextdds.TopicBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *cookie:* rti.connextdds.Cookie) → None

A writer-side filtering API to clean up a previously compiled instance of the content filter.

writer_return_loan (*self:* rti.connextdds.TopicBuiltinTopicData.WriterContentFilter, *writer_filter_data: Optional[object]*, *cookies:* rti.connextdds.CookieVector) → None

A writer-side filtering API to return the loan on the list of DataReaders returned by writer_evaluate.

class WriterContentFilterHelper

Bases: *WriterContentFilter*

__init__ (*self:* rti.connextdds.TopicBuiltinTopicData.WriterContentFilterHelper) → None

__module__ = 'rti.connextdds'

add_cookie (*self:* rti.connextdds.TopicBuiltinTopicData.WriterContentFilterHelper, *cookie:* rti.connextdds.Cookie) → None

A helper function which will add a Cookie to the Cookie sequence that is then returned by the writer_evaluate function.

writer_evaluate_helper (*self:* rti.connextdds.TopicBuiltinTopicData.WriterContentFilterHelper, *writer_filter_data: Optional[object]*, *sample:* rti.connextdds.TopicBuiltinTopicData, *meta_data:* rti.connextdds.FilterSampleInfo) → None

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching DataReader.

__eq__ (*self:* rti.connextdds.TopicBuiltinTopicData, *arg0:* rti.connextdds.TopicBuiltinTopicData) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.TopicBuiltinTopicData) → None

Create a default TopicBuiltinTopicData.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TopicBuiltinTopicData, *arg0:* rti.connextdds.TopicBuiltinTopicData) → bool

Test for inequality.

property deadline

Get the Deadline policy of the corresponding Topic.

property destination_order

Get the DestinationOrder policy of the corresponding Topic.

property durability

Get the Durability policy of the corresponding Topic.

property durability_service

Get the DurabilityService policy of the corresponding Topic.

property history

Get the History policy of the corresponding Topic.

property key

Get the DCPS key to distinguish entries.

property latency_budget

Get the LatencyBudget policy of the corresponding Topic.

property lifespan

Get the Lifespan policy of the corresponding Topic.

property liveliness

Get the Liveliness policy of the corresponding Topic.

property name

Get the name of the Topic.

property ownership

Get the Ownership policy of the corresponding Topic.

property reliability

Get the Reliability policy of the corresponding Topic.

property representation

Get the DataRepresentation policy of the corresponding Topic.

property resource_limits

Get the ResourceLimits policy of the corresponding Topic.

property topic_data

Get the TopicData policy of the corresponding Topic.

topic_name = 'DCPSTopic'

property transport_priority

Get the TransportPriority policy of the corresponding Topic.

property type_name

Get the name of the type attached to the Topic.

class rti.connextdds.**TopicBuiltinTopicDataSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.TopicBuiltinTopicDataSeq, *arg0:* rti.connextdds.TopicBuiltinTopicDataSeq) → *rti.connextdds.TopicBuiltinTopicDataSeq*

__bool__(*self:* rti.connextdds.TopicBuiltinTopicDataSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.TopicBuiltinTopicDataSeq, *x:* rti.connextdds.TopicBuiltinTopicData) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.TopicBuiltinTopicDataSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.TopicBuiltinTopicDataSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.TopicBuiltinTopicDataSeq, *arg0:* rti.connextdds.TopicBuiltinTopicDataSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.TopicBuiltinTopicDataSeq, s: slice) -> rti.connextdds.TopicBuiltinTopicDataSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.TopicBuiltinTopicDataSeq, arg0: int) -> rti.connextdds.TopicBuiltinTopicData

__hash__ = None

__iadd__(*self:* rti.connextdds.TopicBuiltinTopicDataSeq, *arg0:* rti.connextdds.TopicBuiltinTopicDataSeq) → *rti.connextdds.TopicBuiltinTopicDataSeq*

__imul__(*self:* rti.connextdds.TopicBuiltinTopicDataSeq, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicDataSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicBuiltinTopicDataSeq) -> None

2. __init__(self: rti.connextdds.TopicBuiltinTopicDataSeq, arg0: rti.connextdds.TopicBuiltinTopicDataSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.TopicBuiltinTopicDataSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.TopicBuiltinTopicDataSeq) → Iterator

__len__ (*self:* rti.connextdds.TopicBuiltinTopicDataSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.TopicBuiltinTopicDataSeq, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicDataSeq*

__ne__ (*self:* rti.connextdds.TopicBuiltinTopicDataSeq, *arg0:* rti.connextdds.TopicBuiltinTopicDataSeq) → bool

__rmul__ (*self:* rti.connextdds.TopicBuiltinTopicDataSeq, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicDataSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.TopicBuiltinTopicDataSeq, arg0: int, arg1: rti.connextdds.TopicBuiltinTopicData) -> None

2. __setitem__(self: rti.connextdds.TopicBuiltinTopicDataSeq, arg0: slice, arg1: rti.connextdds.TopicBuiltinTopicDataSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.TopicBuiltinTopicDataSeq, *x:* rti.connextdds.TopicBuiltinTopicData) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.TopicBuiltinTopicDataSeq) → None

Clear the contents

count (*self:* rti.connextdds.TopicBuiltinTopicDataSeq, *x:* rti.connextdds.TopicBuiltinTopicData) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.TopicBuiltinTopicDataSeq, L: rti.connextdds.TopicBuiltinTopicDataSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.TopicBuiltinTopicDataSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.TopicBuiltinTopicDataSeq, *i: int*, *x:* rti.connextdds.TopicBuiltinTopicData) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.TopicBuiltinTopicDataSeq) -> rti.connextdds.TopicBuiltinTopicData

Remove and return the last item

2. pop(self: rti.connextdds.TopicBuiltinTopicDataSeq, i: int) -> rti.connextdds.TopicBuiltinTopicData

Remove and return the item at index `i`

remove (*self:* rti.connextdds.TopicBuiltinTopicDataSeq, *x:* rti.connextdds.TopicBuiltinTopicData) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**TopicBuiltinTopicDataTimestampedSeq**

Bases: `pybind11_object`

__add__ (*self:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, *arg0:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq) → *rti.connextdds.TopicBuiltinTopicDataTimestampedSeq*

__bool__ (*self:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, *x:* *Tuple[*rti.connextdds.TopicBuiltinTopicData, rti.connextdds.Time*]*) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, *arg0:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, s: slice) -> rti.connextdds.TopicBuiltinTopicDataTimestampedSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, arg0: int) -> Tuple[rti.connextdds.TopicBuiltinTopicData, rti.connextdds.Time]

__hash__ = None

__iadd__ (*self:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, *arg0:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq) → *rti.connextdds.TopicBuiltinTopicDataTimestampedSeq*

__imul__ (*self:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicDataTimestampedSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicBuiltinTopicDataTimestampedSeq) -> None

2. __init__(self: rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, arg0: rti.connextdds.TopicBuiltinTopicDataTimestampedSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq) → Iterator

__len__ (*self:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicDataTimestampedSeq*

__ne__ (*self:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, *arg0:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__rmul__ (*self:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, *arg0: int*) → *rti.connextdds.TopicBuiltinTopicDataTimestampedSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, arg0: int, arg1: Tuple[rti.connextdds.TopicBuiltinTopicData, rti.connextdds.Time]) -> None

2. __setitem__(self: rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, arg0: slice, arg1: rti.connextdds.TopicBuiltinTopicDataTimestampedSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, *x:* *Tuple[*rti.connextdds.TopicBuiltinTopicData, rti.connextdds.Time*]*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq) → None

Clear the contents

count (*self:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, *x:* *Tuple[*rti.connextdds.TopicBuiltinTopicData, rti.connextdds.Time*]*) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, L: rti.connextdds.TopicBuiltinTopicDataTimestampedSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, *i: int*, *x:* *Tuple[*rti.connextdds.TopicBuiltinTopicData, rti.connextdds.Time*]*) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.TopicBuiltinTopicDataTimestampedSeq) -> Tuple[rti.connextdds.TopicBuiltinTopicData, rti.connextdds.Time]

Remove and return the last item

2. pop(self: rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, i: int) -> Tuple[rti.connextdds.TopicBuiltinTopicData, rti.connextdds.Time]

Remove and return the item at index `i`

remove (*self:* rti.connextdds.TopicBuiltinTopicDataTimestampedSeq, *x:* *Tuple[*rti.connextdds.TopicBuiltinTopicData, rti.connextdds.Time*]*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**TopicData**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.TopicData, *arg0:* rti.connextdds.TopicData) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicData) -> None

Create an empty TopicData QoS policy.

2. __init__(self: rti.connextdds.TopicData, data: rti.connextdds.Uint8Seq) -> None

Create a TopicData object from a data sequence.

__iter__(*self:* rti.connextdds.TopicData) → Iterator

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.TopicData, *arg0:* rti.connextdds.TopicData) → bool

Test for inequality.

property value

Get/set a copy of the TopicData value.

class rti.connextdds.**TopicDescription**

Bases: *ITopicDescription*

__eq__(*self:* rti.connextdds.TopicDescription, *arg0:* rti.connextdds.TopicDescription) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicDescription, topic: rti.connextdds.ITopicDescription) -> None

Cast a Topic to a TopicDescription.

2. __init__(self: rti.connextdds.TopicDescription, entity: rti.connextdds.IEntity) -> None

Cast a Topic to a TopicDescription.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.TopicDescription, *arg0:* rti.connextdds.TopicDescription) → bool

Test for inequality.

class rti.connextdds.**TopicListener**

Bases: pybind11_object

__init__(*self:* rti.connextdds.TopicListener) → None

__module__ = 'rti.connextdds'

on_inconsistent_topic(*self:* rti.connextdds.TopicListener, *arg0:* rti.connextdds.Topic, *arg1:* rti.connextdds.InconsistentTopicStatus) → None

Inconsistent topic callback.

class rti.connextdds.**TopicQos**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.TopicQos, *arg0:* rti.connextdds.TopicQos) → bool

Test for equality

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicQos) -> None

Create a TopicQos with the default value for each policy.

2. __init__(self: rti.connextdds.TopicQos, topic: rti.connextdds.IAnyTopic) -> None

Create a TopicQos with settings equivalent to those of the provided Topic.

3. __init__(self: rti.connextdds.TopicQos, other: rti.connextdds.TopicQos) -> None

Create a copy of a TopicQos object.

__lshift__(**args*, ***kwargs*)

Overloaded function.

1. __lshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.TopicData) -> rti.connextdds.TopicQos

Set the TopicDataQoS.

2. __lshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.Durability) -> rti.connextdds.TopicQos

Set the DurabilityQoS.

3. __lshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.DurabilityService) -> rti.connextdds.TopicQos

Set the DurabilityServiceQoS.

4. __lshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.Deadline) -> rti.connextdds.TopicQos

Set the DeadlineQoS.

5. __lshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.LatencyBudget) -> rti.connextdds.TopicQos

Set the LatencyBudgetQoS.

6. __lshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.Liveliness) -> rti.connextdds.TopicQos

Set the LivelinessQoS.

7. __lshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.Reliability) -> rti.connextdds.TopicQos

Set the ReliabilityQoS.

8. __lshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.DestinationOrder) -> rti.connextdds.TopicQos

Set the DestinationOrderQoS.

9. __lshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.History) -> rti.connextdds.TopicQos

Set the HistoryQoS.

10. __lshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.ResourceLimits) -> rti.connextdds.TopicQos

Set the ResourceLimitsQoS.

11. __lshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.TransportPriority) -> rti.connextdds.TopicQos

Set the TransportPriorityQoS.

12. __lshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.Lifespan) -> rti.connextdds.TopicQos

Set the LifespanQoS.

13. __lshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.Ownership) -> rti.connextdds.TopicQos

Set the OwnershipQoS.

14. __lshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.DataRepresentation) -> rti.connextdds.TopicQos

Set the DataRepresentationQoS.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.TopicQos, *arg0:* rti.connextdds.TopicQos) → bool

Test for inequality.

__rshift__(**args*, ***kwargs*)

Overloaded function.

1. __rshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.TopicData) -> rti.connextdds.TopicQos

Get the TopicDataQoS.

2. __rshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.Durability) -> rti.connextdds.TopicQos

Get the DurabilityQoS.

3. __rshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.DurabilityService) -> rti.connextdds.TopicQos

Get the DurabilityServiceQoS.

4. __rshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.Deadline) -> rti.connextdds.TopicQos

Get the DeadlineQoS.

5. __rshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.LatencyBudget) -> rti.connextdds.TopicQos

Get the LatencyBudgetQoS.

6. __rshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.Liveliness) -> rti.connextdds.TopicQos

Get the LivelinessQoS.

7. __rshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.Reliability) -> rti.connextdds.TopicQos

Get the ReliabilityQoS.

8. __rshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.DestinationOrder) -> rti.connextdds.TopicQos

Get the DestinationOrderQoS.

9. __rshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.History) -> rti.connextdds.TopicQos

Get the HistoryQoS.

10. __rshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.ResourceLimits) -> rti.connextdds.TopicQos

Get the ResourceLimitsQoS.

11. __rshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.TransportPriority) -> rti.connextdds.TopicQos

Get the TransportPriorityQoS.

12. __rshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.Lifespan) -> rti.connextdds.TopicQos

Get the LifespanQoS.

13. __rshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.Ownership) -> rti.connextdds.TopicQos

    Get the OwnershipQoS.

14. __rshift__(self: rti.connextdds.TopicQos, arg0: rti.connextdds.DataRepresentation) -> rti.connextdds.TopicQos

    Get the DataRepresentationQoS.

__str__ (*self:* rti.connextdds.TopicQos) → str

property data_representation

Get/set DataRepresentation QoS.

property deadline

Get/set Deadline QoS.

property destination_order

Get/set DestinationOrder QoS.

property durability

Get/set Durability QoS.

property durability_service

Get/set DurabilityService QoS.

property history

Get/set History QoS.

property latency_budget

Get/set LatencyBudget QoS.

property lifespan

Get/set Lifespan QoS.

property liveliness

Get/set Liveliness QoS.

property ownership

Get/set Ownership QoS.

property reliability

Get/set Reliability QoS.

property resource_limits

Get/set ResourceLimits QoS.

to_string (*self:* rti.connextdds.TopicQos, *format:* rti.connextdds.QosPrintFormat = *QosPrintFormat()*, *base: Optional[*rti.connextdds.TopicQos*]* = *None*, *print_all: bool* = *False*) → str

Convert QoS to string based on params.

property topic_data

Get/set TopicData QoS.

property transport_priority

Get/set TransportPriority QoS.

class rti.connextdds.**TopicQuery**

Bases: pybind11_object

__enter__(*self:* rti.connextdds.TopicQuery) → *rti.connextdds.TopicQuery*

Enter a context managed block for a TopicQuery.

__eq__(*self:* rti.connextdds.TopicQuery, *arg0:* rti.connextdds.TopicQuery) → bool

Compare DataStateEx objects for equality.

__exit__(*self:* rti.connextdds.TopicQuery, *arg0: object*, *arg1: object*, *arg2: object*) → None

Exit a context managed block for a TopicQuery.

__hash__ = None

__init__(*self:* rti.connextdds.TopicQuery, *reader:* rti.connextdds.IAnyDataReader, *selection:* rti.connextdds.TopicQuerySelection) → None

Creates a TopicQuery for a given DataReader.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.TopicQuery, *arg0:* rti.connextdds.TopicQuery) → bool

Compare DataStateEx objects for inequality.

close(*self:* rti.connextdds.TopicQuery) → None

Deletes and cancels this TopicQuery.

property closed

Indicates whether this TopicQuery has been closed with close().

property datareader

Gets the DataReader associated to this TopicQuery.

static find(*reader:* rti.connextdds.IAnyDataReader, *guid:* rti.connextdds.Guid) → Optional[*rti.connextdds.TopicQuery*]

Lookup a TopicQuery by its GUID in the reader that createdit

property guid

The TopicQuery GUID.

retain(*self:* rti.connextdds.TopicQuery) → None

Disable automatic destruction of this TopicQuery.

static select_all(*reader:* rti.connextdds.IAnyDataReader) → *rti.connextdds.TopicQuery*

Create a TopicQuery that requests all data.

unretain(*self:* rti.connextdds.TopicQuery) → None

static use_reader_content_filter(*reader:* rti.connextdds.IAnyDataReader) → *rti.connextdds.TopicQuery*

Create a TopicQuery with a DataReader's content filter.

class rti.connextdds.**TopicQueryData**

Bases: pybind11_object

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

static create_from_service_request(*service_request:* rti.connextdds.ServiceRequest) → *rti.connextdds.TopicQueryData*

Creates a TopicQueryData from a ServiceRequest.

property original_related_reader_guid

Identifies the DataReader that created the TopicQuery.

property selection

The data selection.

property topic_name

The topic name of the DataReader.

class rti.connextdds.**TopicQueryDispatch**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.TopicQueryDispatch, *arg0:* rti.connextdds.TopicQueryDispatch) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicQueryDispatch) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.TopicQueryDispatch, enable: bool, publication_period: rti.connextdds.Duration, samples_per_period: int) -> None

Creates a policy with the provided values for enable, publication_period and samples_per_period.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.TopicQueryDispatch, *arg0:* rti.connextdds.TopicQueryDispatch) → bool

Test for inequality.

property enable

Allows this writer to dispatch TopicQueries.

property publication_period

The periodic interval at which samples are published.

property samples_per_period

The maximum number of samples to publish in each publication_period.

class rti.connextdds.**TopicQuerySelection**

Bases: pybind11_object

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicQuerySelection, filter: rti.connextdds.Filter) -> None

Creates a TopicQuerySelection.

2. __init__(self: rti.connextdds.TopicQuerySelection, filter: rti.connextdds.Filter, kind: rti.connextdds.TopicQuerySelectionKind) -> None

Creates a TopicQuerySelection with a selection kind.

