

RMW desert

1.0

Generated by Doxygen 1.9.8



<b>1 Namespace Index</b>	<b>1</b>
1.1 Namespace List	1
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Namespace Documentation</b>	<b>9</b>
5.1 CStringHelper Namespace Reference	9
5.1.1 Detailed Description	10
5.1.2 Function Documentation	10
5.1.2.1 assign_string()	10
5.1.2.2 assign_u16string()	10
5.1.2.3 assign_vector_string()	10
5.1.2.4 assign_vector_string_to_sequence()	11
5.1.2.5 assign_vector_u16string()	11
5.1.2.6 assign_vector_u16string_to_sequence()	11
5.1.2.7 convert_sequence_to_std_vector_string()	12
5.1.2.8 convert_sequence_to_std_vector_u16string()	12
5.1.2.9 convert_to_std_string()	13
5.1.2.10 convert_to_std_u16string()	13
5.1.2.11 convert_to_std_vector_string()	13
5.1.2.12 convert_to_std_vector_u16string()	14
5.2 Discovery Namespace Reference	14
5.2.1 Detailed Description	15
5.2.2 Function Documentation	15
5.2.2.1 demangle_publisher_from_topic()	15
5.2.2.2 demangle_service_from_topic()	16
5.2.2.3 demangle_service_reply_from_topic()	16
5.2.2.4 demangle_service_request_from_topic()	16
5.2.2.5 demangle_subscriber_from_topic()	17
5.2.2.6 demangle_topic()	17
5.2.2.7 discovery_thread()	17
5.2.2.8 discovery_thread_start()	18
5.2.2.9 discovery_thread_stop()	18
5.2.2.10 identity_demangle()	18
5.2.2.11 resolve_prefix()	19
5.2.2.12 send_discovery_beacon()	19
5.2.2.13 send_discovery_request()	20

5.3 MessageSerialization Namespace Reference . . . . .	20
5.3.1 Detailed Description . . . . .	21
5.3.2 Function Documentation . . . . .	21
5.3.2.1 deserialize() . . . . .	21
5.3.2.2 deserialize_field() [1/2] . . . . .	21
5.3.2.3 deserialize_field() [2/2] . . . . .	22
5.3.2.4 serialize() . . . . .	22
5.3.2.5 serialize_field() [1/2] . . . . .	22
5.3.2.6 serialize_field() [2/2] . . . . .	23
<b>6 Class Documentation</b>	<b>25</b>
6.1 CircularQueue< T, MaxLen, Container > Class Template Reference . . . . .	25
6.2 DesertClient Class Reference . . . . .	26
6.2.1 Constructor & Destructor Documentation . . . . .	26
6.2.1.1 DesertClient() . . . . .	26
6.2.2 Member Function Documentation . . . . .	27
6.2.2.1 get_gid() . . . . .	27
6.2.2.2 get_request_type_name() . . . . .	27
6.2.2.3 get_response_type_name() . . . . .	27
6.2.2.4 get_service_name() . . . . .	27
6.2.2.5 has_data() . . . . .	28
6.2.2.6 read_response() . . . . .	28
6.2.2.7 send_request() . . . . .	28
6.3 DesertGuardCondition Class Reference . . . . .	29
6.3.1 Member Function Documentation . . . . .	29
6.3.1.1 get_has_triggered() . . . . .	29
6.3.1.2 has_triggered() . . . . .	29
6.3.1.3 trigger() . . . . .	30
6.4 DesertNode Class Reference . . . . .	30
6.4.1 Constructor & Destructor Documentation . . . . .	30
6.4.1.1 DesertNode() . . . . .	30
6.4.2 Member Function Documentation . . . . .	31
6.4.2.1 add_client() . . . . .	31
6.4.2.2 add_publisher() . . . . .	31
6.4.2.3 add_service() . . . . .	31
6.4.2.4 add_subscriber() . . . . .	32
6.4.2.5 get_gid() . . . . .	32
6.4.2.6 remove_client() . . . . .	32
6.4.2.7 remove_publisher() . . . . .	32
6.4.2.8 remove_service() . . . . .	33
6.4.2.9 remove_subscriber() . . . . .	33
6.5 DesertPublisher Class Reference . . . . .	33