__module__ = 'rti.connextdds'

property filter

The filter.

property kind

Indicates whether the sample selection is limited to cached samples or not.

class rti.connextdds.**TopicQuerySelectionKind**

Bases: pybind11_object

CONTINUOUS = <TopicQuerySelectionKind.CONTINUOUS: 1>

HISTORY_SNAPSHOT = <TopicQuerySelectionKind.HISTORY_SNAPSHOT: 0>

class TopicQuerySelectionKind

Bases: pybind11_object

Members:

HISTORY_SNAPSHOT : [default] Indicates that the TopicQuery may only select samples that were in the DataWriter cache upon reception.

CONTINUOUS : Indicates that the TopicQuery may continue selecting samples published after its reception.

The subscribing application must explicitly delete the TopicQuery (see TopicQuery.close()) to signal the DataWriters to stop publishing samples for it.

CONTINUOUS = <TopicQuerySelectionKind.CONTINUOUS: 1>

HISTORY_SNAPSHOT = <TopicQuerySelectionKind.HISTORY_SNAPSHOT: 0>

__eq__ (*self: object*, *other: object*) → bool

__getstate__ (*self: object*) → int

__hash__ (*self: object*) → int

__index__ (*self:* rti.connextdds.TopicQuerySelectionKind.TopicQuerySelectionKind) → int

__init__ (*self:* rti.connextdds.TopicQuerySelectionKind.TopicQuerySelectionKind, *value: int*) → None

__int__ (*self:* rti.connextdds.TopicQuerySelectionKind.TopicQuerySelectionKind) → int

__members__ = {'CONTINUOUS': <TopicQuerySelectionKind.CONTINUOUS: 1>, 'HISTORY_SNAPSHOT': <TopicQuerySelectionKind.HISTORY_SNAPSHOT: 0>}

__module__ = 'rti.connextdds'

__ne__ (*self: object*, *other: object*) → bool

__repr__ (*self: object*) → str

__setstate__ (*self:* rti.connextdds.TopicQuerySelectionKind.TopicQuerySelectionKind, *state: int*) → None

__str__ ()

name(self: handle) -> str

property name

property value

__eq__ (*self:* rti.connextdds.TopicQuerySelectionKind, *arg0:* rti.connextdds.TopicQuerySelectionKind) → bool

Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.TopicQuerySelectionKind, *arg0:* rti.connextdds.TopicQuerySelectionKind) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.TopicQuerySelectionKind, *arg0:* rti.connextdds.TopicQuerySelectionKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicQuerySelectionKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.TopicQuerySelectionKind, arg0: rti.connextdds.TopicQuerySelectionKind.TopicQuerySelectionKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.TopicQuerySelectionKind) → *rti.connextdds.TopicQuerySelectionKind.TopicQuerySelectionKind*

Int conversion.

__le__ (*self:* rti.connextdds.TopicQuerySelectionKind, *arg0:* rti.connextdds.TopicQuerySelectionKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.TopicQuerySelectionKind, *arg0:* rti.connextdds.TopicQuerySelectionKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TopicQuerySelectionKind, *arg0:* rti.connextdds.TopicQuerySelectionKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.TopicQuerySelectionKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**TopicSeq**

Bases: pybind11_object

__add__ (*self: List[*rti.connextdds.Topic*]*, *arg0: List[*rti.connextdds.Topic*]*) → List[*rti.connextdds.Topic*]

__bool__ (*self: List[*rti.connextdds.Topic*]*) → bool

Check whether the list is nonempty

__contains__ (*self: List[*rti.connextdds.Topic*]*, *x:* rti.connextdds.Topic) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: List[rti.connextdds.Topic], arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: List[rti.connextdds.Topic], arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self: List[*rti.connextdds.Topic*]*, *arg0: List[*rti.connextdds.Topic*]*) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: List[rti.connextdds.Topic], s: slice) -> List[rti.connextdds.Topic]

Retrieve list elements using a slice object

2. __getitem__(self: List[rti.connextdds.Topic], arg0: int) -> rti.connextdds.Topic

__hash__ = None

__iadd__(*self: List[*rti.connextdds.Topic*]*, *arg0: List[*rti.connextdds.Topic*]*) → List[*rti.connextdds.Topic*]

__imul__(*self: List[*rti.connextdds.Topic*]*, *arg0: int*) → List[*rti.connextdds.Topic*]

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TopicSeq) -> None

2. __init__(self: rti.connextdds.TopicSeq, arg0: List[rti.connextdds.Topic]) -> None

Copy constructor

3. __init__(self: rti.connextdds.TopicSeq, arg0: Iterable) -> None

__iter__(*self: List[*rti.connextdds.Topic*]*) → Iterator

__len__(*self: List[*rti.connextdds.Topic*]*) → int

__module__ = 'rti.connextdds'

__mul__(*self: List[*rti.connextdds.Topic*]*, *arg0: int*) → List[*rti.connextdds.Topic*]

__ne__(*self: List[*rti.connextdds.Topic*]*, *arg0: List[*rti.connextdds.Topic*]*) → bool

__rmul__(*self: List[*rti.connextdds.Topic*]*, *arg0: int*) → List[*rti.connextdds.Topic*]

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: List[rti.connextdds.Topic], arg0: int, arg1: rti.connextdds.Topic) -> None

2. __setitem__(self: List[rti.connextdds.Topic], arg0: slice, arg1: List[rti.connextdds.Topic]) -> None

Assign list elements using a slice object

append(*self: List[*rti.connextdds.Topic*]*, *x:* rti.connextdds.Topic) → None

Add an item to the end of the list

clear(*self: List[*rti.connextdds.Topic*]*) → None

Clear the contents

count (*self: List[*rti.connextdds.Topic*]*, *x:* rti.connextdds.Topic) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: List[rti.connextdds.Topic], L: List[rti.connextdds.Topic]) -> None

Extend the list by appending all the items in the given list

2. extend(self: List[rti.connextdds.Topic], L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self: List[*rti.connextdds.Topic*]*, *i: int*, *x:* rti.connextdds.Topic) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: List[rti.connextdds.Topic]) -> rti.connextdds.Topic

Remove and return the last item

2. pop(self: List[rti.connextdds.Topic], i: int) -> rti.connextdds.Topic

Remove and return the item at index `i`

remove (*self: List[*rti.connextdds.Topic*]*, *x:* rti.connextdds.Topic) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**TransportBuiltin**

Bases: `pybind11_object`

__eq__ (*self:* rti.connextdds.TransportBuiltin, *arg0:* rti.connextdds.TransportBuiltin) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TransportBuiltin) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.TransportBuiltin, mask: rti.connextdds.TransportBuiltinMask) -> None

Creates an instnace with the transport selection that the mask specifies.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TransportBuiltin, *arg0:* rti.connextdds.TransportBuiltin) → bool

Test for inequality.

all = <rti.connextdds.TransportBuiltin object>

property mask

The selected transports through a mask.

none = <rti.connextdds.TransportBuiltin object>

shmem = <rti.connextdds.TransportBuiltin object>

udpv4 = <rti.connextdds.TransportBuiltin object>

udpv6 = <rti.connextdds.TransportBuiltin object>

class rti.connextdds.**TransportBuiltinMask**

Bases: pybind11_object

ALL = 11111111

NONE = 00000000

SHMEM = 00000010

UDPv4 = 00000001

UDPv6 = 00001000

__and__(*self:* rti.connextdds.TransportBuiltinMask, *arg0:* rti.connextdds.TransportBuiltinMask) → *rti.connextdds.TransportBuiltinMask*

Bitwise logical AND of masks.

__bool__(*self:* rti.connextdds.TransportBuiltinMask) → bool

Test if any bits are set.

__contains__(*self:* rti.connextdds.TransportBuiltinMask, *arg0:* rti.connextdds.TransportBuiltinMask) → bool

__eq__(*self:* rti.connextdds.TransportBuiltinMask, *arg0:* rti.connextdds.TransportBuiltinMask) → bool

Compare masks for equality.

__getitem__(*self:* rti.connextdds.TransportBuiltinMask, *arg0: int*) → bool

Get individual mask bit.

__hash__ = None

__iand__(*self:* rti.connextdds.TransportBuiltinMask, *arg0:* rti.connextdds.TransportBuiltinMask) → *rti.connextdds.TransportBuiltinMask*

Set mask to logical AND with another mask.

__ilshift__(*self:* rti.connextdds.TransportBuiltinMask, *arg0: int*) → *rti.connextdds.TransportBuiltinMask*

Left shift bits in mask.

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TransportBuiltinMask) -> None

Create a TransportBuiltinMask equivalent to TransportBuiltinMask.NONE

2. __init__(self: rti.connextdds.TransportBuiltinMask, value: int) -> None

Creates a mask from the bits in an integer.

__int__ (*self:* rti.connextdds.TransportBuiltinMask) → int

Convert mask to int.

__ior__ (*self:* rti.connextdds.TransportBuiltinMask, *arg0:* rti.connextdds.TransportBuiltinMask) → *rti.connextdds.TransportBuiltinMask*

Set mask to logical OR with another mask.

__irshift__ (*self:* rti.connextdds.TransportBuiltinMask, *arg0: int*) → *rti.connextdds.TransportBuiltinMask*

Right shift bits in mask.

__ixor__ (*self:* rti.connextdds.TransportBuiltinMask, *arg0:* rti.connextdds.TransportBuiltinMask) → *rti.connextdds.TransportBuiltinMask*

Set mask to logical XOR with another mask.

__lshift__ (*self:* rti.connextdds.TransportBuiltinMask, *arg0: int*) → *rti.connextdds.TransportBuiltinMask*

Left shift bits in mask.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TransportBuiltinMask, *arg0:* rti.connextdds.TransportBuiltinMask) → bool

Compare masks for inequality.

__or__ (*self:* rti.connextdds.TransportBuiltinMask, *arg0:* rti.connextdds.TransportBuiltinMask) → *rti.connextdds.TransportBuiltinMask*

Bitwise logical OR of masks.

__repr__ (*self:* rti.connextdds.TransportBuiltinMask) → str

Convert mask to string.

__rshift__ (*self:* rti.connextdds.TransportBuiltinMask, *arg0: int*) → *rti.connextdds.TransportBuiltinMask*

Right shift bits in mask.

__setitem__ (*self:* rti.connextdds.TransportBuiltinMask, *arg0: int*, *arg1: bool*) → None

Set individual mask bit

__str__ (*self:* rti.connextdds.TransportBuiltinMask) → str

Convert mask to string.

__xor__ (*self:* rti.connextdds.TransportBuiltinMask, *arg0:* rti.connextdds.TransportBuiltinMask) → *rti.connextdds.TransportBuiltinMask*

Bitwise logical XOR of masks.

property count

Returns the number of bits set in the mask.

flip (**args*, ***kwargs*)

Overloaded function.

1. flip(self: rti.connextdds.TransportBuiltinMask) -> rti.connextdds.TransportBuiltinMask

Flip all bits in the mask.

2. flip(self: rti.connextdds.TransportBuiltinMask, pos: int) -> rti.connextdds.TransportBuiltinMask

Flip the mask bit at the specified position.

reset (**args*, ***kwargs*)

Overloaded function.

1. reset(self: rti.connextdds.TransportBuiltinMask) -> rti.connextdds.TransportBuiltinMask

Clear all bits in the mask.

2. reset(self: rti.connextdds.TransportBuiltinMask, pos: int) -> rti.connextdds.TransportBuiltinMask

Clear the mask bit at the specified position.

set (**args*, ***kwargs*)

Overloaded function.

1. set(self: rti.connextdds.TransportBuiltinMask) -> rti.connextdds.TransportBuiltinMask

Set all bits in the mask.

2. set(self: rti.connextdds.TransportBuiltinMask, pos: int, value: bool = True) -> rti.connextdds.TransportBuiltinMask

Set the mask bit at the specified position to the provided value (default: true).

property size

Returns the number of bits in the mask type.

test (*self:* rti.connextdds.TransportBuiltinMask, *pos: int*) → bool

Test whether the mask bit at position "pos" is set.

test_all (*self:* rti.connextdds.TransportBuiltinMask) → bool

Test if all bits are set.

test_any (*self:* rti.connextdds.TransportBuiltinMask) → bool

Test if any bits are set.

test_none(*self:* rti.connextdds.TransportBuiltinMask) → bool

Test if none of the bits are set.

class rti.connextdds.**TransportClassId**

Bases: pybind11_object

ANY = <TransportClassId.ANY: 0>

INTRA = <TransportClassId.INTRA: 3>

INVALID = <TransportClassId.INVALID: -1>

RESERVED_RANGE = <TransportClassId.RESERVED_RANGE: 1000>

SHMEM = <TransportClassId.SHMEM: 16777216>

SHMEM_510 = <TransportClassId.SHMEM_510: 2>

TCPV4_LAN = <TransportClassId.TCPV4_LAN: 8>

TCPV4_WAN = <TransportClassId.TCPV4_WAN: 9>

TLSV4_LAN = <TransportClassId.TLSV4_LAN: 10>

TLSV4_WAN = <TransportClassId.TLSV4_WAN: 11>

class TransportClassId

Bases: pybind11_object

Members:

INVALID : Invalid.

ANY : Any.

UDPv4 : UDPv4.

UDPv4_WAN : UDPv4 WAN.

SHMEM : Shared memory.

SHMEM_510 : Shared Memory from 5.1.0 and earlier.

INTRA : Intra-participant.

UDPv6 : UDPv6.

UDPv6_510 : UDPv6 for 5.1.0 and earlier.

TCPV4_LAN : TCPv4 LAN mode.

TCPV4_WAN : TCPv4 WAN mode.

TLSV4_LAN : TCPv4 LAN mode with TLS.

TLSV4_WAN : TCPv4 WAN mode with TLS.

RESERVED_RANGE : eserved for additional user-defined transport plugins.

ANY = <TransportClassId.ANY: 0>

INTRA = <TransportClassId.INTRA: 3>

INVALID = <TransportClassId.INVALID: -1>

RESERVED_RANGE = <TransportClassId.RESERVED_RANGE: 1000>

SHMEM = <TransportClassId.SHMEM: 16777216>

SHMEM_510 = <TransportClassId.SHMEM_510: 2>

TCPV4_LAN = <TransportClassId.TCPV4_LAN: 8>

TCPV4_WAN = <TransportClassId.TCPV4_WAN: 9>

TLSV4_LAN = <TransportClassId.TLSV4_LAN: 10>

TLSV4_WAN = <TransportClassId.TLSV4_WAN: 11>

UDPv4 = <TransportClassId.UDPv4: 1>

UDPv4_WAN = <TransportClassId.UDPv4_WAN: 16777217>

UDPv6 = <TransportClassId.SHMEM_510: 2>

UDPv6_510 = <TransportClassId.UDPv6_510: 5>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.TransportClassId.TransportClassId) → int

__init__(*self:* rti.connextdds.TransportClassId.TransportClassId, *value: int*) → None

__int__(*self:* rti.connextdds.TransportClassId.TransportClassId) → int

__members__ = {'ANY': <TransportClassId.ANY: 0>, 'INTRA': <TransportClassId.INTRA: 3>, 'INVALID': <TransportClassId.INVALID: -1>, 'RESERVED_RANGE': <TransportClassId.RESERVED_RANGE: 1000>, 'SHMEM': <TransportClassId.SHMEM: 16777216>, 'SHMEM_510': <TransportClassId.SHMEM_510: 2>, 'TCPV4_LAN': <TransportClassId.TCPV4_LAN: 8>, 'TCPV4_WAN': <TransportClassId.TCPV4_WAN: 9>, 'TLSV4_LAN': <TransportClassId.TLSV4_LAN: 10>, 'TLSV4_WAN': <TransportClassId.TLSV4_WAN: 11>, 'UDPv4': <TransportClassId.UDPv4: 1>, 'UDPv4_WAN': <TransportClassId.UDPv4_WAN: 16777217>, 'UDPv6': <TransportClassId.SHMEM_510: 2>, 'UDPv6_510': <TransportClassId.UDPv6_510: 5>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.TransportClassId.TransportClassId, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

UDPv4 = <TransportClassId.UDPv4: 1>

UDPv4_WAN = <TransportClassId.UDPv4_WAN: 16777217>

UDPv6 = <TransportClassId.SHMEM_510: 2>

UDPv6_510 = <TransportClassId.UDPv6_510: 5>

__eq__(*self:* rti.connextdds.TransportClassId, *arg0:* rti.connextdds.TransportClassId) → bool

Apply operator to underlying enumerated values.

__ge__(*self:* rti.connextdds.TransportClassId, *arg0:* rti.connextdds.TransportClassId) → bool

Apply operator to underlying enumerated values.

__gt__(*self:* rti.connextdds.TransportClassId, *arg0:* rti.connextdds.TransportClassId) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TransportClassId) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.TransportClassId, arg0: rti.connextdds.TransportClassId.TransportClassId) -> None

Copy constructor.

__int__(*self:* rti.connextdds.TransportClassId) → *rti.connextdds.TransportClassId.TransportClassId*

Int conversion.

__le__(*self:* rti.connextdds.TransportClassId, *arg0:* rti.connextdds.TransportClassId) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.TransportClassId, *arg0:* rti.connextdds.TransportClassId) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TransportClassId, *arg0:* rti.connextdds.TransportClassId) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.TransportClassId) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**TransportInfo**

Bases: pybind11_object

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TransportInfo) -> None

Creates an instance with default values.

2. __init__(self: rti.connextdds.TransportInfo, class_id: rti.connextdds.TransportClassId.TransportClassId, message_size_max: int) -> None

Creates an instance with the specified class id and max size.

__module__ = 'rti.connextdds'

property class_id

The class_id identifies the transport associated with the message_size_max.

property message_size_max

The maximum size of an RTPS message in bytes that can be sent or received by the transport plugin identified by the class_id.

class rti.connextdds.**TransportInfoSeq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.TransportInfoSeq, *arg0:* rti.connextdds.TransportInfoSeq) → *rti.connextdds.TransportInfoSeq*

__bool__ (*self:* rti.connextdds.TransportInfoSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.TransportInfoSeq, *x:* rti.connextdds.TransportInfo) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.TransportInfoSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.TransportInfoSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.TransportInfoSeq, *arg0:* rti.connextdds.TransportInfoSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.TransportInfoSeq, s: slice) -> rti.connextdds.TransportInfoSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.TransportInfoSeq, arg0: int) -> rti.connextdds.TransportInfo

__hash__ = None

__iadd__(*self:* rti.connextdds.TransportInfoSeq, *arg0:* rti.connextdds.TransportInfoSeq) → *rti.connextdds.TransportInfoSeq*

__imul__(*self:* rti.connextdds.TransportInfoSeq, *arg0: int*) → *rti.connextdds.TransportInfoSeq*

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TransportInfoSeq) -> None

2. __init__(self: rti.connextdds.TransportInfoSeq, arg0: rti.connextdds.TransportInfoSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.TransportInfoSeq, arg0: Iterable) -> None

__iter__(*self:* rti.connextdds.TransportInfoSeq) → Iterator

__len__(*self:* rti.connextdds.TransportInfoSeq) → int

__module__ = 'rti.connextdds'

__mul__(*self:* rti.connextdds.TransportInfoSeq, *arg0: int*) → *rti.connextdds.TransportInfoSeq*

__ne__(*self:* rti.connextdds.TransportInfoSeq, *arg0:* rti.connextdds.TransportInfoSeq) → bool

__rmul__(*self:* rti.connextdds.TransportInfoSeq, *arg0: int*) → *rti.connextdds.TransportInfoSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.TransportInfoSeq, arg0: int, arg1: rti.connextdds.TransportInfo) -> None

2. __setitem__(self: rti.connextdds.TransportInfoSeq, arg0: slice, arg1: rti.connextdds.TransportInfoSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.TransportInfoSeq, *x:* rti.connextdds.TransportInfo) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.TransportInfoSeq) → None

Clear the contents

count (*self:* rti.connextdds.TransportInfoSeq, *x:* rti.connextdds.TransportInfo) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.TransportInfoSeq, L: rti.connextdds.TransportInfoSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.TransportInfoSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.TransportInfoSeq, *i: int*, *x:* rti.connextdds.TransportInfo) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.TransportInfoSeq) -> rti.connextdds.TransportInfo

Remove and return the last item

2. pop(self: rti.connextdds.TransportInfoSeq, i: int) -> rti.connextdds.TransportInfo

Remove and return the item at index `i`

remove (*self:* rti.connextdds.TransportInfoSeq, *x:* rti.connextdds.TransportInfo) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**TransportInfoVector**

Bases: `pybind11_object`

A DDS standard container with functionality similar to a C++ vector.

__eq__ (*self:* rti.connextdds.TransportInfoVector, *arg0:* rti.connextdds.TransportInfoVector) → bool

Compare TransportInfoVectors for equality.

__getitem__ (*self:* rti.connextdds.TransportInfoVector, *arg0: int*) → *rti.connextdds.TransportInfo*

Get the value at the specified index.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TransportInfoVector) -> None

Create an empty TransportInfoVector

2. __init__(self: rti.connextdds.TransportInfoVector, size: int) -> None

Create a TransportInfoVector with a preallocated size.

3. __init__(self: rti.connextdds.TransportInfoVector, vector: rti.connextdds.TransportInfoVector) -> None

Create a copy from another TransportInfoVector.

4. __init__(self: rti.connextdds.TransportInfoVector, arg0: Iterable) -> None

5. __init__(self: rti.connextdds.TransportInfoVector, list: rti.connextdds.TransportInfoSeq) -> None

Create a TransportInfoVector from a list of values.

__iter__ (*self:* rti.connextdds.TransportInfoVector) → Iterator

Iterate over the contents of the vector.

__len__ (*self:* rti.connextdds.TransportInfoVector) → int

Get the length of the TransportInfoVector.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TransportInfoVector, *arg0:* rti.connextdds.TransportInfoVector) → bool

Compare TransportInfoVectors for inequality.

__setitem__ (*self:* rti.connextdds.TransportInfoVector, *arg0: int*, *arg1:* rti.connextdds.TransportInfo) → None

Set the value at the specified index.

clear (*self:* rti.connextdds.TransportInfoVector) → None

Resize TransportInfoVector to 0.

resize (*self:* rti.connextdds.TransportInfoVector, *size: int*) → None

Resize TransportInfoVector.

class rti.connextdds.**TransportMulticast**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.TransportMulticast, *arg0:* rti.connextdds.TransportMulticast) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TransportMulticast) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.TransportMulticast, value: rti.connextdds.TransportMulticastSettingsSeq, kind: rti.connextdds.TransportMulticastKind) -> None

Creates an instance with the speicfied multicast settings.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.TransportMulticast, *arg0:* rti.connextdds.TransportMulticast) → bool

Test for inequality.

property kind

A value that specifies a way to determine how to obtain the multicast address.

property value

A sequence of multicast communications settings.

class rti.connextdds.**TransportMulticastKind**

Bases: pybind11_object

AUTOMATIC = <TransportMulticastKind.AUTOMATIC: 0>

class TransportMulticastKind

Bases: pybind11_object

Members:

AUTOMATIC : Selects the multicast address automatically.

NOTE: This setting is required when using the TransportMulticastMapping.

UNICAST : Selects a unicast-only mode.

AUTOMATIC = <TransportMulticastKind.AUTOMATIC: 0>

UNICAST = <TransportMulticastKind.UNICAST: 1>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.TransportMulticastKind.TransportMulticastKind) → int

__init__(*self:* rti.connextdds.TransportMulticastKind.TransportMulticastKind, *value: int*) → None

__int__(*self:* rti.connextdds.TransportMulticastKind.TransportMulticastKind) → int

__members__ = {'AUTOMATIC': <TransportMulticastKind.AUTOMATIC: 0>, 'UNICAST': <TransportMulticastKind.UNICAST: 1>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.TransportMulticastKind.TransportMulticastKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

UNICAST = <TransportMulticastKind.UNICAST: 1>

__eq__(*self:* rti.connextdds.TransportMulticastKind, *arg0:* rti.connextdds.TransportMulticastKind) → bool

Apply operator to underlying enumerated values.

__ge__(*self:* rti.connextdds.TransportMulticastKind, *arg0:* rti.connextdds.TransportMulticastKind) → bool

Apply operator to underlying enumerated values.

__gt__(*self:* rti.connextdds.TransportMulticastKind, *arg0:* rti.connextdds.TransportMulticastKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TransportMulticastKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.TransportMulticastKind, arg0: rti.connextdds.TransportMulticastKind.TransportMulticastKind) -> None

Copy constructor.

__int__(*self:* rti.connextdds.TransportMulticastKind) → *rti.connextdds.TransportMulticastKind.TransportMulticastKind*

Int conversion.

__le__(*self:* rti.connextdds.TransportMulticastKind, *arg0:* rti.connextdds.TransportMulticastKind) → bool

Apply operator to underlying enumerated values.

__lt__(*self:* rti.connextdds.TransportMulticastKind, *arg0:* rti.connextdds.TransportMulticastKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TransportMulticastKind, *arg0:* rti.connextdds.TransportMulticastKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.TransportMulticastKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**TransportMulticastMapping**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.TransportMulticastMapping, *arg0:* rti.connextdds.TransportMulticastMapping) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TransportMulticastMapping) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.TransportMulticastMapping, mappings: rti.connextdds.MulticastMappingSeq) -> None

Creates an object with with specified mappings.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TransportMulticastMapping, *arg0:* rti.connextdds.TransportMulticastMapping) → bool

Test for inequality.

property value

A sequence of transport multicast mappings.

class rti.connextdds.**TransportMulticastMappingFunction**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.TransportMulticastMappingFunction, *arg0:* rti.connextdds.TransportMulticastMappingFunction) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.TransportMulticastMappingFunction, *dll: str*, *function_name: str*) → None

Create a mapping function instance.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TransportMulticastMappingFunction, *arg0:* rti.connextdds.TransportMulticastMappingFunction) → bool

Test for inequality.

property dll

The name of the dynamic library containing the mapping function.

property function_name

The name of the mapping function.

class rti.connextdds.**TransportMulticastSeq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.TransportMulticastSeq, *arg0:* rti.connextdds.TransportMulticastSeq) → *rti.connextdds.TransportMulticastSeq*

__bool__ (*self:* rti.connextdds.TransportMulticastSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.TransportMulticastSeq, *x:* rti.connextdds.TransportMulticast) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.TransportMulticastSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.TransportMulticastSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.TransportMulticastSeq, *arg0:* rti.connextdds.TransportMulticastSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.TransportMulticastSeq, s: slice) -> rti.connextdds.TransportMulticastSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.TransportMulticastSeq, arg0: int) -> rti.connextdds.TransportMulticast

__hash__ = None

__iadd__ (*self:* rti.connextdds.TransportMulticastSeq, *arg0:* rti.connextdds.TransportMulticastSeq) → *rti.connextdds.TransportMulticastSeq*

__imul__ (*self:* rti.connextdds.TransportMulticastSeq, *arg0: int*) → *rti.connextdds.TransportMulticastSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TransportMulticastSeq) -> None
2. __init__(self: rti.connextdds.TransportMulticastSeq, arg0: rti.connextdds.TransportMulticastSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.TransportMulticastSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.TransportMulticastSeq) → Iterator

__len__ (*self:* rti.connextdds.TransportMulticastSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.TransportMulticastSeq, *arg0: int*) → *rti.connextdds.TransportMulticastSeq*

__ne__ (*self:* rti.connextdds.TransportMulticastSeq, *arg0:* rti.connextdds.TransportMulticastSeq) → bool

__rmul__ (*self:* rti.connextdds.TransportMulticastSeq, *arg0: int*) → *rti.connextdds.TransportMulticastSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.TransportMulticastSeq, arg0: int, arg1: rti.connextdds.TransportMulticast) -> None
2. __setitem__(self: rti.connextdds.TransportMulticastSeq, arg0: slice, arg1: rti.connextdds.TransportMulticastSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.TransportMulticastSeq, *x:* rti.connextdds.TransportMulticast) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.TransportMulticastSeq) → None

Clear the contents

count (*self:* rti.connextdds.TransportMulticastSeq, *x:* rti.connextdds.TransportMulticast) → int

Return the number of times x appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.TransportMulticastSeq, L: rti.connextdds.TransportMulticastSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.TransportMulticastSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert(*self:* rti.connextdds.TransportMulticastSeq, *i: int*, *x:* rti.connextdds.TransportMulticast) → None

Insert an item at a given position.

pop(**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.TransportMulticastSeq) -> rti.connextdds.TransportMulticast

Remove and return the last item

2. pop(self: rti.connextdds.TransportMulticastSeq, i: int) -> rti.connextdds.TransportMulticast

Remove and return the item at index `i`

remove(*self:* rti.connextdds.TransportMulticastSeq, *x:* rti.connextdds.TransportMulticast) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**TransportMulticastSettings**

Bases: `pybind11_object`

__eq__(*self:* rti.connextdds.TransportMulticastSettings, *arg0:* rti.connextdds.TransportMulticastSettings) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.TransportMulticastSettings, *transports:* rti.connextdds.StringSeq, *receive_address: str*, *receive_port: int*) → None

Creates an instance with the specified transport aliases, receive address and receive port.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.TransportMulticastSettings, *arg0:* rti.connextdds.TransportMulticastSettings) → bool

Test for inequality.

property receive_address

The multicast group address on which the entity can receive data.

property receive_port

The multicast port on which the entity can receive data.

property transports

A sequence of transport aliases that specifies the transports on which to receive multicast traffic for the entity.

This property's getter returns a deep copy.

class rti.connextdds.**TransportMulticastSettingsSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.TransportMulticastSettingsSeq, *arg0:* rti.connextdds.TransportMulticastSettingsSeq) → *rti.connextdds.TransportMulticastSettingsSeq*

__bool__(*self:* rti.connextdds.TransportMulticastSettingsSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.TransportMulticastSettingsSeq, *x:* rti.connextdds.TransportMulticastSettings) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.TransportMulticastSettingsSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.TransportMulticastSettingsSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.TransportMulticastSettingsSeq, *arg0:* rti.connextdds.TransportMulticastSettingsSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.TransportMulticastSettingsSeq, s: slice) -> rti.connextdds.TransportMulticastSettingsSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.TransportMulticastSettingsSeq, arg0: int) -> rti.connextdds.TransportMulticastSettings

__hash__ = None

__iadd__(*self:* rti.connextdds.TransportMulticastSettingsSeq, *arg0:* rti.connextdds.TransportMulticastSettingsSeq) → *rti.connextdds.TransportMulticastSettingsSeq*

__imul__(*self:* rti.connextdds.TransportMulticastSettingsSeq, *arg0: int*) → *rti.connextdds.TransportMulticastSettingsSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TransportMulticastSettingsSeq) -> None

2. __init__(self: rti.connextdds.TransportMulticastSettingsSeq, arg0: rti.connextdds.TransportMulticastSettingsSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.TransportMulticastSettingsSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.TransportMulticastSettingsSeq) → Iterator

__len__ (*self:* rti.connextdds.TransportMulticastSettingsSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.TransportMulticastSettingsSeq, *arg0: int*) → *rti.connextdds.TransportMulticastSettingsSeq*

__ne__ (*self:* rti.connextdds.TransportMulticastSettingsSeq, *arg0:* rti.connextdds.TransportMulticastSettingsSeq) → bool

__rmul__ (*self:* rti.connextdds.TransportMulticastSettingsSeq, *arg0: int*) → *rti.connextdds.TransportMulticastSettingsSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.TransportMulticastSettingsSeq, arg0: int, arg1: rti.connextdds.TransportMulticastSettings) -> None

2. __setitem__(self: rti.connextdds.TransportMulticastSettingsSeq, arg0: slice, arg1: rti.connextdds.TransportMulticastSettingsSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.TransportMulticastSettingsSeq, *x:* rti.connextdds.TransportMulticastSettings) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.TransportMulticastSettingsSeq) → None

Clear the contents

count (*self:* rti.connextdds.TransportMulticastSettingsSeq, *x:* rti.connextdds.TransportMulticastSettings) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.TransportMulticastSettingsSeq, L: rti.connextdds.TransportMulticastSettingsSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.TransportMulticastSettingsSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.TransportMulticastSettingsSeq, *i: int*, *x:* rti.connextdds.TransportMulticastSettings) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.TransportMulticastSettingsSeq) -> rti.connextdds.TransportMulticastSettings

Remove and return the last item

2. pop(self: rti.connextdds.TransportMulticastSettingsSeq, i: int) -> rti.connextdds.TransportMulticastSettings

Remove and return the item at index `i`

remove (*self:* rti.connextdds.TransportMulticastSettingsSeq, *x:* rti.connextdds.TransportMulticastSettings) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**TransportPriority**

Bases: `pybind11_object`

__eq__ (*self:* rti.connextdds.TransportPriority, *arg0:* rti.connextdds.TransportPriority) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TransportPriority) -> None

Creates a policy with priority 0.

2. __init__(self: rti.connextdds.TransportPriority, priority: int) -> None

Creates a policy with the specified priority.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TransportPriority, *arg0:* rti.connextdds.TransportPriority) → bool

Test for inequality.

property value

The priority.

class rti.connextdds.**TransportSelection**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.TransportSelection, *arg0:* rti.connextdds.TransportSelection) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TransportSelection) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.TransportSelection, enabled_transports: rti.connextdds.StringSeq) -> None

Creates an instance with the specified transport aliases.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.TransportSelection, *arg0:* rti.connextdds.TransportSelection) → bool

Test for inequality.

property value

A sequence of transport aliases that specifies the transport instances available for use by the entity.

This property's getter returns a deep copy.

class rti.connextdds.**TransportUnicast**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.TransportUnicast, *arg0:* rti.connextdds.TransportUnicast) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TransportUnicast) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.TransportUnicast, settings: rti.connextdds.TransportUnicastSettingsSeq) -> None

Creates an instance with the specified settings.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.TransportUnicast, *arg0:* rti.connextdds.TransportUnicast) → bool

Test for inequality.

property value

The unicast settings.

This property's getter returns a deep copy.

class rti.connextdds.**TransportUnicastSettings**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.TransportUnicastSettings, *arg0:* rti.connextdds.TransportUnicastSettings) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TransportUnicastSettings) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.TransportUnicastSettings, transports: rti.connextdds.StringSeq, receive_port: int = 0) -> None

Creates the default policy.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.TransportUnicastSettings, *arg0:* rti.connextdds.TransportUnicastSettings) → bool

Test for inequality.

property receive_port

The unicast port on which the entity can receive data.

property transports

A sequence of transport aliases that specifies the unicast interfaces on which to receive unicast traffic for the entity.