6.5.1 Constructor & Destructor Documentation	34
6.5.1.1 DesertPublisher()	34
6.5.2 Member Function Documentation	34
6.5.2.1 get_gid()	34
6.5.2.2 get_topic_name()	34
6.5.2.3 get_type_name()	35
6.5.2.4 push()	35
6.6 DesertService Class Reference	35
6.6.1 Constructor & Destructor Documentation	36
6.6.1.1 DesertService()	36
6.6.2 Member Function Documentation	36
6.6.2.1 get_gid()	36
6.6.2.2 get_request_type_name()	36
6.6.2.3 get_response_type_name()	37
6.6.2.4 get_service_name()	37
6.6.2.5 has_data()	37
6.6.2.6 read_request()	37
6.6.2.7 send_response()	38
6.7 DesertSubscriber Class Reference	38
6.7.1 Constructor & Destructor Documentation	39
6.7.1.1 DesertSubscriber()	39
6.7.2 Member Function Documentation	39
6.7.2.1 get_gid()	39
6.7.2.2 get_topic_name()	39
6.7.2.3 get_type_name()	39
6.7.2.4 has_data()	40
6.7.2.5 read_data()	40
6.8 DesertWaitset Class Reference	40
6.9 GenericCSequence< T > Struct Template Reference	40
6.10 rmw_context_impl_s Struct Reference	41
6.11 cbor::RxStream Class Reference	41
6.11.1 Constructor & Destructor Documentation	42
6.11.1.1 RxStream()	42
6.11.2 Member Function Documentation	42
6.11.2.1 clear_buffer()	42
6.11.2.2 data_available()	43
6.11.2.3 deserialize_integer()	43
6.11.2.4 deserialize_sequence()	43
6.11.2.5 get_identifier()	44
6.11.2.6 get_name()	44
6.11.2.7 get_type()	44
6.11.2.8 interpret_packets()	44

6.11.2.9 operator>>() [1/16]	44
6.11.2.10 operator>>() [2/16]	45
6.11.2.11 operator>>() [3/16]	45
6.11.2.12 operator>>() [4/16]	45
6.11.2.13 operator>>() [5/16]	45
6.11.2.14 operator>>() [6/16]	46
6.11.2.15 operator>>() [7/16]	46
6.11.2.16 operator>>() [8/16]	46
6.11.2.17 operator>>() [9/16]	46
6.11.2.18 operator>>() [10/16]	47
6.11.2.19 operator>>() [11/16]	47
6.11.2.20 operator>>() [12/16]	47
6.11.2.21 operator>>() [13/16]	47
6.11.2.22 operator>>() [14/16]	48
6.11.2.23 operator>>() [15/16]	48
6.11.2.24 operator>>() [16/16]	48
6.11.2.25 push_packet()	48
6.12 TcpDaemon Class Reference	49
6.12.1 Member Function Documentation	49
6.12.1.1 enqueue_packet()	49
6.12.1.2 init()	49
6.12.1.3 read_packet()	50
6.13 TopicsConfig Class Reference	50
6.13.1 Member Function Documentation	50
6.13.1.1 get_identifier_topic()	50
6.13.1.2 get_topic_identifier()	51
6.13.1.3 load_configuration()	51
6.14 cbor::TxStream Class Reference	51
6.14.1 Constructor & Destructor Documentation	52
6.14.1.1 TxStream()	52
6.14.2 Member Function Documentation	52
6.14.2.1 end_transmission()	52
6.14.2.2 operator<<() [1/16]	53
6.14.2.3 operator<<() [2/16]	53
6.14.2.4 operator<<() [3/16]	53
6.14.2.5 operator<<() [4/16]	53
6.14.2.6 operator<<() [5/16]	54
6.14.2.7 operator<<() [6/16]	54
6.14.2.8 operator<<() [7/16]	54
6.14.2.9 operator<<() [8/16]	54
6.14.2.10 operator<<() [9/16]	54
6.14.2.11 operator<<() [10/16]	55

6.14.2.12 operator<<() [11/16]	55
6.14.2.13 operator<<() [12/16]	55
6.14.2.14 operator<<() [13/16]	55
6.14.2.15 operator<<() [14/16]	57
6.14.2.16 operator<<() [15/16]	57
6.14.2.17 operator<<() [16/16]	57
6.14.2.18 serialize_sequence()	57
6.14.2.19 start_transmission() [1/2]	58
6.14.2.20 start_transmission() [2/2]	58
<b>7 File Documentation</b>	<b>59</b>
7.1 src/desert_classes/CBorStream.h File Reference	59
7.1.1 Detailed Description	60
7.2 CBorStream.h	60
7.3 src/desert_classes/CStringHelper.h File Reference	63
7.3.1 Detailed Description	64
7.4 CStringHelper.h	64
7.5 src/desert_classes/demangle.h File Reference	65
7.5.1 Detailed Description	67
7.6 demangle.h	67
7.7 src/desert_classes/DesertClient.h File Reference	68
7.7.1 Detailed Description	69
7.8 DesertClient.h	69
7.9 src/desert_classes/DesertGuardCondition.h File Reference	70
7.9.1 Detailed Description	70
7.10 DesertGuardCondition.h	70
7.11 src/desert_classes/DesertNode.h File Reference	71
7.11.1 Detailed Description	71
7.12 DesertNode.h	72
7.13 src/desert_classes/DesertPublisher.h File Reference	73
7.13.1 Detailed Description	73
7.14 DesertPublisher.h	74
7.15 src/desert_classes/DesertService.h File Reference	74
7.15.1 Detailed Description	75
7.16 DesertService.h	76
7.17 src/desert_classes/DesertSubscriber.h File Reference	77
7.17.1 Detailed Description	77
7.18 DesertSubscriber.h	78
7.19 src/desert_classes/DesertWaitSet.h File Reference	78
7.19.1 Detailed Description	79
7.20 DesertWaitSet.h	79
7.21 src/desert_classes/Discovery.h File Reference	79

---

7.21.1 Detailed Description . . . . .	81
7.22 Discovery.h . . . . .	81
7.23 src/desert_classes/macros.h File Reference . . . . .	82
7.23.1 Detailed Description . . . . .	82
7.23.2 Macro Definition Documentation . . . . .	82
7.23.2.1 SPECIALIZE_GENERIC_C_SEQUENCE . . . . .	82
7.24 macros.h . . . . .	83
7.25 src/desert_classes/MessageSerialization.h File Reference . . . . .	83
7.25.1 Detailed Description . . . . .	85
7.26 MessageSerialization.h . . . . .	85
7.27 src/desert_classes/rmw_context_impl_s.h File Reference . . . . .	90
7.27.1 Detailed Description . . . . .	91
7.28 rmw_context_impl_s.h . . . . .	91
7.29 src/desert_classes/TcpDaemon.h File Reference . . . . .	91
7.29.1 Detailed Description . . . . .	92
7.30 TcpDaemon.h . . . . .	93
7.31 src/desert_classes/TopicsConfig.h File Reference . . . . .	93
7.31.1 Detailed Description . . . . .	94
7.32 TopicsConfig.h . . . . .	95
<b>Index</b>	<b>97</b>



# Chapter 1

## Namespace Index

### 1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

<a href="#">CStringHelper</a>		
	Namespace containing C sequence handling functions . . . . .	9
<a href="#">Discovery</a>		
	Namespace containing discovery functions . . . . .	14
<a href="#">MessageSerialization</a>		
	Namespace containing serialization functions . . . . .	20



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

DesertClient . . . . .	26
DesertGuardCondition . . . . .	29
DesertNode . . . . .	30
DesertPublisher . . . . .	33
DesertService . . . . .	35
DesertSubscriber . . . . .	38
DesertWaitset . . . . .	40
GenericCSequence< T > . . . . .	40
std::queue	
CircularQueue< std::vector< std::pair< void *, int > >, MAX_BUFFER_CAPACITY > . . . . .	25
CircularQueue< T, MaxLen, Container > . . . . .	25
rmw_context_impl_s . . . . .	41
cbor::RxStream . . . . .	41
TcpDaemon . . . . .	49
TopicsConfig . . . . .	50
cbor::TxStream . . . . .	51



## Chapter 3

# Class Index

### 3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">CircularQueue&lt; T, MaxLen, Container &gt;</a>	25
<a href="#">DesertClient</a>	26
<a href="#">DesertGuardCondition</a>	29
<a href="#">DesertNode</a>	30
<a href="#">DesertPublisher</a>	33
<a href="#">DesertService</a>	35
<a href="#">DesertSubscriber</a>	38
<a href="#">DesertWaitset</a>	40
<a href="#">GenericCSequence&lt; T &gt;</a>	40
<a href="#">rmw_context_impl_s</a>	41
<a href="#">cbor::RxStream</a>	41
<a href="#">TcpDaemon</a>	49
<a href="#">TopicsConfig</a>	50
<a href="#">cbor::TxStream</a>	51