This property's getter returns a deep copy.

class rti.connextdds.**TransportUnicastSettingsSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.TransportUnicastSettingsSeq, *arg0:* rti.connextdds.TransportUnicastSettingsSeq) → *rti.connextdds.TransportUnicastSettingsSeq*

__bool__(*self:* rti.connextdds.TransportUnicastSettingsSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.TransportUnicastSettingsSeq, *x:* rti.connextdds.TransportUnicastSettings) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.TransportUnicastSettingsSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.TransportUnicastSettingsSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.TransportUnicastSettingsSeq, *arg0:* rti.connextdds.TransportUnicastSettingsSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.TransportUnicastSettingsSeq, s: slice) -> rti.connextdds.TransportUnicastSettingsSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.TransportUnicastSettingsSeq, arg0: int) -> rti.connextdds.TransportUnicastSettings

__hash__ = None

__iadd__ (*self:* rti.connextdds.TransportUnicastSettingsSeq, *arg0:* rti.connextdds.TransportUnicastSettingsSeq) → *rti.connextdds.TransportUnicastSettingsSeq*

__imul__ (*self:* rti.connextdds.TransportUnicastSettingsSeq, *arg0: int*) → *rti.connextdds.TransportUnicastSettingsSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TransportUnicastSettingsSeq) -> None

2. __init__(self: rti.connextdds.TransportUnicastSettingsSeq, arg0: rti.connextdds.TransportUnicastSettingsSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.TransportUnicastSettingsSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.TransportUnicastSettingsSeq) → Iterator

__len__ (*self:* rti.connextdds.TransportUnicastSettingsSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.TransportUnicastSettingsSeq, *arg0: int*) → *rti.connextdds.TransportUnicastSettingsSeq*

__ne__ (*self:* rti.connextdds.TransportUnicastSettingsSeq, *arg0:* rti.connextdds.TransportUnicastSettingsSeq) → bool

__rmul__ (*self:* rti.connextdds.TransportUnicastSettingsSeq, *arg0: int*) → *rti.connextdds.TransportUnicastSettingsSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.TransportUnicastSettingsSeq, arg0: int, arg1: rti.connextdds.TransportUnicastSettings) -> None

2. __setitem__(self: rti.connextdds.TransportUnicastSettingsSeq, arg0: slice, arg1: rti.connextdds.TransportUnicastSettingsSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.TransportUnicastSettingsSeq, *x:* rti.connextdds.TransportUnicastSettings) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.TransportUnicastSettingsSeq) → None

Clear the contents

count (*self:* rti.connextdds.TransportUnicastSettingsSeq, *x:* rti.connextdds.TransportUnicastSettings) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.TransportUnicastSettingsSeq, L: rti.connextdds.TransportUnicastSettingsSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.TransportUnicastSettingsSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.TransportUnicastSettingsSeq, *i: int*, *x:* rti.connextdds.TransportUnicastSettings) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.TransportUnicastSettingsSeq) -> rti.connextdds.TransportUnicastSettings

Remove and return the last item

2. pop(self: rti.connextdds.TransportUnicastSettingsSeq, i: int) -> rti.connextdds.TransportUnicastSettings

Remove and return the item at index i

remove (*self:* rti.connextdds.TransportUnicastSettingsSeq, *x:* rti.connextdds.TransportUnicastSettings) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**TriggeredConditions**

Bases: pybind11_object

__contains__ (*self:* rti.connextdds.TriggeredConditions, *arg0:* rti.connextdds.ICondition) → bool

__getitem__ (*self:* rti.connextdds.TriggeredConditions, *arg0: int*) → *rti.connextdds.Condition*

__init__ (**args*, ***kwargs*)

__iter__ (*self:* rti.connextdds.TriggeredConditions) → *rti.connextdds.TriggeredConditionsIterator*

__len__ (*self:* rti.connextdds.TriggeredConditions) → int

__module__ = 'rti.connextdds'

__reversed__ (*self:* rti.connextdds.TriggeredConditions) → *rti.connextdds.TriggeredConditionsIterator*

class rti.connextdds.**TriggeredConditionsIterator**

Bases: pybind11_object

__init__ (**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__next__ (*self:* rti.connextdds.TriggeredConditionsIterator) → *rti.connextdds.Condition*

Get next triggered condition.

class rti.connextdds.**TypeConsistencyEnforcement**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.TypeConsistencyEnforcement, *arg0:* rti.connextdds.TypeConsistencyEnforcement) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TypeConsistencyEnforcement) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.TypeConsistencyEnforcement, kind: rti.connextdds.TypeConsistencyEnforcementKind) -> None

Creates an instance with a specific enforcement kind.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TypeConsistencyEnforcement, *arg0:* rti.connextdds.TypeConsistencyEnforcement) → bool

Test for inequality.

allow_type_coercion = <rti.connextdds.TypeConsistencyEnforcement object>

auto_type_coercion = <rti.connextdds.TypeConsistencyEnforcement object>

disallow_type_coercion = <rti.connextdds.TypeConsistencyEnforcement object>

property force_type_validation

Controls whether type validation is forced.

property ignore_enum_literal_names

Controls whether enumeration literal names are ignored.

property ignore_member_names

Controls whether member names are ignored.

property ignore_sequence_bounds

Controls whether sequence bounds are ignored.

property ignore_string_bounds

Controls whether string bounds are ignored.

property kind

The enforcement kind.

property prevent_type_widening

Controls whether type widening is prevented.

class rti.connextdds.**TypeConsistencyEnforcementKind**

Bases: pybind11_object

ALLOW_TYPE_COERCION = <TypeConsistencyEnforcementKind.ALLOW_TYPE_COERCION: 1>

AUTO_TYPE_COERCION = <TypeConsistencyEnforcementKind.AUTO_TYPE_COERCION: 2>

DISALLOW_TYPE_COERCION = <TypeConsistencyEnforcementKind.DISALLOW_TYPE_COERCION: 0>

class TypeConsistencyEnforcementKind

Bases: pybind11_object

Members:

DISALLOW_TYPE_COERCION : The DataWriter and the DataReader must support the same data type in order for them to communicate.

ALLOW_TYPE_COERCION : The DataWriter and the DataReader need not support the same data type in order for them to communicate as long as the DataReader's type is assignable from the DataWriter's type.

AUTO_TYPE_COERCION : [default] This AUTO value will be applied as TypeConsistencyKind.DISALLOW_TYPE_COERCION when the data type is annotated with @transfer_mode(SHMEM_REF) while using C, Traditional C++, or Modern C++ APIs. In all other cases, this AUTO value will be applied as TypeConsistencyKind.ALLOW_TYPE_COERCION.

ALLOW_TYPE_COERCION =
<TypeConsistencyEnforcementKind.ALLOW_TYPE_COERCION: 1>

AUTO_TYPE_COERCION =
<TypeConsistencyEnforcementKind.AUTO_TYPE_COERCION: 2>

DISALLOW_TYPE_COERCION =
<TypeConsistencyEnforcementKind.DISALLOW_TYPE_COERCION: 0>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.TypeConsistencyEnforcementKind.TypeConsistencyEnforcementKind) → int

__init__(*self:* rti.connextdds.TypeConsistencyEnforcementKind.TypeConsistencyEnforcementKind, *value: int*) → None

__int__(*self:* rti.connextdds.TypeConsistencyEnforcementKind.TypeConsistencyEnforcementKind) → int

__members__ = {'ALLOW_TYPE_COERCION':
<TypeConsistencyEnforcementKind.ALLOW_TYPE_COERCION: 1>,
'AUTO_TYPE_COERCION':
<TypeConsistencyEnforcementKind.AUTO_TYPE_COERCION: 2>,
'DISALLOW_TYPE_COERCION':
<TypeConsistencyEnforcementKind.DISALLOW_TYPE_COERCION: 0>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__ (*self:* rti.connextdds.TypeConsistencyEnforcementKind.TypeConsistencyEnforcementKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

__eq__ (*self:* rti.connextdds.TypeConsistencyEnforcementKind, *arg0:* rti.connextdds.TypeConsistencyEnforcementKind) → bool

Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.TypeConsistencyEnforcementKind, *arg0:* rti.connextdds.TypeConsistencyEnforcementKind) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.TypeConsistencyEnforcementKind, *arg0:* rti.connextdds.TypeConsistencyEnforcementKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TypeConsistencyEnforcementKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.TypeConsistencyEnforcementKind, arg0: rti.connextdds.TypeConsistencyEnforcementKind.TypeConsistencyEnforcementKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.TypeConsistencyEnforcementKind) → *rti.connextdds.TypeConsistencyEnforcementKind.TypeConsistencyEnforcementKind*

Int conversion.

__le__ (*self:* rti.connextdds.TypeConsistencyEnforcementKind, *arg0:* rti.connextdds.TypeConsistencyEnforcementKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.TypeConsistencyEnforcementKind, *arg0:* rti.connextdds.TypeConsistencyEnforcementKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TypeConsistencyEnforcementKind, *arg0:* rti.connextdds.TypeConsistencyEnforcementKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.TypeConsistencyEnforcementKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**TypeKind**

Bases: pybind11_object

AGGREGATION_TYPE = <TypeKind.AGGREGATION_TYPE: 256>

ALIAS_TYPE = <TypeKind.ALIAS_TYPE: 32771>

ANNOTATION_TYPE = <TypeKind.ANNOTATION_TYPE: 128>

ARRAY_TYPE = <TypeKind.ARRAY_TYPE: 33284>

BITSET_TYPE = <TypeKind.BITSET_TYPE: 32770>

BOOLEAN_TYPE = <TypeKind.BOOLEAN_TYPE: 16385>

CHAR_16_TYPE = <TypeKind.CHAR_16_TYPE: 16397>

CHAR_8_TYPE = <TypeKind.CHAR_8_TYPE: 16396>

COLLECTION_TYPE = <TypeKind.COLLECTION_TYPE: 512>

CONSTRUCTED_TYPE = <TypeKind.CONSTRUCTED_TYPE: 32768>

ENUMERATION_TYPE = <TypeKind.ENUMERATION_TYPE: 32769>

FLOAT_128_TYPE = <TypeKind.FLOAT_128_TYPE: 16395>

FLOAT_32_TYPE = <TypeKind.FLOAT_32_TYPE: 16393>

FLOAT_64_TYPE = <TypeKind.FLOAT_64_TYPE: 16394>

INT_16_TYPE = <TypeKind.INT_16_TYPE: 16387>

INT_32_TYPE = <TypeKind.INT_32_TYPE: 16389>

INT_64_TYPE = <TypeKind.INT_64_TYPE: 16391>

MAP_TYPE = <TypeKind.MAP_TYPE: 33287>

NO_TYPE = <TypeKind.NO_TYPE: 0>

PRIMITIVE_TYPE = <TypeKind.PRIMITIVE_TYPE: 16384>

SEQUENCE_TYPE = <TypeKind.SEQUENCE_TYPE: 33285>

STRING_TYPE = <TypeKind.STRING_TYPE: 33286>

STRUCTURE_FWD_DECL_TYPE = <TypeKind.STRUCTURE_FWD_DECL_TYPE: 33035>

STRUCTURE_TYPE = <TypeKind.STRUCTURE_TYPE: 33033>

class TypeKind

Bases: pybind11_object

Members:

NO_TYPE : Sentinel indicating no value.

PRIMITIVE_TYPE : Primitive type.

CONSTRUCTED_TYPE : Constructed type.

COLLECTION_TYPE : Collection type.

AGGREGATION_TYPE : Aggregation type.

ANNOTATION_TYPE : Annotation type.

BOOLEAN_TYPE : Boolean type.

UINT_8_TYPE : Unsigned 8-bit int type.

INT_16_TYPE : Signed 16-bit int type.

UINT_16_TYPE : Unsigned 16-bit int type.

INT_32_TYPE : Signed 32-bit int type.

UINT_32_TYPE : Unsigned 32-bit int type.

INT_64_TYPE : Signed 64-bit int type.

UINT_64_TYPE : Unsigned 64-bit int type

FLOAT_32_TYPE : 32-bit floating point type.

FLOAT_64_TYPE : 64-bit floating point type.

FLOAT_128_TYPE : 128-bit floating point type.

CHAR_8_TYPE : 8-bit character type.

CHAR_16_TYPE : 16-bit character type.

ENUMERATION_TYPE : Enumeration type.

BITSET_TYPE : Bitset type.

ALIAS_TYPE : Alias type.

ARRAY_TYPE : Array type.

SEQUENCE_TYPE : Sequence type.

STRING_TYPE : String type.

WSTRING_TYPE : Wide character string type.

MAP_TYPE : Map type

UNION_TYPE : Union type.

STRUCTURE_TYPE : Structure type.

UNION_FWD_DECL_TYPE : Forward declaration of a union type.

STRUCTURE_FWD_DECL_TYPE : Forward declaration of structure type.

AGGREGATION_TYPE = <TypeKind.AGGREGATION_TYPE: 256>

ALIAS_TYPE = <TypeKind.ALIAS_TYPE: 32771>

ANNOTATION_TYPE = <TypeKind.ANNOTATION_TYPE: 128>

ARRAY_TYPE = <TypeKind.ARRAY_TYPE: 33284>

BITSET_TYPE = <TypeKind.BITSET_TYPE: 32770>

BOOLEAN_TYPE = <TypeKind.BOOLEAN_TYPE: 16385>

CHAR_16_TYPE = <TypeKind.CHAR_16_TYPE: 16397>

CHAR_8_TYPE = <TypeKind.CHAR_8_TYPE: 16396>

COLLECTION_TYPE = <TypeKind.COLLECTION_TYPE: 512>

CONSTRUCTED_TYPE = <TypeKind.CONSTRUCTED_TYPE: 32768>

ENUMERATION_TYPE = <TypeKind.ENUMERATION_TYPE: 32769>

FLOAT_128_TYPE = <TypeKind.FLOAT_128_TYPE: 16395>

FLOAT_32_TYPE = <TypeKind.FLOAT_32_TYPE: 16393>

FLOAT_64_TYPE = <TypeKind.FLOAT_64_TYPE: 16394>

INT_16_TYPE = <TypeKind.INT_16_TYPE: 16387>

INT_32_TYPE = <TypeKind.INT_32_TYPE: 16389>

INT_64_TYPE = <TypeKind.INT_64_TYPE: 16391>

MAP_TYPE = <TypeKind.MAP_TYPE: 33287>

NO_TYPE = <TypeKind.NO_TYPE: 0>

PRIMITIVE_TYPE = <TypeKind.PRIMITIVE_TYPE: 16384>

SEQUENCE_TYPE = <TypeKind.SEQUENCE_TYPE: 33285>

STRING_TYPE = <TypeKind.STRING_TYPE: 33286>

STRUCTURE_FWD_DECL_TYPE = <TypeKind.STRUCTURE_FWD_DECL_TYPE: 33035>

STRUCTURE_TYPE = <TypeKind.STRUCTURE_TYPE: 33033>

UINT_16_TYPE = <TypeKind.UINT_16_TYPE: 16388>

UINT_32_TYPE = <TypeKind.UINT_32_TYPE: 16390>

UINT_64_TYPE = <TypeKind.UINT_64_TYPE: 16392>

UINT_8_TYPE = <TypeKind.UINT_8_TYPE: 16386>

UNION_FWD_DECL_TYPE = <TypeKind.UNION_FWD_DECL_TYPE: 33034>

UNION_TYPE = <TypeKind.UNION_TYPE: 33032>

WSTRING_TYPE = <TypeKind.WSTRING_TYPE: 33288>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.TypeKind.TypeKind) → int

__init__(*self:* rti.connextdds.TypeKind.TypeKind, *value: int*) → None

__int__(*self:* rti.connextdds.TypeKind.TypeKind) → int

__members__ = {'AGGREGATION_TYPE': <TypeKind.AGGREGATION_TYPE: 256>, 'ALIAS_TYPE': <TypeKind.ALIAS_TYPE: 32771>, 'ANNOTATION_TYPE': <TypeKind.ANNOTATION_TYPE: 128>, 'ARRAY_TYPE': <TypeKind.ARRAY_TYPE: 33284>, 'BITSET_TYPE': <TypeKind.BITSET_TYPE: 32770>, 'BOOLEAN_TYPE': <TypeKind.BOOLEAN_TYPE: 16385>, 'CHAR_16_TYPE': <TypeKind.CHAR_16_TYPE: 16397>, 'CHAR_8_TYPE': <TypeKind.CHAR_8_TYPE: 16396>, 'COLLECTION_TYPE': <TypeKind.COLLECTION_TYPE: 512>, 'CONSTRUCTED_TYPE': <TypeKind.CONSTRUCTED_TYPE: 32768>, 'ENUMERATION_TYPE': <TypeKind.ENUMERATION_TYPE: 32769>, 'FLOAT_128_TYPE': <TypeKind.FLOAT_128_TYPE: 16395>, 'FLOAT_32_TYPE': <TypeKind.FLOAT_32_TYPE: 16393>, 'FLOAT_64_TYPE': <TypeKind.FLOAT_64_TYPE: 16394>, 'INT_16_TYPE': <TypeKind.INT_16_TYPE: 16387>, 'INT_32_TYPE': <TypeKind.INT_32_TYPE: 16389>, 'INT_64_TYPE': <TypeKind.INT_64_TYPE: 16391>, 'MAP_TYPE': <TypeKind.MAP_TYPE: 33287>, 'NO_TYPE': <TypeKind.NO_TYPE: 0>, 'PRIMITIVE_TYPE': <TypeKind.PRIMITIVE_TYPE: 16384>, 'SEQUENCE_TYPE': <TypeKind.SEQUENCE_TYPE: 33285>, 'STRING_TYPE': <TypeKind.STRING_TYPE: 33286>, 'STRUCTURE_FWD_DECL_TYPE': <TypeKind.STRUCTURE_FWD_DECL_TYPE: 33035>, 'STRUCTURE_TYPE': <TypeKind.STRUCTURE_TYPE: 33033>, 'UINT_16_TYPE': <TypeKind.UINT_16_TYPE: 16388>, 'UINT_32_TYPE': <TypeKind.UINT_32_TYPE: 16390>, 'UINT_64_TYPE': <TypeKind.UINT_64_TYPE: 16392>, 'UINT_8_TYPE': <TypeKind.UINT_8_TYPE: 16386>, 'UNION_FWD_DECL_TYPE': <TypeKind.UNION_FWD_DECL_TYPE: 33034>, 'UNION_TYPE': <TypeKind.UNION_TYPE: 33032>, 'WSTRING_TYPE': <TypeKind.WSTRING_TYPE: 33288>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.TypeKind.TypeKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

UINT_16_TYPE = <TypeKind.UINT_16_TYPE: 16388>

UINT_32_TYPE = <TypeKind.UINT_32_TYPE: 16390>

UINT_64_TYPE = <TypeKind.UINT_64_TYPE: 16392>

UINT_8_TYPE = <TypeKind.UINT_8_TYPE: 16386>

UNION_FWD_DECL_TYPE = <TypeKind.UNION_FWD_DECL_TYPE: 33034>

UNION_TYPE = <TypeKind.UNION_TYPE: 33032>

WSTRING_TYPE = <TypeKind.WSTRING_TYPE: 33288>

__eq__ (*self:* rti.connextdds.TypeKind, *arg0:* rti.connextdds.TypeKind) → bool

Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.TypeKind, *arg0:* rti.connextdds.TypeKind) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.TypeKind, *arg0:* rti.connextdds.TypeKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.TypeKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.TypeKind, arg0: rti.connextdds.TypeKind.TypeKind) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.TypeKind) → *rti.connextdds.TypeKind.TypeKind*

Int conversion.