## Chapter 4

# File Index

### 4.1 File List

Here is a list of all documented files with brief descriptions:

src/desert_classes/ <a href="#">CBorStream.h</a>	59
Classes used to convert data types into a CBOR encoded stream . . . . .	
src/desert_classes/ <a href="#">CStringHelper.h</a>	63
Namespace containing C sequence handling functions . . . . .	
src/desert_classes/ <a href="#">demangle.h</a>	65
Functions used to demangle topic names during discovery operations . . . . .	
src/desert_classes/ <a href="#">DesertClient.h</a>	68
Implementation of the Client structure for DESERT . . . . .	
src/desert_classes/ <a href="#">DesertGuardCondition.h</a>	70
Implementation of the GuardCondition structure for DESERT . . . . .	
src/desert_classes/ <a href="#">DesertNode.h</a>	71
Implementation of the Node structure for DESERT . . . . .	
src/desert_classes/ <a href="#">DesertPublisher.h</a>	73
Implementation of the Publisher structure for DESERT . . . . .	
src/desert_classes/ <a href="#">DesertService.h</a>	74
Implementation of the Service structure for DESERT . . . . .	
src/desert_classes/ <a href="#">DesertSubscriber.h</a>	77
Implementation of the Subscriber structure for DESERT . . . . .	
src/desert_classes/ <a href="#">DesertWaitSet.h</a>	78
Implementation of the WaitSet structure for DESERT . . . . .	
src/desert_classes/ <a href="#">Discovery.h</a>	79
Namespace used to provide discovery functionalities . . . . .	
src/desert_classes/ <a href="#">macros.h</a>	82
Header containing C sequence macros . . . . .	
src/desert_classes/ <a href="#">MessageSerialization.h</a>	83
Namespace containing serialization functions . . . . .	
src/desert_classes/ <a href="#">rmw_context_impl_s.h</a>	90
Implementation for the context variable . . . . .	
src/desert_classes/ <a href="#">TcpDaemon.h</a>	91
Class used to send and receive data from the DESERT socket . . . . .	
src/desert_classes/ <a href="#">TopicsConfig.h</a>	93
Class used to store configurations . . . . .	





## Chapter 5

# Namespace Documentation

### 5.1 CStringHelper Namespace Reference

Namespace containing C sequence handling functions.

#### Functions

- `std::string` [convert\\_to\\_std\\_string](#) (void \*str)  
*Convert a `rosidl_runtime_c__String` into `std::string`.*
- `std::vector< std::string >` [convert\\_to\\_std\\_vector\\_string](#) (void \*str\_array, size\_t size)  
*Convert a `rosidl_runtime_c__String` into a vector of `std::string`.*
- `std::vector< std::string >` [convert\\_sequence\\_to\\_std\\_vector\\_string](#) (void \*str\_seq)  
*Convert a `rosidl_runtime_c__String__Sequence` into a vector of `std::string`.*
- `std::u16string` [convert\\_to\\_std\\_u16string](#) (void \*str)  
*Convert a `rosidl_runtime_c__U16String` into `std::u16string`.*
- `std::vector< std::u16string >` [convert\\_to\\_std\\_vector\\_u16string](#) (void \*str\_array, size\_t size)  
*Convert a `rosidl_runtime_c__U16String` into a vector of `std::u16string`.*
- `std::vector< std::u16string >` [convert\\_sequence\\_to\\_std\\_vector\\_u16string](#) (void \*str\_seq)  
*Convert a `rosidl_runtime_c__U16String__Sequence` into a vector of `std::u16string`.*
- void [assign\\_string](#) (std::string str, void \*field)  
*Assigning to a `rosidl_runtime_c__String` the value contained in a `std::string`.*
- void [assign\\_vector\\_string](#) (std::vector< std::string > cpp\_string\_vector, void \*str\_array, size\_t size)  
*Assigning to a `rosidl_runtime_c__String` the value contained in a vector of `std::string`.*
- void [assign\\_vector\\_string\\_to\\_sequence](#) (std::vector< std::string > cpp\_string\_vector, void \*str\_seq)  
*Assigning to a `rosidl_runtime_c__String__Sequence` the value contained in a vector of `std::string`.*
- void [assign\\_u16string](#) (std::u16string str, void \*field)  
*Assigning to a `rosidl_runtime_c__U16String` the value contained in a `std::u16string`.*
- void [assign\\_vector\\_u16string](#) (std::vector< std::u16string > cpp\_string\_vector, void \*str\_array, size\_t size)  
*Assigning to a `rosidl_runtime_c__U16String` the value contained in a vector of `std::u16string`.*
- void [assign\\_vector\\_u16string\\_to\\_sequence](#) (std::vector< std::u16string > cpp\_string\_vector, void \*str\_seq)  
*Assigning to a `rosidl_runtime_c__U16String__Sequence` the value contained in a vector of `std::u16string`.*

### 5.1.1 Detailed Description

Namespace containing C sequence handling functions.

The C data type implementation is more complicated than the C++ one, because complex types like vectors have to be manually managed and this header contains functions to convert C strings and generic sequences into respectively C++ strings and vectors.

### 5.1.2 Function Documentation

#### 5.1.2.1 `assign_string()`

```
void CStringHelper::assign_string (
    std::string str,
    void * field )
```

Assing to a `rosidl_runtime_c__String` the value contained in a `std::string`.

This function stores the data contained in a C++ string in a `rosidl_runtime_c__String` pointed by the `field` parameter.

##### Parameters

<i>str</i>	C++ style string containing data
<i>field</i>	Pointer containing the destination of the string

#### 5.1.2.2 `assign_u16string()`

```
void CStringHelper::assign_u16string (
    std::u16string str,
    void * field )
```

Assing to a `rosidl_runtime_c__U16String` the value contained in a `std::u16string`.

This function stores the data contained in a C++ `u16string` in a `rosidl_runtime_c__U16String` pointed by the `field` parameter.

##### Parameters

<i>str</i>	C++ style <code>u16string</code> containing data
<i>field</i>	Pointer containing the destination of the <code>u16string</code>

#### 5.1.2.3 `assign_vector_string()`

```
void CStringHelper::assign_vector_string (
    std::vector< std::string > cpp_string_vector,
    void * str_array,
    size_t size )
```

Assing to a `rosidl_runtime_c__String` the value contained in a vector of `std::string`.

This function stores the data contained in a C++ vector of strings in a `rosidl_runtime_c__String` fixed size sequence pointed by the `str_array` parameter.

#### Parameters

<i>cpp_string_vector</i>	C++ style vector of string containing data
<i>str_array</i>	Pointer containing the destination of the string sequence
<i>size</i>	Number of elements in the array

#### 5.1.2.4 assign\_vector\_string\_to\_sequence()

```
void CStringHelper::assign_vector_string_to_sequence (
    std::vector< std::string > cpp_string_vector,
    void * str_seq )
```

Assing to a `rosidl_runtime_c__String__Sequence` the value contained in a vector of `std::string`.

This function stores the data contained in a C++ vector of strings in a `rosidl_runtime_c__String__Sequence` variable size sequence pointed by the `str_array` parameter.

#### Parameters

<i>cpp_string_vector</i>	C++ style vector of string containing data
<i>str_seq</i>	Pointer containing the destination of the string sequence

#### 5.1.2.5 assign\_vector\_u16string()

```
void CStringHelper::assign_vector_u16string (
    std::vector< std::u16string > cpp_string_vector,
    void * str_array,
    size_t size )
```

Assing to a `rosidl_runtime_c__U16String` the value contained in a vector of `std::u16string`.

This function stores the data contained in a C++ vector of u16strings in a `rosidl_runtime_c__U16String` fixed size sequence pointed by the `str_array` parameter.

#### Parameters

<i>cpp_string_vector</i>	C++ style vector of u16strings containing data
<i>str_array</i>	Pointer containing the destination of the u16string sequence
<i>size</i>	Number of elements in the array

#### 5.1.2.6 assign\_vector\_u16string\_to\_sequence()

```
void CStringHelper::assign_vector_u16string_to_sequence (
```

```
std::vector< std::u16string > cpp_string_vector,
void * str_seq )
```

Assigning to a `rosidl_runtime_c__U16String__Sequence` the value contained in a vector of `std::u16string`.

This function stores the data contained in a C++ vector of u16strings in a `rosidl_runtime_c__U16String__Sequence` variable size sequence pointed by the `str_array` parameter.

#### Parameters

<code>cpp_string_vector</code>	C++ style vector of u16strings containing data
<code>str_seq</code>	Pointer containing the destination of the u16string sequence

#### 5.1.2.7 `convert_sequence_to_std_vector_string()`

```
std::vector< std::string > CStringHelper::convert_sequence_to_std_vector_string (
void * str_seq )
```

Convert a `rosidl_runtime_c__String__Sequence` into a vector of `std::string`.

This function converts a `rosidl_runtime_c__String__Sequence` variable size sequence into a C++ vector of strings.

#### Parameters

<code>str_seq</code>	Pointer to the first original C-style string
----------------------	--

#### Returns

A C++ vector of strings

#### 5.1.2.8 `convert_sequence_to_std_vector_u16string()`

```
std::vector< std::u16string > CStringHelper::convert_sequence_to_std_vector_u16string (
void * str_seq )
```

Convert a `rosidl_runtime_c__U16String__Sequence` into a vector of `std::u16string`.

This function converts a `rosidl_runtime_c__U16String__Sequence` variable size sequence into a C++ vector of u16string.