__le__ (*self:* rti.connextdds.TypeKind, *arg0:* rti.connextdds.TypeKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.TypeKind, *arg0:* rti.connextdds.TypeKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TypeKind, *arg0:* rti.connextdds.TypeKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.TypeKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**TypeSupport**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.TypeSupport, *arg0:* rti.connextdds.TypeSupport) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.TypeSupport) → None

Creates the default policy.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.TypeSupport, *arg0:* rti.connextdds.TypeSupport) → bool

Test for inequality.

property cdr_padding_kind

Determines whether or not the padding bytes will be set to zero during CDR serialization.

rti.connextdds.**ULongLongSeq**

alias of *Uint64Seq*

rti.connextdds.**ULongLongType**

alias of *Uint64Type*

rti.connextdds.**ULongSeq**

alias of *Uint32Seq*

rti.connextdds.**ULongType**

alias of *Uint32Type*

rti.connextdds.**UShortType**

alias of *Uint16Type*

class rti.connextdds.**Uint16Seq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.Uint16Seq, *arg0:* rti.connextdds.Uint16Seq) → *rti.connextdds.Uint16Seq*

__bool__ (*self:* rti.connextdds.Uint16Seq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.Uint16Seq, *x: int*) → bool

Return true the container contains x

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.Uint16Seq, arg0: int) -> None

Delete the list elements at index i

2. __delitem__(self: rti.connextdds.Uint16Seq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.Uint16Seq, *arg0:* rti.connextdds.Uint16Seq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.Uint16Seq, s: slice) -> rti.connextdds.Uint16Seq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.Uint16Seq, arg0: int) -> int

__getstate__ (*self:* rti.connextdds.Uint16Seq) → bytes

__hash__ = None

__iadd__ (*self:* rti.connextdds.Uint16Seq, *arg0:* rti.connextdds.Uint16Seq) → *rti.connextdds.Uint16Seq*

__imul__ (*self:* rti.connextdds.Uint16Seq, *arg0: int*) → *rti.connextdds.Uint16Seq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Uint16Seq, arg0: buffer) -> None
2. __init__(self: rti.connextdds.Uint16Seq) -> None
3. __init__(self: rti.connextdds.Uint16Seq, arg0: rti.connextdds.Uint16Seq) -> None

Copy constructor

4. __init__(self: rti.connextdds.Uint16Seq, arg0: Iterable) -> None
5. __init__(self: rti.connextdds.Uint16Seq, arg0: int) -> None

__iter__ (*self:* rti.connextdds.Uint16Seq) → Iterator

__len__ (*self:* rti.connextdds.Uint16Seq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.Uint16Seq, *arg0: int*) → *rti.connextdds.Uint16Seq*

__ne__ (*self:* rti.connextdds.Uint16Seq, *arg0:* rti.connextdds.Uint16Seq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__ (*self:* rti.connextdds.Uint16Seq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.Uint16Seq, *arg0: int*) → *rti.connextdds.Uint16Seq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.Uint16Seq, arg0: int, arg1: int) -> None
2. __setitem__(self: rti.connextdds.Uint16Seq, arg0: slice, arg1: rti.connextdds.Uint16Seq) -> None

Assign list elements using a slice object

__setstate__ (*self:* rti.connextdds.Uint16Seq, *arg0: bytes*) → None

append (*self:* rti.connextdds.Uint16Seq, *x: int*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.Uint16Seq) → None

Clear the contents

count (*self:* rti.connextdds.Uint16Seq, *x: int*) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.Uint16Seq, L: rti.connextdds.Uint16Seq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.Uint16Seq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.Uint16Seq, *i: int*, *x: int*) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.Uint16Seq) -> int

Remove and return the last item

2. pop(self: rti.connextdds.Uint16Seq, i: int) -> int

Remove and return the item at index `i`

remove (*self:* rti.connextdds.Uint16Seq, *x: int*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

resize (*self:* rti.connextdds.Uint16Seq, *arg0: int*) → None

resizes the vector to the given size

class rti.connextdds.**Uint16Type**

Bases: *DynamicType*

__eq__ (*self:* rti.connextdds.Uint16Type, *arg0:* rti.connextdds.Uint16Type) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.Uint16Type) → None

Get the singleton for Uint16Type

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Uint16Type, *arg0:* rti.connextdds.Uint16Type) → bool

Test for inequality.

class rti.connextdds.**Uint32Seq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.Uint32Seq, *arg0:* rti.connextdds.Uint32Seq) → *rti.connextdds.Uint32Seq*

__bool__ (*self:* rti.connextdds.Uint32Seq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.Uint32Seq, *x: int*) → bool

Return true the container contains x

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.Uint32Seq, arg0: int) -> None

Delete the list elements at index i

2. __delitem__(self: rti.connextdds.Uint32Seq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.Uint32Seq, *arg0:* rti.connextdds.Uint32Seq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.Uint32Seq, s: slice) -> rti.connextdds.Uint32Seq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.Uint32Seq, arg0: int) -> int

__getstate__ (*self:* rti.connextdds.Uint32Seq) → bytes

__hash__ = None

__iadd__ (*self:* rti.connextdds.Uint32Seq, *arg0:* rti.connextdds.Uint32Seq) → *rti.connextdds.Uint32Seq*

__imul__ (*self:* rti.connextdds.Uint32Seq, *arg0: int*) → *rti.connextdds.Uint32Seq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Uint32Seq, arg0: buffer) -> None
2. __init__(self: rti.connextdds.Uint32Seq) -> None
3. __init__(self: rti.connextdds.Uint32Seq, arg0: rti.connextdds.Uint32Seq) -> None

Copy constructor

4. __init__(self: rti.connextdds.Uint32Seq, arg0: Iterable) -> None
5. __init__(self: rti.connextdds.Uint32Seq, arg0: int) -> None

__iter__ (*self:* rti.connextdds.Uint32Seq) → Iterator

__len__ (*self:* rti.connextdds.Uint32Seq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.Uint32Seq, *arg0: int*) → *rti.connextdds.Uint32Seq*

__ne__ (*self:* rti.connextdds.Uint32Seq, *arg0:* rti.connextdds.Uint32Seq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__ (*self:* rti.connextdds.Uint32Seq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.Uint32Seq, *arg0: int*) → *rti.connextdds.Uint32Seq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.Uint32Seq, arg0: int, arg1: int) -> None
2. __setitem__(self: rti.connextdds.Uint32Seq, arg0: slice, arg1: rti.connextdds.Uint32Seq) -> None

Assign list elements using a slice object

__setstate__ (*self:* rti.connextdds.Uint32Seq, *arg0: bytes*) → None

append (*self:* rti.connextdds.Uint32Seq, *x: int*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.Uint32Seq) → None

Clear the contents

count (*self:* rti.connextdds.Uint32Seq, *x: int*) → int

Return the number of times x appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.Uint32Seq, L: rti.connextdds.Uint32Seq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.Uint32Seq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert(*self:* rti.connextdds.Uint32Seq, *i: int*, *x: int*) → None

Insert an item at a given position.

pop(**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.Uint32Seq) -> int

Remove and return the last item

2. pop(self: rti.connextdds.Uint32Seq, i: int) -> int

Remove and return the item at index `i`

remove(*self:* rti.connextdds.Uint32Seq, *x: int*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

resize(*self:* rti.connextdds.Uint32Seq, *arg0: int*) → None

resizes the vector to the given size

class rti.connextdds.**Uint32Type**

Bases: *DynamicType*

__eq__(*self:* rti.connextdds.Uint32Type, *arg0:* rti.connextdds.Uint32Type) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.Uint32Type) → None

Get the singleton for Uint32Type

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.Uint32Type, *arg0:* rti.connextdds.Uint32Type) → bool

Test for inequality.

class rti.connextdds.**Uint64Seq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.Uint64Seq, *arg0:* rti.connextdds.Uint64Seq) → *rti.connextdds.Uint64Seq*

__bool__(*self:* rti.connextdds.Uint64Seq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.Uint64Seq, *x: int*) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.Uint64Seq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.Uint64Seq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.Uint64Seq, *arg0:* rti.connextdds.Uint64Seq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.Uint64Seq, s: slice) -> rti.connextdds.Uint64Seq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.Uint64Seq, arg0: int) -> int

__getstate__ (*self:* rti.connextdds.Uint64Seq) → bytes

__hash__ = None

__iadd__ (*self:* rti.connextdds.Uint64Seq, *arg0:* rti.connextdds.Uint64Seq) → *rti.connextdds.Uint64Seq*

__imul__ (*self:* rti.connextdds.Uint64Seq, *arg0: int*) → *rti.connextdds.Uint64Seq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Uint64Seq, arg0: buffer) -> None

2. __init__(self: rti.connextdds.Uint64Seq) -> None

3. __init__(self: rti.connextdds.Uint64Seq, arg0: rti.connextdds.Uint64Seq) -> None

Copy constructor

4. __init__(self: rti.connextdds.Uint64Seq, arg0: Iterable) -> None

5. __init__(self: rti.connextdds.Uint64Seq, arg0: int) -> None

__iter__ (*self:* rti.connextdds.Uint64Seq) → Iterator

__len__ (*self:* rti.connextdds.Uint64Seq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.Uint64Seq, *arg0: int*) → *rti.connextdds.Uint64Seq*

__ne__ (*self:* rti.connextdds.Uint64Seq, *arg0:* rti.connextdds.Uint64Seq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__ (*self:* rti.connextdds.Uint64Seq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.Uint64Seq, *arg0: int*) → *rti.connextdds.Uint64Seq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.Uint64Seq, arg0: int, arg1: int) -> None
2. __setitem__(self: rti.connextdds.Uint64Seq, arg0: slice, arg1: rti.connextdds.Uint64Seq) -> None

Assign list elements using a slice object

__setstate__ (*self:* rti.connextdds.Uint64Seq, *arg0: bytes*) → None

append (*self:* rti.connextdds.Uint64Seq, *x: int*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.Uint64Seq) → None

Clear the contents

count (*self:* rti.connextdds.Uint64Seq, *x: int*) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.Uint64Seq, L: rti.connextdds.Uint64Seq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.Uint64Seq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.Uint64Seq, *i: int*, *x: int*) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.Uint64Seq) -> int

Remove and return the last item

2. pop(self: rti.connextdds.Uint64Seq, i: int) -> int

Remove and return the item at index `i`

remove (*self:* rti.connextdds.Uint64Seq, *x: int*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

resize (*self:* rti.connextdds.Uint64Seq, *arg0: int*) → None

resizes the vector to the given size

class rti.connextdds.**Uint64Type**

Bases: *DynamicType*

__eq__ (*self:* rti.connextdds.Uint64Type, *arg0:* rti.connextdds.Uint64Type) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.Uint64Type) → None

Get the singleton for Uint64Type

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Uint64Type, *arg0:* rti.connextdds.Uint64Type) → bool

Test for inequality.

class rti.connextdds.**Uint8Seq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.Uint8Seq, *arg0:* rti.connextdds.Uint8Seq) → *rti.connextdds.Uint8Seq*

__bool__ (*self:* rti.connextdds.Uint8Seq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.Uint8Seq, *x: int*) → bool

Return true the container contains x

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.Uint8Seq, arg0: int) -> None

Delete the list elements at index i

2. __delitem__(self: rti.connextdds.Uint8Seq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.Uint8Seq, *arg0:* rti.connextdds.Uint8Seq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.Uint8Seq, s: slice) -> rti.connextdds.Uint8Seq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.Uint8Seq, arg0: int) -> int

__getstate__ (*self:* rti.connextdds.Uint8Seq) → bytes

__hash__ = None

__iadd__ (*self:* rti.connextdds.Uint8Seq, *arg0:* rti.connextdds.Uint8Seq) → *rti.connextdds.Uint8Seq*

__imul__ (*self:* rti.connextdds.Uint8Seq, *arg0: int*) → *rti.connextdds.Uint8Seq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Uint8Seq, arg0: buffer) -> None
2. __init__(self: rti.connextdds.Uint8Seq) -> None
3. __init__(self: rti.connextdds.Uint8Seq, arg0: rti.connextdds.Uint8Seq) -> None

Copy constructor

4. __init__(self: rti.connextdds.Uint8Seq, arg0: Iterable) -> None
5. __init__(self: rti.connextdds.Uint8Seq, arg0: int) -> None

__iter__ (*self:* rti.connextdds.Uint8Seq) → Iterator

__len__ (*self:* rti.connextdds.Uint8Seq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.Uint8Seq, *arg0: int*) → *rti.connextdds.Uint8Seq*

__ne__ (*self:* rti.connextdds.Uint8Seq, *arg0:* rti.connextdds.Uint8Seq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__ (*self:* rti.connextdds.Uint8Seq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.Uint8Seq, *arg0: int*) → *rti.connextdds.Uint8Seq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.Uint8Seq, arg0: int, arg1: int) -> None
2. __setitem__(self: rti.connextdds.Uint8Seq, arg0: slice, arg1: rti.connextdds.Uint8Seq) -> None

Assign list elements using a slice object

__setstate__ (*self:* rti.connextdds.Uint8Seq, *arg0: bytes*) → None

append (*self:* rti.connextdds.Uint8Seq, *x: int*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.Uint8Seq) → None

Clear the contents

count (*self:* rti.connextdds.Uint8Seq, *x: int*) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.Uint8Seq, L: rti.connextdds.Uint8Seq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.Uint8Seq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.Uint8Seq, *i: int*, *x: int*) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.Uint8Seq) -> int

Remove and return the last item

2. pop(self: rti.connextdds.Uint8Seq, i: int) -> int

Remove and return the item at index `i`

remove (*self:* rti.connextdds.Uint8Seq, *x: int*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

resize (*self:* rti.connextdds.Uint8Seq, *arg0: int*) → None

resizes the vector to the given size

class rti.connextdds.**Uint8Type**

Bases: *DynamicType*

__eq__ (*self:* rti.connextdds.Uint8Type, *arg0:* rti.connextdds.Uint8Type) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.Uint8Type) → None

Get the singleton for Uint8Type

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Uint8Type, *arg0:* rti.connextdds.Uint8Type) → bool

Test for inequality.

class rti.connextdds.**UnidimensionalCollectionType**

Bases: *CollectionType*

UNBOUNDED = 2147483647

__eq__(*self:* rti.connextdds.UnidimensionalCollectionType, *arg0:* rti.connextdds.UnidimensionalCollectionType) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.UnidimensionalCollectionType, *arg0:* rti.connextdds.UnidimensionalCollectionType) → bool

Test for inequality.

property bounds

Gets the maximum length of this collection.

class rti.connextdds.**UnionMember**

Bases: pybind11_object

DEFAULT_LABEL = 1073741825

INVALID_ID = 2147483647

__eq__(*self:* rti.connextdds.UnionMember, *arg0:* rti.connextdds.UnionMember) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.UnionMember, name: str, dynamic_type: rti.connextdds.DynamicType, label: int) -> None

Creates a union member with a name, type, and selected by a single label.

2. __init__(self: rti.connextdds.UnionMember, name: str, dynamic_type: rti.connextdds.DynamicType, labels: rti.connextdds.Int32Seq) -> None

Create a union member with a name, type, and selected by one or more labels.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.UnionMember, *arg0:* rti.connextdds.UnionMember) → bool

Test for inequality.

property has_id

Indicates if union member has an ID annotation.

property id

Returns the ID annotation of this member, or INVALID_ID if this member has no ID.

property label_count

Gets the number of labels that select this member.

property labels

The labels that select this member.

property name

The union member's name.

property pointer

Boolean flag for pointer status of union member.

property type

Gets the union member's type.

class rti.connextdds.**UnionMemberSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.UnionMemberSeq, *arg0:* rti.connextdds.UnionMemberSeq) → *rti.connextdds.UnionMemberSeq*

__bool__(*self:* rti.connextdds.UnionMemberSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.UnionMemberSeq, *x:* rti.connextdds.UnionMember) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.UnionMemberSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.UnionMemberSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__(*self:* rti.connextdds.UnionMemberSeq, *arg0:* rti.connextdds.UnionMemberSeq) → bool

__getitem__(**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.UnionMemberSeq, s: slice) -> rti.connextdds.UnionMemberSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.UnionMemberSeq, arg0: int) -> rti.connextdds.UnionMember

__hash__ = None

__iadd__ (*self:* rti.connextdds.UnionMemberSeq, *arg0:* rti.connextdds.UnionMemberSeq) → *rti.connextdds.UnionMemberSeq*

__imul__ (*self:* rti.connextdds.UnionMemberSeq, *arg0: int*) → *rti.connextdds.UnionMemberSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.UnionMemberSeq) -> None
2. __init__(self: rti.connextdds.UnionMemberSeq, arg0: rti.connextdds.UnionMemberSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.UnionMemberSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.UnionMemberSeq) → Iterator

__len__ (*self:* rti.connextdds.UnionMemberSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.UnionMemberSeq, *arg0: int*) → *rti.connextdds.UnionMemberSeq*

__ne__ (*self:* rti.connextdds.UnionMemberSeq, *arg0:* rti.connextdds.UnionMemberSeq) → bool

__rmul__ (*self:* rti.connextdds.UnionMemberSeq, *arg0: int*) → *rti.connextdds.UnionMemberSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.UnionMemberSeq, arg0: int, arg1: rti.connextdds.UnionMember) -> None
2. __setitem__(self: rti.connextdds.UnionMemberSeq, arg0: slice, arg1: rti.connextdds.UnionMemberSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.UnionMemberSeq, *x:* rti.connextdds.UnionMember) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.UnionMemberSeq) → None

Clear the contents

count (*self:* rti.connextdds.UnionMemberSeq, *x:* rti.connextdds.UnionMember) → int

Return the number of times x appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.UnionMemberSeq, L: rti.connextdds.UnionMemberSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.UnionMemberSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.UnionMemberSeq, *i: int*, *x:* rti.connextdds.UnionMember) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.UnionMemberSeq) -> rti.connextdds.UnionMember

Remove and return the last item

2. pop(self: rti.connextdds.UnionMemberSeq, i: int) -> rti.connextdds.UnionMember

Remove and return the item at index `i`

remove (*self:* rti.connextdds.UnionMemberSeq, *x:* rti.connextdds.UnionMember) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**UnionType**

Bases: *ACTUnionMember*

__eq__ (*self:* rti.connextdds.UnionType, *arg0:* rti.connextdds.UnionType) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.UnionType, name: str, discriminator_type: rti.connextdds.DynamicType) -> None

Creates an empty Union.