#### Parameters

<code>str_seq</code>	Pointer to the first original C-style u16string
----------------------	---

#### Returns

A C++ vector of u16string

#### 5.1.2.9 convert\_to\_std\_string()

```
std::string CStringHelper::convert_to_std_string (
    void * str )
```

Convert a `rosidl_runtime_c__String` into `std::string`.

This function converts a `rosidl_runtime_c__String` into a C++ string.

##### Parameters

<i>str</i>	The original C-style string
------------	-----------------------------

##### Returns

A C++ string

#### 5.1.2.10 convert\_to\_std\_u16string()

```
std::u16string CStringHelper::convert_to_std_u16string (
    void * str )
```

Convert a `rosidl_runtime_c__U16String` into `std::u16string`.

This function converts a `rosidl_runtime_c__U16String` into a C++ u16string.

##### Parameters

<i>str</i>	The original C-style u16string
------------	--------------------------------

##### Returns

A C++ u16string

#### 5.1.2.11 convert\_to\_std\_vector\_string()

```
std::vector< std::string > CStringHelper::convert_to_std_vector_string (
    void * str_array,
    size_t size )
```

Convert a `rosidl_runtime_c__String` into a vector of `std::string`.

This function converts a `rosidl_runtime_c__String` fixed size sequence into a C++ vector of strings.

##### Parameters

<i>str_array</i>	Pointer to the first original C-style string
<i>size</i>	Number of elements in the array

**Returns**

A C++ vector of strings

**5.1.2.12 convert\_to\_std\_vector\_u16string()**

```
std::vector< std::u16string > CStringHelper::convert_to_std_vector_u16string (
    void * str_array,
    size_t size )
```

Convert a `rosidl_runtime_c__U16String` into a vector of `std::u16string`.

This function converts a `rosidl_runtime_c__U16String` fixed size sequence into a C++ vector of `u16string`.

**Parameters**

<i>str_array</i>	Pointer to the first original C-style u16string
<i>size</i>	Number of elements in the array

**Returns**

A C++ vector of u16strings

**5.2 Discovery Namespace Reference**

Namespace containing discovery functions.

**Typedefs**

- using **DemangleFunction** = `std::string (*)(const std::string &)`

**Functions**

- `char * integer_to_string (int x)`
- `std::string resolve_prefix (const std::string &name, const std::string &prefix)`  
*Resolve a prefix.*
- `std::string demangle_publisher_from_topic (const std::string &topic_name)`  
*Demangle a publisher.*
- `std::string demangle_subscriber_from_topic (const std::string &topic_name)`  
*Demangle a subscriber.*
- `std::string demangle_topic (const std::string &topic_name)`  
*Demangle a topic.*
- `std::string demangle_service_request_from_topic (const std::string &topic_name)`  
*Demangle a service request.*
- `std::string demangle_service_reply_from_topic (const std::string &topic_name)`  
*Demangle a service reply.*
- `std::string demangle_service_from_topic (const std::string &topic_name)`  
*Demangle a service.*

- `std::string identity_demangle` (const `std::string` &name)  
*No demangle.*
- `void discovery_thread` (`rmw_context_impl_t` \*impl)  
*Thread handling discovery beacons.*
- `rmw_ret_t discovery_thread_start` (`rmw_context_impl_t` \*impl)  
*Initialize the discovery thread.*
- `rmw_ret_t discovery_thread_stop` (`rmw_context_impl_t` \*impl)  
*Stop the discovery thread.*
- `void send_discovery_beacon` (`cbor::TxStream` stream, `std::string` node\_name, `std::string` node\_namespace, `int` entity\_type, `rmw_gid_t` entity\_gid, `std::string` topic\_name, `std::string` type\_name, `bool` disconnect)  
*Send a discovery beacon.*
- `void send_discovery_request` (`cbor::TxStream` stream)  
*Send a discovery request.*

## Variables

- `const char *const ros_topic_publisher_prefix` = `integer_to_string(PUBLISHER_TYPE)`
- `const char *const ros_topic_subscriber_prefix` = `integer_to_string(SUBSCRIBER_TYPE)`
- `const char *const ros_service_requester_prefix` = `integer_to_string(CLIENT_TYPE)`
- `const char *const ros_service_response_prefix` = `integer_to_string(SERVICE_TYPE)`

## 5.2.1 Detailed Description

Namespace containing discovery functions.

The middleware layer of a ROS stack must implement functionalities used to inform each node of the network structure of the other nodes connected, with their names and topics. Since this operation is quite resource-consuming and the underwater channel has a limited bandwidth, it is possible to disable it.