2. __init__(self: rti.connextdds.UnionType, name: str, discriminator_type: rti.connextdds.DynamicType, members: rti.connextdds.UnionMemberSeq) -> None

Creates a Union with the specified UnionMembers.

3. __init__(self: rti.connextdds.UnionType, type: rti.connextdds.DynamicType) -> None

Cast a DynamicType to a UnionType.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.UnionType, *arg0:* rti.connextdds.UnionType) → bool

Test for inequality.

add_members (*self:* rti.connextdds.UnionType, *members:* rti.connextdds.UnionMemberSeq) → *rti.connextdds.UnionType*

Adds all the members of a container at the end.

property discriminator

The union discriminator type.

property extensibility_kind

Union's extensibility kind.

find_member_by_id(*self:* rti.connextdds.UnionType, *id: int*) → int

Gets the index of the member selected by an ID.

find_member_by_label(*self:* rti.connextdds.UnionType, *label: int*) → int

Gets the index of the member selected by a label.

exception rti.connextdds.**UnsupportedError**

Bases: *Exception*

__module__ = 'rti.connextdds'

class rti.connextdds.**UserData**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.UserData, *arg0:* rti.connextdds.UserData) → bool

Test for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.UserData) -> None

Creates an instance with an empty sequence of bytes.

2. __init__(self: rti.connextdds.UserData, data: rti.connextdds.Uint8Seq) -> None

Creates an instance with a sequence of bytes.

__iter__(*self:* rti.connextdds.UserData) → Iterator

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.UserData, *arg0:* rti.connextdds.UserData) → bool

Test for inequality.

property value

The user data.

class rti.connextdds.**UserDataSample**

Bases: pybind11_object

An alias for a Python object representing a data sample of an IDL type

__init__(**args*, ***kwargs*)

__module__ = 'rti.connextdds'

rti.connextdds.**UshortSeq**

alias of *Uint16Seq*

class rti.connextdds.**VendorId**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.VendorId, *arg0:* rti.connextdds.VendorId) → bool

Test for equality.

__hash__ = None

__init__(*self:* rti.connextdds.VendorId) → None

Creates unknown vendor ID.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.VendorId, *arg0:* rti.connextdds.VendorId) → bool

Test for inequality.

unknown = <rti.connextdds.VendorId object>

property value

Provides access to the bytes that represent the vendor id.

class rti.connextdds.**Verbosity**

Bases: pybind11_object

EXCEPTION = <Verbosity.EXCEPTION: 3>

SILENT = <Verbosity.SILENT: 0>

STATUS_ALL = <Verbosity.STATUS_ALL: 63>

STATUS_LOCAL = <Verbosity.STATUS_LOCAL: 15>

STATUS_REMOTE = <Verbosity.STATUS_REMOTE: 31>

class Verbosity

Bases: pybind11_object

Members:

SILENT : No further output will be logged.

EXCEPTION : Only error messages will be logged.

An error indicates something wrong in the functioning of RTI Connext. The most common cause of errors is incorrect configuration.

WARNING : Both error and warning messages will be logged.

A warning indicates that RTI Connext is taking an action that may or may not be what you intended. Some configuration information is also logged at this verbosity to aid in debugging.

STATUS_LOCAL : Errors, warnings, and verbose information about the lifecycles of local RTI Connext objects will be logged.

STATUS_REMOTE : Errors, warnings, and verbose information about the lifecycles of remote RTI Connext objects will be logged.

STATUS_ALL : Errors, warnings, verbose information about the lifecycles of local and remote RTI Connext objects, and periodic information about RTI Connext threads will be logged.

EXCEPTION = <Verbosity.EXCEPTION: 3>

SILENT = <Verbosity.SILENT: 0>

STATUS_ALL = <Verbosity.STATUS_ALL: 63>

STATUS_LOCAL = <Verbosity.STATUS_LOCAL: 15>

STATUS_REMOTE = <Verbosity.STATUS_REMOTE: 31>

WARNING = <Verbosity.WARNING: 7>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.Verbosity.Verbosity) → int

__init__(*self:* rti.connextdds.Verbosity.Verbosity, *value: int*) → None

__int__(*self:* rti.connextdds.Verbosity.Verbosity) → int

__members__ = {'EXCEPTION': <Verbosity.EXCEPTION: 3>, 'SILENT': <Verbosity.SILENT: 0>, 'STATUS_ALL': <Verbosity.STATUS_ALL: 63>, 'STATUS_LOCAL': <Verbosity.STATUS_LOCAL: 15>, 'STATUS_REMOTE': <Verbosity.STATUS_REMOTE: 31>, 'WARNING': <Verbosity.WARNING: 7>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.Verbosity.Verbosity, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

WARNING = <Verbosity.WARNING: 7>

__eq__ (*self:* rti.connextdds.Verbosity, *arg0:* rti.connextdds.Verbosity) → bool

Apply operator to underlying enumerated values.

__ge__ (*self:* rti.connextdds.Verbosity, *arg0:* rti.connextdds.Verbosity) → bool

Apply operator to underlying enumerated values.

__gt__ (*self:* rti.connextdds.Verbosity, *arg0:* rti.connextdds.Verbosity) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.Verbosity) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.Verbosity, arg0: rti.connextdds.Verbosity.Verbosity) -> None

Copy constructor.

__int__ (*self:* rti.connextdds.Verbosity) → *rti.connextdds.Verbosity.Verbosity*

Int conversion.

__le__ (*self:* rti.connextdds.Verbosity, *arg0:* rti.connextdds.Verbosity) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.Verbosity, *arg0:* rti.connextdds.Verbosity) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.Verbosity, *arg0:* rti.connextdds.Verbosity) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.Verbosity) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**ViewState**

Bases: pybind11_object

ANY = <rti.connextdds.ViewState object>

NEW_VIEW = <rti.connextdds.ViewState object>

NOT_NEW_VIEW = <rti.connextdds.ViewState object>

__and__ (*self:* rti.connextdds.ViewState, *arg0:* rti.connextdds.ViewState) → *rti.connextdds.ViewState*

Bitwise logical AND of masks.

__bool__ (*self:* rti.connextdds.ViewState) → *rti.connextdds.ViewState*

Test if any bits are set.

__contains__ (*self:* rti.connextdds.ViewState, *arg0:* rti.connextdds.ViewState) → bool

__eq__ (*self:* rti.connextdds.ViewState, *arg0:* rti.connextdds.ViewState) → bool

Compare masks for equality.

__getitem__ (*self:* rti.connextdds.ViewState, *arg0: int*) → bool

Get individual mask bit.

__hash__ = None

__iand__ (*self:* rti.connextdds.ViewState, *arg0:* rti.connextdds.ViewState) → *rti.connextdds.ViewState*

Set mask to logical AND with another mask.

__ilshift__ (*self:* rti.connextdds.ViewState, *arg0: int*) → *rti.connextdds.ViewState*

Left shift bits in mask.

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.ViewState) -> None

Create a ViewState with no bits set.

2. __init__(self: rti.connextdds.ViewState, value: int) -> None

Creates a mask from the bits in an integer.

__int__ (*self:* rti.connextdds.ViewState) → int

Convert mask to int.

__ior__ (*self:* rti.connextdds.ViewState, *arg0:* rti.connextdds.ViewState) → *rti.connextdds.ViewState*

Set mask to logical OR with another mask.

__irshift__ (*self:* rti.connextdds.ViewState, *arg0: int*) → *rti.connextdds.ViewState*

Right shift bits in mask.

__ixor__ (*self:* rti.connextdds.ViewState, *arg0:* rti.connextdds.ViewState) → *rti.connextdds.ViewState*

Set mask to logical XOR with another mask.

__lshift__ (*self:* rti.connextdds.ViewState, *arg0: int*) → *rti.connextdds.ViewState*

Left shift bits in mask.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.ViewState, *arg0:* rti.connextdds.ViewState) → bool

Compare masks for inequality.

__or__ (*self:* rti.connextdds.ViewState, *arg0:* rti.connextdds.ViewState) → *rti.connextdds.ViewState*

Bitwise logical OR of masks.

__rshift__ (*self:* rti.connextdds.ViewState, *arg0: int*) → *rti.connextdds.ViewState*

Right shift bits in mask.

__setitem__ (*self:* rti.connextdds.ViewState, *arg0: int*, *arg1: bool*) → None

Set individual mask bit

__str__ (*self:* rti.connextdds.ViewState) → str

__xor__ (*self:* rti.connextdds.ViewState, *arg0:* rti.connextdds.ViewState) → *rti.connextdds.ViewState*

Bitwise logical XOR of masks.

property count

Returns the number of bits set in the mask.

flip (**args*, ***kwargs*)

Overloaded function.

1. flip(self: rti.connextdds.ViewState) -> rti.connextdds.ViewState

Flip all bits in the mask.

2. flip(self: rti.connextdds.ViewState, pos: int) -> rti.connextdds.ViewState

Flip the mask bit at the specified position.

reset (**args*, ***kwargs*)

Overloaded function.

1. reset(self: rti.connextdds.ViewState) -> rti.connextdds.ViewState

Clear all bits in the mask.

2. reset(self: rti.connextdds.ViewState, pos: int) -> rti.connextdds.ViewState

Clear the mask bit at the specified position.

set (**args*, ***kwargs*)

Overloaded function.

1. set(self: rti.connextdds.ViewState) -> rti.connextdds.ViewState

Set all bits in the mask.

2. set(self: rti.connextdds.ViewState, pos: int, value: bool = True) -> rti.connextdds.ViewState

Set the mask bit at the specified position to the provided value (default: true).

property size

Returns the number of bits in the mask type.

test (*self:* rti.connextdds.ViewState, *pos: int*) → bool

Test whether the mask bit at position "pos" is set.

test_all (*self:* rti.connextdds.ViewState) → bool

Test if all bits are set.

test_any (*self:* rti.connextdds.ViewState) → bool

Test if any bits are set.

test_none (*self:* rti.connextdds.ViewState) → bool

Test if none of the bits are set.

class rti.connextdds.**WStringType**

Bases: *UnidimensionalCollectionType*

__eq__ (*self:* rti.connextdds.WStringType, *arg0:* rti.connextdds.WStringType) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.WStringType, bounds: int) -> None

Creates a bounded wide string.

2. __init__(self: rti.connextdds.WStringType) -> None

Creates an unbounded wide string.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.WStringType, *arg0:* rti.connextdds.WStringType) → bool

Test for inequality.

class rti.connextdds.**WaitSet**

Bases: pybind11_object

__iadd__ (*self:* rti.connextdds.WaitSet, *arg0:* rti.connextdds.ICondition) → *rti.connextdds.WaitSet*

Attach a condition to a WaitSet.

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.WaitSet) -> None

Create a WaitSet with no conditions attached.

2. __init__(self: rti.connextdds.WaitSet, property: rti.connextdds.WaitSetProperty) -> None

Create a WaitSet with no conditions attached with the specified WaitSetProperty settings.

__isub__ (*self:* rti.connextdds.WaitSet, *arg0:* rti.connextdds.ICondition) → *rti.connextdds.WaitSet*

Detach a condition from a WaitSet.

__module__ = 'rti.connextdds'

attach_condition (*self:* rti.connextdds.WaitSet, *condition:* rti.connextdds.ICondition) → None

Attach a condition to the WaitSet.

property conditions

Get/set the attached conditions for the WaitSet.

detach_all (*self:* rti.connextdds.WaitSet) → None

Detach all conditions from the WaitSet.

detach_condition (*self:* rti.connextdds.WaitSet, *condition:* rti.connextdds.ICondition) → bool

Detach a condition from the WaitSet.

dispatch (**args*, ***kwargs*)

Overloaded function.

1. dispatch(self: rti.connextdds.WaitSet, timeout: rti.connextdds.Duration) -> None

Dispatch handlers for triggered conditions attached to this WaitSet with a timeout.

2. dispatch(self: rti.connextdds.WaitSet) -> None

Dispatch handlers for triggered conditions attached to this WaitSet with no timeout.

dispatch_async (**args*, ***kwargs*)

Overloaded function.

1. dispatch_async(self: rti.connextdds.WaitSet, timeout: rti.connextdds.Duration) -> object

Dispatch handlers for triggered conditions attached to this WaitSet with a timeout. This call is awaitable and only for use with asyncio.

2. dispatch_async(self: rti.connextdds.WaitSet) -> object

Dispatch handlers for triggered conditions attached to this WaitSet with no timeout. This call is awaitable and only for use with asyncio.

property property

Get/set the WaitSetProperty settings for the WaitSet.

wait (**args*, ***kwargs*)

Overloaded function.

1. wait(self: rti.connextdds.WaitSet, timeout: rti.connextdds.Duration) -> rti.connextdds.TriggeredConditions

Wait for conditions attached to this WaitSet to trigger with a timeout.

2. wait(self: rti.connextdds.WaitSet) -> rti.connextdds.TriggeredConditions

Wait indefinitely for conditions attached to this WaitSet to trigger.

wait_async (**args*, ***kwargs*)

Overloaded function.

1. wait_async(self: rti.connextdds.WaitSet, timeout: rti.connextdds.Duration) -> object

Wait for conditions attached to this WaitSet to trigger with a timeout. This call is awaitable and only for use with asyncio.

2. wait_async(self: rti.connextdds.WaitSet) -> object

Wait indefinitely for conditions attached to this WaitSet to trigger. This call is awaitable and only for use with asyncio.

class rti.connextdds.**WaitSetProperty**

Bases: pybind11_object

__eq__(*self:* rti.connextdds.WaitSetProperty, *arg0:* rti.connextdds.WaitSetProperty) → bool

Compare objects for equality.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.WaitSetProperty) -> None

Constructs a WaitSetProperty with default values.

2. __init__(self: rti.connextdds.WaitSetProperty, event_count: int, event_delay: rti.connextdds.Duration) -> None

Construct a WaitSetProperty with the given settings.

__module__ = 'rti.connextdds'

__ne__(*self:* rti.connextdds.WaitSetProperty, *arg0:* rti.connextdds.WaitSetProperty) → bool

Determine if objects are unequal.

property event_count

Get/set the event count that will wake the WaitSet.

property event_delay

Get/set the delay that will cause the WaitSet to wake even if the event count is not hit.

class rti.connextdds.**WcharSeq**

Bases: pybind11_object

__add__(*self:* rti.connextdds.WcharSeq, *arg0:* rti.connextdds.WcharSeq) → *rti.connextdds.WcharSeq*

__bool__(*self:* rti.connextdds.WcharSeq) → bool

Check whether the list is nonempty

__contains__(*self:* rti.connextdds.WcharSeq, *x: str*) → bool

Return true the container contains `x`

__delitem__(**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.WcharSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.WcharSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.WcharSeq, *arg0:* rti.connextdds.WcharSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.WcharSeq, s: slice) -> rti.connextdds.WcharSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.WcharSeq, arg0: int) -> str

__hash__ = None

__iadd__ (*self:* rti.connextdds.WcharSeq, *arg0:* rti.connextdds.WcharSeq) → *rti.connextdds.WcharSeq*

__imul__ (*self:* rti.connextdds.WcharSeq, *arg0: int*) → *rti.connextdds.WcharSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.WcharSeq) -> None

2. __init__(self: rti.connextdds.WcharSeq, arg0: rti.connextdds.WcharSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.WcharSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.WcharSeq) → Iterator

__len__ (*self:* rti.connextdds.WcharSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.WcharSeq, *arg0: int*) → *rti.connextdds.WcharSeq*

__ne__ (*self:* rti.connextdds.WcharSeq, *arg0:* rti.connextdds.WcharSeq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__repr__ (*self:* rti.connextdds.WcharSeq) → str

Return the canonical string representation of this list.

__rmul__ (*self:* rti.connextdds.WcharSeq, *arg0: int*) → *rti.connextdds.WcharSeq*

__setitem__(**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.WcharSeq, arg0: int, arg1: str) -> None
2. __setitem__(self: rti.connextdds.WcharSeq, arg0: slice, arg1: rti.connextdds.WcharSeq) -> None

Assign list elements using a slice object

append(*self:* rti.connextdds.WcharSeq, *x: str*) → None

Add an item to the end of the list

clear(*self:* rti.connextdds.WcharSeq) → None

Clear the contents

count(*self:* rti.connextdds.WcharSeq, *x: str*) → int

Return the number of times `x` appears in the list

extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.WcharSeq, L: rti.connextdds.WcharSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.WcharSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert(*self:* rti.connextdds.WcharSeq, *i: int*, *x: str*) → None

Insert an item at a given position.

pop(**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.WcharSeq) -> str

Remove and return the last item

2. pop(self: rti.connextdds.WcharSeq, i: int) -> str

Remove and return the item at index `i`

remove(*self:* rti.connextdds.WcharSeq, *x: str*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

class rti.connextdds.**WcharType**

Bases: *DynamicType*

__eq__(*self:* rti.connextdds.WcharType, *arg0:* rti.connextdds.WcharType) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.WcharType) → None

Get the singleton for WcharType

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.WcharType, *arg0:* rti.connextdds.WcharType) → bool

Test for inequality.

class rti.connextdds.**WireProtocol**

Bases: pybind11_object

RTPS_AUTO_ID = 0

__eq__ (*self:* rti.connextdds.WireProtocol, *arg0:* rti.connextdds.WireProtocol) → bool

Test for equality.

__hash__ = None

__init__ (*self:* rti.connextdds.WireProtocol) → None

Creates the default policy.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.WireProtocol, *arg0:* rti.connextdds.WireProtocol) → bool

Test for inequality.

property check_crc

Checks if the received RTPS message is valid by comparing the computed CRC with the received RTPS CRC submessage when this field is set to true.

property compute_crc

Adds RTPS CRC submessage to every message when this field is set to true.

property participant_id

A value used to distinguish among different participants belonging to the same domain on the same host.

property rtps_app_id

The RTPS App ID of the domain participant.

property rtps_auto_id_kind

Kind of auto mechanism used to calculate the GUID prefix.

property rtps_host_id

The RTPS Host ID of the domain participant.

property rtps_instance_id

The RTPS Instance ID of the domain participant.

property rtps_reserved_port_mask

Specifies which well-known ports to reserve when enabling the participant.

This property's getter returns a deep copy.

property rtps_well_known_ports

The RTPS well-known port mappings.

This property's getter returns a deep copy.

class rti.connextdds.**WireProtocolAutoKind**

Bases: pybind11_object

RTPS_AUTO_ID_FROM_IP = <WireProtocolAutoKind.RTPS_AUTO_ID_FROM_IP: 0>

RTPS_AUTO_ID_FROM_MAC = <WireProtocolAutoKind.RTPS_AUTO_ID_FROM_MAC: 1>

RTPS_AUTO_ID_FROM_UUID = <WireProtocolAutoKind.RTPS_AUTO_ID_FROM_UUID: 2>

class WireProtocolAutoKind

Bases: pybind11_object

Members:

RTPS_AUTO_ID_FROM_IP : Select the IPv4 based algorithm.

RTPS_AUTO_ID_FROM_MAC : Select the MAC based algorithm.

RTPS_AUTO_ID_FROM_UUID : Select the UUID based algorithm.

RTPS_AUTO_ID_FROM_IP = <WireProtocolAutoKind.RTPS_AUTO_ID_FROM_IP: 0>

RTPS_AUTO_ID_FROM_MAC = <WireProtocolAutoKind.RTPS_AUTO_ID_FROM_MAC: 1>

RTPS_AUTO_ID_FROM_UUID = <WireProtocolAutoKind.RTPS_AUTO_ID_FROM_UUID: 2>

__eq__(*self: object*, *other: object*) → bool

__getstate__(*self: object*) → int

__hash__(*self: object*) → int

__index__(*self:* rti.connextdds.WireProtocolAutoKind.WireProtocolAutoKind) → int

__init__(*self:* rti.connextdds.WireProtocolAutoKind.WireProtocolAutoKind, *value: int*) → None

__int__(*self:* rti.connextdds.WireProtocolAutoKind.WireProtocolAutoKind) → int

__members__ = {'RTPS_AUTO_ID_FROM_IP':
<WireProtocolAutoKind.RTPS_AUTO_ID_FROM_IP: 0>,
'RTPS_AUTO_ID_FROM_MAC':
<WireProtocolAutoKind.RTPS_AUTO_ID_FROM_MAC: 1>,
'RTPS_AUTO_ID_FROM_UUID':
<WireProtocolAutoKind.RTPS_AUTO_ID_FROM_UUID: 2>}

__module__ = 'rti.connextdds'

__ne__(*self: object*, *other: object*) → bool

__repr__(*self: object*) → str

__setstate__(*self:* rti.connextdds.WireProtocolAutoKind.WireProtocolAutoKind, *state: int*) → None

__str__()

name(self: handle) -> str

property name

property value

__eq__(*self:* rti.connextdds.WireProtocolAutoKind, *arg0:* rti.connextdds.WireProtocolAutoKind) → bool

Apply operator to underlying enumerated values.

__ge__(*self:* rti.connextdds.WireProtocolAutoKind, *arg0:* rti.connextdds.WireProtocolAutoKind) → bool

Apply operator to underlying enumerated values.

__gt__(*self:* rti.connextdds.WireProtocolAutoKind, *arg0:* rti.connextdds.WireProtocolAutoKind) → bool

Apply operator to underlying enumerated values.

__hash__ = None

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.WireProtocolAutoKind) -> None

Initializes enum to 0.

2. __init__(self: rti.connextdds.WireProtocolAutoKind, arg0: rti.connextdds.WireProtocolAutoKind.WireProtocolAutoKind) -> None

Copy constructor.

__int__(*self:* rti.connextdds.WireProtocolAutoKind) → *rti.connextdds.WireProtocolAutoKind.WireProtocolAutoKind*

Int conversion.

__le__ (*self:* rti.connextdds.WireProtocolAutoKind, *arg0:* rti.connextdds.WireProtocolAutoKind) → bool

Apply operator to underlying enumerated values.

__lt__ (*self:* rti.connextdds.WireProtocolAutoKind, *arg0:* rti.connextdds.WireProtocolAutoKind) → bool

Apply operator to underlying enumerated values.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.WireProtocolAutoKind, *arg0:* rti.connextdds.WireProtocolAutoKind) → bool

Apply operator to underlying enumerated values.

__str__ (*self:* rti.connextdds.WireProtocolAutoKind) → str

String conversion.

property underlying

Retrieves the actual enumerated value.

class rti.connextdds.**WriteParams**

Bases: pybind11_object

__init__ (*self:* rti.connextdds.WriteParams) → None

Create a WriteParams object with default values.

__module__ = 'rti.connextdds'

property cookie

The cookie for writing.

property flag

The sample flag for writing.

property handle

The instance handle for writing.

property identity

The sample identity.

property priority

The priority for writing.

property related_reader_guid

The related reader GUID for writing.

property related_sample_identity

The related sample identity.

property related_source_guid

The related source GUID for writing.

property replace_automatic_values

Indicates if the replacement of automatic values has been activated or not.

reset (*self:* rti.connextdds.WriteParams) → None

Reset all fields to their default values.

property source_guid

The source GUID for writing.

property source_timestamp

The source timestamp for writing.

class rti.connextdds.**WriterDataLifecycle**

Bases: pybind11_object

__eq__ (*self:* rti.connextdds.WriterDataLifecycle, *arg0:* rti.connextdds.WriterDataLifecycle) → bool

Test for equality.

__hash__ = None

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.WriterDataLifecycle) -> None

Creates the default policy.

2. __init__(self: rti.connextdds.WriterDataLifecycle, autodispose: bool) -> None

Creates an instance with a value for auto-dispose unregistered instances and default values for the rest of parameters.

__module__ = 'rti.connextdds'

__ne__ (*self:* rti.connextdds.WriterDataLifecycle, *arg0:* rti.connextdds.WriterDataLifecycle) → bool

Test for inequality.

property autodispose_unregistered_instances

Indicates whether the DataWriter should automatically dispose an instance when it unregisters it.

property autopurge_disposed_instances_delay

Maximum duration for which the DataWriter will maintain information regarding an instance once it has disposed the instance.

This property's getter returns a deep copy.

property autopurge_unregistered_instances_delay

Maximum duration for which the DataWriter will maintain information regarding an instance once it has unregistered the instance.

This property's getter returns a deep copy.

class rti.connextdds.**WstringSeq**

Bases: pybind11_object

__add__ (*self:* rti.connextdds.WstringSeq, *arg0:* rti.connextdds.WstringSeq) → *rti.connextdds.WstringSeq*

__bool__ (*self:* rti.connextdds.WstringSeq) → bool

Check whether the list is nonempty

__contains__ (*self:* rti.connextdds.WstringSeq, *x: str*) → bool

Return true the container contains `x`

__delitem__ (**args*, ***kwargs*)

Overloaded function.

1. __delitem__(self: rti.connextdds.WstringSeq, arg0: int) -> None

Delete the list elements at index `i`

2. __delitem__(self: rti.connextdds.WstringSeq, arg0: slice) -> None

Delete list elements using a slice object

__eq__ (*self:* rti.connextdds.WstringSeq, *arg0:* rti.connextdds.WstringSeq) → bool

__getitem__ (**args*, ***kwargs*)

Overloaded function.

1. __getitem__(self: rti.connextdds.WstringSeq, s: slice) -> rti.connextdds.WstringSeq

Retrieve list elements using a slice object

2. __getitem__(self: rti.connextdds.WstringSeq, arg0: int) -> str

__hash__ = None

__iadd__ (*self:* rti.connextdds.WstringSeq, *arg0:* rti.connextdds.WstringSeq) → *rti.connextdds.WstringSeq*

__imul__ (*self:* rti.connextdds.WstringSeq, *arg0: int*) → *rti.connextdds.WstringSeq*

__init__ (**args*, ***kwargs*)

Overloaded function.

1. __init__(self: rti.connextdds.WstringSeq) -> None

2. __init__(self: rti.connextdds.WstringSeq, arg0: rti.connextdds.WstringSeq) -> None

Copy constructor

3. __init__(self: rti.connextdds.WstringSeq, arg0: Iterable) -> None

__iter__ (*self:* rti.connextdds.WstringSeq) → Iterator

__len__ (*self:* rti.connextdds.WstringSeq) → int

__module__ = 'rti.connextdds'

__mul__ (*self:* rti.connextdds.WstringSeq, *arg0: int*) → *rti.connextdds.WstringSeq*

__ne__ (*self:* rti.connextdds.WstringSeq, *arg0:* rti.connextdds.WstringSeq) → bool

__pybind11_module_local_v4_gcc_libstdcpp_cxxabi1002__ = <capsule object NULL>

__rmul__ (*self:* rti.connextdds.WstringSeq, *arg0: int*) → *rti.connextdds.WstringSeq*

__setitem__ (**args*, ***kwargs*)

Overloaded function.

1. __setitem__(self: rti.connextdds.WstringSeq, arg0: int, arg1: str) -> None
2. __setitem__(self: rti.connextdds.WstringSeq, arg0: slice, arg1: rti.connextdds.WstringSeq) -> None

Assign list elements using a slice object

append (*self:* rti.connextdds.WstringSeq, *x: str*) → None

Add an item to the end of the list

clear (*self:* rti.connextdds.WstringSeq) → None

Clear the contents

count (*self:* rti.connextdds.WstringSeq, *x: str*) → int

Return the number of times `x` appears in the list

extend (**args*, ***kwargs*)

Overloaded function.

1. extend(self: rti.connextdds.WstringSeq, L: rti.connextdds.WstringSeq) -> None

Extend the list by appending all the items in the given list

2. extend(self: rti.connextdds.WstringSeq, L: Iterable) -> None

Extend the list by appending all the items in the given list

insert (*self:* rti.connextdds.WstringSeq, *i: int*, *x: str*) → None

Insert an item at a given position.

pop (**args*, ***kwargs*)

Overloaded function.

1. pop(self: rti.connextdds.WstringSeq) -> str

Remove and return the last item

2. pop(self: rti.connextdds.WstringSeq, i: int) -> str

Remove and return the item at index `i`

remove (*self:* rti.connextdds.WstringSeq, *x: str*) → None

Remove the first item from the list whose value is x. It is an error if there is no such item.

rti.connextdds.**set_activity_context**(*attribute_mask:* rti.connextdds.ActivityContextMask) → None

Set the activity context mask.

rti.connextdds.**sleep**(*duration:* rti.connextdds.Duration) → None

Sleep for a specified duration

rti.connextdds.**spin**(*spin_count: int*) → None

Performs a spin operation (active wait) as many times as indicated.

rti.connextdds.**spin_per_microsecond**() → int

Returns the number of spin operations neededto wait 1 microsecond

rti.connextdds.**unregister_thread**() → None

Unregister a thread with the middleware

2.3.2 rti.types

This module defines the `struct`, `union`, `enum`, `alias` decorators that allow turning Python `data-classes` into DDS topic-types.

See *Data Types* for more information.

`rti.types` also contains `rti.types.builtin`.

rti.types.builtin

This submodule defines the DDS built-in types. These simple, general-purpose types are available for the user's convenience to quickly write a publisher or subscriber application.

class rti.types.builtin.**Bytes**(*value: ~typing.Sequence[uint8] = <factory>*)

Built-in type consisting of a sequence of bytes

This type can be used to create a dds.Topic, e.g. `topic = dds.Topic(participant, "My Topic", Bytes)`.

value: Sequence[uint8]

The byte payload

class rti.types.builtin.**KeyedBytes**(*key: str = ''*, *value: ~typing.Sequence[uint8] = <factory>*)

Built-in type consisting of a sequence of bytes and a string that is the key.

This type can be used to create a dds.Topic, e.g. `topic = dds.Topic(participant, "My Topic", KeyedBytes)`.

key: str = ''

The key

value: Sequence[uint8]

The byte payload

class rti.types.builtin.**KeyedString**(*key: str = ''*, *value: str = ''*)

Built-in type consisting of a string payload and a second string that is the key.

This type can be used to create a dds.Topic, e.g. `topic = dds.Topic(participant, "My Topic", KeyedString)`.

key: str = ''

The key

value: str = ''

The string payload

class rti.types.builtin.**String**(*value: str = ''*)

Built-in type consisting of a string

This type can be used to create a dds.Topic, e.g. ` topic = dds.Topic(participant, "My Topic", String) `

value: str = ''

The string payload

2.3.3 rti.rpc

`rti.rpc` is the package containing the RTI Connext Request-Reply and Remote Procedure APIs.

See *Request-Reply and Remote Procedure Calls* for an overview of the API.

class rti.rpc.**Requester**(*request_type: Union[type,* DynamicType*]*, *reply_type: Union[type,* DynamicType*]*, *participant:* DomainParticipant, *service_name: Optional[str] = None*, *request_topic: Optional[Union[*Topic, Topic*, str, object]] = None*, *reply_topic: Optional[Union[*Topic, Topic*, str, object]] = None*, *datawriter_qos: Optional[*DataWriterQos*] = None*, *datareader_qos: Optional[*DataReaderQos*] = None*, *publisher: Optional[*Publisher*] = None*, *subscriber: Optional[*Subscriber*] = None*, *on_reply_available: Optional[Callable[[object], object]] = None*)

A Requester allows sending requests and receiving replies

Parameters

- **request_type** – The type of the request data. It can be an `@idl.struct`, an `@idl.union`, or a dds.DynamicType. (See *Data Types*.)
- **reply_type** – The type of the reply data.
- **participant** – The DomainParticipant that will hold the request writer and reply reader.
- **service_name** – Name that will be used to derive the topic name, defaults to None (rely only on custom topics).

- **request_topic** – Topic object or name that will be used for the request data, must be set if service_name is None, otherwise overrides service_name, defaults to None (use service_name).
- **reply_topic** – Topic object or name that will be used for the reply data, must be set if service_name is None, otherwise overrides service_name, defaults to None (use service_name).
- **datawriter_qos** – QoS object to use for request writer, defaults to None (use default RequestReply QoS).
- **datareader_qos** – QoS object to use for reply reader, defaults to None (use default RequestReply QoS).
- **publisher** – Publisher used to hold request writer, defaults to None (use participant builtin publisher).
- **subscriber** – Subscriber used to hold reply reader, defaults to None (use participant builtin subscriber).
- **on_reply_available** – The callback that handles incoming replies.

__init__(*request_type: Union[type,* DynamicType*]*, *reply_type: Union[type,* DynamicType*]*, *participant:* DomainParticipant, *service_name: Optional[str] = None*, *request_topic: Optional[Union[*Topic*,* Topic*, str, object]] = None*, *reply_topic: Optional[Union[*Topic*,* Topic*, str, object]] = None*, *datawriter_qos: Optional[*DataWriterQos*] = None*, *datareader_qos: Optional[*DataReaderQos*] = None*, *publisher: Optional[*Publisher*] = None*, *subscriber: Optional[*Subscriber*] = None*, *on_reply_available: Optional[Callable[[object], object]] = None*) → None

__weakref__

list of weak references to the object (if defined)

close() → None

Close the resources for this request-reply object.

property closed: bool

Returns true if this request-reply object has been closed.

Getter

Returns the number of matched requesters.

classmethod is_final_reply(*reply_info: Union[*SampleInfo*, object]*) → bool

Check a reply is the last of the sequence.

Parameters

reply_info – The reply info with the flags to check.

Returns

Boolean indicating whether reply is the last for a request.

classmethod is_related_reply(*request_id:* SampleIdentity, *reply_info:* SampleInfo) → bool

Check a request if against a reply's metadata for correlation.

Parameters

- **request_id** – The request id used to correlate replies.
- **reply_info** – The reply info used for the correlation check.

Returns
Boolean indicating whether the request and reply are correlated.

property matched_replier_count: int

The number of discovered matched repliers.

Getter
Returns the number of matched repliers.

property on_reply_available: Optional[Callable[[object], object]]

The listener callback used to process received replies.

Getter
Returns the callback function.

Setter
Set the callback function.

read_replies(*related_request_id: Optional[*SampleIdentity*] = None*) → Union[*LoanedSamples*, *LoanedSamples*]

Read received replies.

Parameters
related_request_id – The id used to correlate replies to a specific request, default None (read any replies).

Returns
A loaned samples object containing the replies.

receive_replies(*max_wait:* Duration, *min_count: int = 1*, *related_request_id: Optional[*SampleIdentity*] = None*) → Union[*LoanedSamples*, *LoanedSamples*, object]

Wait for replies and take them.

Parameters

- **max_wait** – Maximum time to wait for replies before timing out.
- **min_count** – Minimum number of replies to receive, default 1.
- **related_request_id** – The request id used to correlate replies, default None (receive any replies).

Raises
dds.TimeoutError – Thrown if min_count not received within max_wait.

Returns
A loaned samples object containing the replies.

property reply_datareader: Union[*DataReader*, *DataReader*, object]

The DataReader used to receive reply data.

Getter
Returns the reply DataReader.

property request_datawriter: Union[*DataWriter*, *DataWriter*]

The DataWriter used to send request data.