## 5.2.2 Function Documentation

### 5.2.2.1 demangle\_publisher\_from\_topic()

```
std::string Discovery::demangle_publisher_from_topic (
    const std::string & topic_name )
```

Demangle a publisher.

Return the topic name for a given topic if it is part of a publisher, else "".

#### Parameters

<i>topic_name</i>	Mangled topic name
-------------------	--------------------

#### Returns

Demangled topic name

### 5.2.2.2 demangle\_service\_from\_topic()

```
std::string Discovery::demangle_service_from_topic (
    const std::string & topic_name )
```

Demangle a service.

Return the service name for a given topic if it is part of a service, else "".

#### Parameters

<i>topic_name</i>	Mangled topic name
-------------------	--------------------

#### Returns

Demangled topic name

### 5.2.2.3 demangle\_service\_reply\_from\_topic()

```
std::string Discovery::demangle_service_reply_from_topic (
    const std::string & topic_name )
```

Demangle a service reply.

Return the service name for a given topic if it is part of a service reply, else "".

#### Parameters

<i>topic_name</i>	Mangled topic name
-------------------	--------------------

#### Returns

Demangled topic name

### 5.2.2.4 demangle\_service\_request\_from\_topic()

```
std::string Discovery::demangle_service_request_from_topic (
    const std::string & topic_name )
```

Demangle a service request.

Return the service name for a given topic if it is part of a service request, else "".

#### Parameters

<i>topic_name</i>	Mangled topic name
-------------------	--------------------



**Returns**

Demangled topic name

**5.2.2.5 demangle\_subscriber\_from\_topic()**

```
std::string Discovery::demangle_subscriber_from_topic (
    const std::string & topic_name )
```

Demangle a subscriber.

Return the topic name for a given topic if it is part of a subscriber, else "".

**Parameters**

<i>topic_name</i>	Mangled topic name
-------------------	--------------------

**Returns**

Demangled topic name

**5.2.2.6 demangle\_topic()**

```
std::string Discovery::demangle_topic (
    const std::string & topic_name )
```

Demangle a topic.

Return the topic name for a given topic if it is part of one, else "".

**Parameters**

<i>topic_name</i>	Mangled topic name
-------------------	--------------------

**Returns**

Demangled topic name

**5.2.2.7 discovery\_thread()**

```
void Discovery::discovery_thread (
    rmw_context_impl_t * impl )
```

Thread handling discovery beacons.

This function allows the middleware to receive and process incoming discovery beacons from the other nodes.

**Parameters**

<i>impl</i>	The middleware context implementation
-------------	---------------------------------------

**5.2.2.8 discovery\_thread\_start()**

```
rmw_ret_t Discovery::discovery_thread_start (
    rmw_context_impl_t * impl )
```

Initialize the discovery thread.

This function is called during the initialization of the middleware and starts the discovery thread.

**Parameters**

<i>impl</i>	The middleware context implementation
-------------	---------------------------------------

**Returns**

Outcome of the operation

**5.2.2.9 discovery\_thread\_stop()**

```
rmw_ret_t Discovery::discovery_thread_stop (
    rmw_context_impl_t * impl )
```

Stop the discovery thread.

This function is called during the termination of the middleware and stops the discovery thread.

**Parameters**

<i>impl</i>	The middleware context implementation
-------------	---------------------------------------

**Returns**

Outcome of the operation

**5.2.2.10 identity\_demangle()**

```
std::string Discovery::identity_demangle (
    const std::string & name )
```

No demangle.

Used when ros names are not mangled.

## Parameters

<i>name</i>	Topic name
-------------	------------

## Returns

Same topic name

**5.2.2.11 resolve\_prefix()**

```
std::string Discovery::resolve_prefix (
    const std::string & name,
    const std::string & prefix )
```

Resolve a prefix.

Returns `name` stripped of `prefix`.

## Parameters

<i>name</i>	Mangled topic name
<i>prefix</i>	Prefix of the entity type

## Returns

Demangled topic name

**5.2.2.12 send\_discovery\_beacon()**

```
void Discovery::send_discovery_beacon (
    cbor::TxStream stream,
    std::string node_name,
    std::string node_namespace,
    int entity_type,
    rmw_gid_t entity_gid,
    std::string topic_name,
    std::string type_name,
    bool disconnect )
```

Send a discovery beacon.

This function sends a beacon in the underwater channel containing all the informations related to a specific entity of a node.