Getter
Returns the request DataWriter.

send_request(*request: Union[object,* DynamicData*]*, *params: Optional[*WriteParams*] = None*) → *SampleIdentity*

Send a request and return the identity of the request for correlating received replies.

Parameters

- **request** – The request to send.
- **params** – Parameters used for writing the request.

Returns
The identity of the request.

take_replies(*related_request_id: Optional[*SampleIdentity*] = None*) → Union[*LoanedSamples*, *LoanedSamples*]

Take received replies.

Parameters
related_request_id – The id used to correlate replies to a specific request, default None (take any replies).

Returns
A loaned samples object containing the replies.

wait_for_replies(*max_wait:* Duration, *min_count: int = 1*, *related_request_id: Optional[*SampleIdentity*] = None*) → bool

Wait for received replies.

Parameters

- **max_wait** – Maximum time to wait for replies before timing out.
- **min_count** – Minimum number of replies to receive, default 1.
- **related_request_id** – The request id used to correlate replies, default None (receive any replies).

Returns
Boolean indicating whether min_count replies were received within max_wait time.

async wait_for_replies_async(*max_wait:* Duration, *min_count: int = 1*, *related_request_id: Optional[*SampleIdentity*] = None*) → bool

Wait for received replies asynchronously.

Parameters

- **max_wait** – Maximum time to wait for replies before timing out.
- **min_count** – Minimum number of replies to receive, default 1.
- **related_request_id** – The request id used to correlate replies, default None (receive any replies).

Returns

Boolean indicating whether min_count replies were received within max_wait time.

class rti.rpc.**Replier**(*request_type: Union[type,* DynamicType*]*, *reply_type: Union[type,* DynamicType*]*, *participant:* DomainParticipant, *service_name: Optional[str] = None*, *request_topic: Optional[Union[*Topic*,* ContentFilteredTopic*, str, object]] = None*, *reply_topic: Optional[Union[*Topic*, str, object]] = None*, *datawriter_qos: Optional[*DataWriterQos*] = None*, *datareader_qos: Optional[*DataReaderQos*] = None*, *publisher: Optional[*Publisher*] = None*, *subscriber: Optional[*Subscriber*] = None*, *on_request_available: Optional[Callable[[object], object]] = None*)

A replier object for handling request-reply interactions with DDS.

Parameters

- **request_type** – The type of the request data.
- **reply_type** – The type of the reply data.
- **participant** – The DomainParticipant that will hold the reply writer and request reader.
- **service_name** – Name that will be used to derive the topic name, defaults to None (rely only on custom topics).
- **request_topic** – Topic object or name that will be used for the request data, must be set if service_name is None, otherwise overrides service_name, defaults to None (use service_name).
- **reply_topic** – Topic object or name that will be used for the reply data, must be set if service_name is None, otherwise overrides service_name, defaults to None (use service_name).
- **datawriter_qos** – QoS object to use for reply writer, defaults to None (use default RequestReply QoS).
- **datareader_qos** – QoS object to use for request reader, defaults to None (use default RequestReply QoS).
- **publisher** – Publisher used to hold reply writer, defaults to None (use participant builtin publisher).

- **subscriber** – Subscriber used to hold request reader, defaults to None (use participant builtin subscriber).
- **on_reply_available** – The callback that handles incoming requests.

__init__(*request_type: Union[type,* DynamicType*]*, *reply_type: Union[type,* DynamicType*]*, *participant:* DomainParticipant, *service_name: Optional[str] = None*, *request_topic: Optional[Union[*Topic*,* ContentFilteredTopic*, str, object]] = None*, *reply_topic: Optional[Union[*Topic*, str, object]] = None*, *datawriter_qos: Optional[*DataWriterQos*] = None*, *datareader_qos: Optional[*DataReaderQos*] = None*, *publisher: Optional[*Publisher*] = None*, *subscriber: Optional[*Subscriber*] = None*, *on_request_available: Optional[Callable[[object], object]] = None*) → None

__weakref__

list of weak references to the object (if defined)

close() → None

Close the resources for this request-reply object.

property closed: bool

Returns true if this request-reply object has been closed.

Getter
Returns the number of matched requesters.

property matched_requester_count: int

The number of discovered matched requesters.

Getter
Returns the number of matched requesters.

property on_request_available

The listener callback used to process received requests.

Getter
Returns the callback function.

Setter
Set the callback function.

Type
Optional[Callable[[*Replier*]]]

read_requests() → Union[*LoanedSamples*, *LoanedSamples*]

Read received requests.

Returns
A loaned samples object containing the requests.

receive_requests(*max_wait:* Duration, *min_count: int = 1*) → Union[*LoanedSamples*, *LoanedSamples*]

Receive a minimum number of requests within a timeout period.

Parameters

- **max_wait** – Maximum time to wait for requests before timing out. .
- **min_count** – Minimum number of requests to receive, default 1.

Raises

dds.TimeoutError – Thrown if min_count not received within max_wait.

Returns

A loaned samples object containing the requests.

property reply_datawriter: Union[*DataWriter*, *DataWriter*]

The DataWriter used to send reply data.

Getter

Returns the reply DataWriter.

property request_datareader: Union[*DataReader*, *DataReader*]

The DataReader used to receive request data.

Getter

Returns the request DataReader.

send_reply(*reply: Union[*DynamicData*, object]*, *param: Union[*SampleIdentity, SampleInfo, WriteParams*]*, *final: bool = True*) → None

Send a reply to a received request.

Parameters

- **reply** – The reply to send.
- **param** – Parameters used for writing the request.
- **final** – Indicates whether this is the final reply for a specific request, default True.

Raises

dds.InvalidArgumentError – Thrown if param is not a type that can be used for correlation.

take_requests() → Union[*LoanedSamples*, *LoanedSamples*]

Take received requests.

Returns

A loaned samples object containing the requests.

Return type

Union[dds.DynamicData.LoanedSamples, object]

wait_for_requests(*max_wait:* Duration, *min_count: int = 1*) → bool

Wait for a minimum number of requests within a timeout period.

Parameters

- **max_wait** – Maximum time to wait for requests before timing out. .
- **min_count** – Minimum number of requests to receive, default 1.

Returns

Boolean indicating whether min_count requests were received within max_wait time.

async wait_for_requests_async(*max_wait:* Duration, *min_count: Optional[int] = 1*) → bool

Wait asynchronously for a minimum number of requests within a timeout period.

Parameters

- **max_wait** – Maximum time to wait for requests before timing out. .
- **min_count** – Minimum number of requests to receive, default 1.

Returns

Boolean indicating whether min_count requests were received within max_wait time.

class rti.rpc.**SimpleReplier**(*request_type: Union[*DynamicType*, type]*, *reply_type: Union[*DynamicType*, type]*, *participant:* DomainParticipant, *handler: Callable[[object], object]*, *service_name: Optional[str] = None*, *request_topic: Optional[Union[*Topic, ContentFilteredTopic*, str, object]] = None*, *reply_topic: Optional[Union[*Topic*, str, object]] = None*, *datawriter_qos: Optional[*DataWriterQos*] = None*, *datareader_qos: Optional[*DataReaderQos*] = None*, *publisher: Optional[*Publisher*] = None*, *subscriber: Optional[*Subscriber*] = None*)

A special replier that uses a user callback to produce one reply per request.

Parameters

- **request_type** – The type of the request data.
- **reply_type** – The type of the reply data.
- **participant** – The DomainParticipant that will hold the request reader and reply writer.
- **handler** – The callback that handles incoming requests and returns a reply. The callback must have a single argument of type `request_type` and must return an instance of type `reply_type`.
- **service_name** – Name that will be used to derive the topic name, defaults to None (rely only on custom topics).
- **request_topic** – Topic object or name that will be used for the request data, must be set if service_name is None, otherwise overrides service_name, defaults to None (use service_name).
- **reply_topic** – Topic object or name that will be used for the reply data, must be set if service_name is None, otherwise overrides service_name, defaults to None (use service_name).
- **datawriter_qos** – QoS object to use for reply writer, defaults to None (use default RequestReply QoS).

- **datareader_qos** – QoS object to use for request reader, defaults to None (use default RequestReply QoS).
- **publisher** – Publisher used to hold reply writer, defaults to None (use participant builtin publisher).
- **subscriber** – Subscriber used to hold request reader, defaults to None (use participant builtin subscriber).

__init__(*request_type: Union[*DynamicType*, type]*, *reply_type: Union[*DynamicType*, type]*, *participant:* DomainParticipant, *handler: Callable[[object], object]*, *service_name: Optional[str] = None*, *request_topic: Optional[Union[*Topic*,* ContentFilteredTopic*, str, object]] = None*, *reply_topic: Optional[Union[*Topic*, str, object]] = None*, *datawriter_qos: Optional[*DataWriterQos*] = None*, *datareader_qos: Optional[*DataReaderQos*] = None*, *publisher: Optional[*Publisher*] = None*, *subscriber: Optional[*Subscriber*] = None*) → None

__weakref__

list of weak references to the object (if defined)

close() → None

Close the resources for this request-reply object.

property closed: bool

Returns true if this request-reply object has been closed.

Getter

Returns the number of matched requesters.

property matched_requester_count: int

The number of discovered matched requesters.

Getter

Returns the number of matched requesters.

class rti.rpc.**Service**(*service_instance: ABC*, *participant:* DomainParticipant, *service_name: str*, *task_count: int = 4*, *datawriter_qos: Optional[*DataWriterQos*] = None*, *datareader_qos: Optional[*DataReaderQos*] = None*, *publisher: Optional[*Publisher*] = None*, *subscriber: Optional[*Subscriber*] = None*)

A service allows running a service_instance in a DDS domain using asyncio.

The service useses a Replier to receive RPC calls and then dispatches them to the service_instance, calling the appropriate method. The value returned by the method is then sent back to the remote caller.

The service runs asynchronously (run method) until the task is cancelled.

close()

Closes the DDS entities used by this service.

property matched_client_count: int

The number of RPC clients that match this service.

async run(*close_on_cancel: bool = False*)

Starts receiving RPC calls (requests) and processing them.

This method runs until the task it returns is cancelled.

If close_on_cancel is True, the service will close the DDS entities when the task is canceled. By default it is False, which means you can call run() again after a run() task is cancelled.

Exceptions raised during the execution of the service are logged as warnings module and do not stop the execution of the service.

class rti.rpc.**ClientBase**(*participant: ~rti.connextdds.DomainParticipant*, *service_name: str*, *max_wait_per_call: ~rti.connextdds.Duration = <rti.connextdds.Duration object>*, *datawriter_qos: ~typing.Optional[~rti.connextdds.DataWriterQos] = None*, *datareader_qos: ~typing.Optional[~rti.connextdds.DataReaderQos] = None*, *publisher: ~typing.Optional[~rti.connextdds.Publisher] = None*, *subscriber: ~typing.Optional[~rti.connextdds.Subscriber] = None*)

Base class for RPC clients.

An actual Client must inherit from a service interface and from this class, for example:

```
` class RobotClient(Robot, rpc.ClientBase): ... `
```

This base class injects an implementation for all the @operation methods found in Robot, which uses a Requester to make RPC calls and return the values it receives.

The base class also provides an __init__, close and other methods.

close()

Closes the DDS entities used by this client.

property matched_service_count: int

The number of RPC services that match this client.

@rti.rpc.**service**(*cls=None*, ***, *type_annotations=[]*, *member_annotations={}*)

This decorator marks an abstract base class as a remote service interface.

A class annotated with this decorator can be used to create a Client or to define the implementation to be run in a Service.

The operations that will be remotely callable need to be marked with the @operation decorator.

@rti.rpc.**operation**(*funcobj=None*, ***, *raises=[]*, *parameter_annotations={}*)

This decorator marks a method as an remote operation of a service interface.

It also marks it as an @abc.abstractmethod.

Only methods marked with this decorator will be callable using an RPC Client or an RPC Service.

exception rti.rpc.**RemoteUnknownOperationError**

Exception thrown by a client operation when the server indicates that the operation is unknown to the server.

exception rti.rpc.**RemoteUnknownExceptionError**

Exception thrown by a client operation when the server operation fails with an exception that is not declared in the interface.

2.3.4 rti.asyncio

Note: This module requires Python 3.7 or newer

This module must be imported in order to use the methods *rti.connextdds.DataReader.take_async()* and *rti.connextdds.DataReader.take_data_async()*.

These two methods are added to the DataReader class when this module is imported.

The module also defines a convenience function rti.asyncio.run, which is similar to asyncio.run, and can synchronously run the main async function in an application.

See *Subscriptions*.

rti.asyncio.**run**(*coroutine*)

Uses the current event loop to run the given coroutine and waits until it finishes. If there is no current running event loop, a new one is created

2.3.5 rti.logging

Submodules

rti.logging.distlog module

class rti.logging.distlog.**LogLevel**

Bases: pybind11_object

Members:

SILENT

FATAL

SEVERE

ERROR

WARNING

NOTICE

INFO

DEBUG

TRACE

DEBUG = <LogLevel.DEBUG: 700>

ERROR = <LogLevel.ERROR: 300>

FATAL = <LogLevel.FATAL: 100>

INFO = <LogLevel.INFO: 600>

NOTICE = <LogLevel.NOTICE: 500>

SEVERE = <LogLevel.SEVERE: 200>

SILENT = <LogLevel.SILENT: 0>

TRACE = <LogLevel.TRACE: 800>

WARNING = <LogLevel.WARNING: 400>

property name

property value

class rti.logging.distlog.**Logger**

Bases: pybind11_object

static debug(*message: str*) → None

Log a debug message.

static error(*message: str*) → None

Log an error message.

static fatal(*message: str*) → None

Log a fatal message.

static filter_level(*level:* rti.logging.distlog.LogLevel) → None

The logger filter level.

static finalize() → None

Destroy the Logger. It should not be accessed after this call.

static info(*message: str*) → None

Log an info message.

static init(*options: Optional[*rti.logging.distlog.LoggerOptions*] = None*) → None

Initializes the distributed logger

static log(**args*, ***kwargs*)

Overloaded function.

1. log(log_level: rti.logging.distlog.LogLevel, message: str) -> None

Log a message with the given log level.

2. log(log_level: rti.logging.distlog.LogLevel, message: str, category: str) -> None

Log a message with the given log level and category.

3. log(message_params: rti.logging.distlog.MessageParams) -> None

Log a message with the given message parameters.

static notice(*message: str*) → None

Log a notice message.

static print_format(*format:* rti.connextdds.PrintFormat) → None

The logger print format.NOTE: This will affect the print format of the associatedDomainParticipant's logger as well.

static severe(*message: str*) → None

Log a severe message.

static trace(*message: str*) → None

Log a trace message.

static verbosity(*category:* rti.connextdds.LogCategory, *level:* rti.connextdds.Verbosity) → None

The logger's verbosity.NOTE: This will affect the verbosity of the associatedDomainParticipant's logger as well.

static warning(*message: str*) → None

Log a warning message.

class rti.logging.distlog.**LoggerOptions**

Bases: pybind11_object

property application_kind

The application_kind.

property domain_id

The domain ID for logging.

property echo_to_stdout

Toggle for echo to stdout.

property filter_level

Toggle for log filter level.

property log_infrastructure_messages

Toggle for logging infrastructure messages.

property participant

The DomainParticipant to use for the logger.

property qos_library

The QoS library name.

property qos_profile

The QoS profile name.

property queue_size
The logger's queue size.

property remote_administration_enabled
Toggle for remote administration.

property thread_settings
The settings for the thread handling logging.

class rti.logging.distlog.**MessageParams**
Bases: pybind11_object

property category
The log message category.

property log_level
The message log level.

property message
The log message.

property timestamp
The timestamp of the log message.

rti.logging.handler module

class rti.logging.handler.**DistlogHandler**(*options: Optional[*LoggerOptions*] = None*)
Bases: Handler

close() → None
Tidy up any resources used by the handler.

This version removes the handler from an internal map of handlers, _handlers, which is used for handler lookup by name. Subclasses should ensure that this gets called from overridden close() methods.

emit(*record: LogRecord*) → None
Do whatever it takes to actually log the specified logging record.

This version is intended to be implemented by subclasses and so raises a NotImplementedError.

Module contents

- *rti.connextdds* - the main package, containing the DDS API.
- *rti.types* - allows defining user types to be published and subscribed to.
- *rti.rpc* - contains the and Request-Reply and Remote Procedure APIs.
- *rti.asyncio* - adds support for asynchronous operations.
- *rti.logging* - provides logging utilities.

2.4 Examples

In addition to the code snippets in the RTI Connext Getting Started Guide and the *API Overview*, you can find full example applications in the Connext Examples repository, including the following:

- Content-Filtered Topic
- Content-Filtered Topic String Filter
- Built-in Topics
- Using Qos Profiles
- Partitions
- Group Coherent Presentation
- WaitSets and Query Conditions
- Asynchronous Publication
- Request-Reply
- Remote Procedure Call
- Basic Security

2.5 Copyrights, Notices, License

Trademarks

Copy and Use Restrictions

The security features of this product include software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/). This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

Notices

Deprecations and Removals

Any deprecations or removals noted in this document serve as notice under the Real-Time Innovations, Inc. Maintenance Policy #4220 and/or any other agreements by and between RTI and customer regarding maintenance and support of RTI's software.

Deprecated means that the item is still supported in the release, but will be removed in a future release. *Removed* means that the item is discontinued or no longer supported. By specifying that an item is deprecated in a release, RTI hereby provides customer notice that RTI reserves the right after one year from the date of such release and, with or without further notice, to immediately terminate maintenance (including without limitation, providing updates and upgrades) for the item, and no longer support the item, in a future release.

Technical Support Real-Time Innovations, Inc. 232 E. Java Drive Sunnyvale, CA 94089 Phone: (408) 990-7444 Email: support@rti.com Website: https://support.rti.com/

Chapter 3

Additional documentation

For more documentation, including the Connext User's Manual and the reference for the C, C++, Java and C# APIs, see RTI Community.

Python Module Index

r

Index

Non-alphabetical

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z