## Parameters

<i>stream</i>	The stream used to send data
<i>node_name</i>	The name of the node holding the entity
<i>node_namespace</i>	The namespace of the node holding the entity
<i>entity_type</i>	The type of the entity

## Parameters

<i>entity_gid</i>	The global identifier of the entity
<i>topic_name</i>	The topic name
<i>type_name</i>	The topic type
<i>disconnect</i>	Flag used to determine if an entity is connecting or disconnecting

**5.2.2.13 send\_discovery\_request()**

```
void Discovery::send_discovery_request (
    cbor::TxStream stream )
```

Send a discovery request.

This function sends a request in the underwater channel to all the nodes requiring them to send their discovery beacons.

## Parameters

<i>stream</i>	The stream used to send data
---------------	------------------------------

**5.3 MessageSerialization Namespace Reference**

Namespace containing serialization functions.

**Functions**

- `template<typename T >`  
`void serialize_field (const INTROSPECTION_CPP_MEMBER *member, void *field, cbor::TxStream &stream)`  
*Serialize a C++ field.*
- `template<typename T >`  
`void serialize_field (const INTROSPECTION_C_MEMBER *member, void *field, cbor::TxStream &stream)`  
*Serialize a C field.*
- `template<typename MembersType >`  
`void serialize (const void *msg, const MembersType *casted_members, cbor::TxStream &stream)`  
*Serialize a ROS message, request or response.*
- `template<typename T >`  
`void deserialize_field (const INTROSPECTION_CPP_MEMBER *member, void *field, cbor::RxStream &stream)`  
*Deserialize a C++ field.*
- `template<typename T >`  
`void deserialize_field (const INTROSPECTION_C_MEMBER *member, void *field, cbor::RxStream &stream)`  
*Deserialize a C field.*
- `template<typename MembersType >`  
`void deserialize (void *msg, const MembersType *casted_members, cbor::RxStream &stream)`  
*Deserialize a ROS message, request or response.*

### 5.3.1 Detailed Description

Namespace containing serialization functions.

The message data structure coming from upper layers is interpreted using type support informations passed by ROS2 during the creation of publishers, subscribers, clients and services. Those functions are used to compute the exact position that every data type must assume in memory and then calls TxStream or RxStream to receive or write them in the assigned location.

### 5.3.2 Function Documentation

#### 5.3.2.1 deserialize()

```
template<typename MembersType >
void MessageSerialization::deserialize (
    void * msg,
    const MembersType * casted_members,
    cbor::RxStream & stream )
```

Deserialize a ROS message, request or response.

Every time DESERT receives data from the channel a memory location is used to store the corresponding member type, and this function merges all the elementary C or C++ types into the whole message. To perform this operation the `deserialize_field` function is called to decode every specific data.

##### Parameters

<i>msg</i>	Pointer to the first byte of the message in memory
<i>casted_members</i>	Pointer to the member containing type support informations
<i>stream</i>	The stream used to receive data

#### 5.3.2.2 deserialize\_field() [1/2]

```
template<typename T >
void MessageSerialization::deserialize_field (
    const INTROSPECTION_C_MEMBER * member,
    void * field,
    cbor::RxStream & stream )
```

Deserialize a C field.

The type support introspection information is used to know if a specific data type is a single item, a sequence or a variable length sequence. Based on this conclusion a specific interpretation is passed to the stream.

##### Parameters

<i>member</i>	Pointer to the member containing type support informations
<i>field</i>	Pointer to the destination memory address of the elementary data
<i>stream</i>	The stream used to receive data

### 5.3.2.3 deserialize\_field() [2/2]

```
template<typename T >
void MessageSerialization::deserialize_field (
    const INTROSPECTION_CPP_MEMBER * member,
    void * field,
    cbor::RxStream & stream )
```

Deserialize a C++ field.

The type support introspection information is used to know if a specific data type is a single item, a sequence or a vector. Based on this conclusion a specific interpretation is passed to the stream.

#### Parameters

<i>member</i>	Pointer to the member containing type support informations
<i>field</i>	Pointer to the destination memory address of the elementary data
<i>stream</i>	The stream used to receive data

### 5.3.2.4 serialize()

```
template<typename MembersType >
void MessageSerialization::serialize (
    const void * msg,
    const MembersType * casted_members,
    cbor::TxStream & stream )
```

Serialize a ROS message, request or response.

Every time ROS has data to send in the channel a memory location is passed with the corresponding message member type, and this function separates all the fields into elementary C or C++ types. Then the `serialize_field` function is called to encode the specific data.

#### Parameters

<i>msg</i>	Pointer to the first byte of the message in memory
<i>casted_members</i>	Pointer to the member containing type support informations
<i>stream</i>	The stream used to send data

### 5.3.2.5 serialize\_field() [1/2]

```
template<typename T >
void MessageSerialization::serialize_field (
    const INTROSPECTION_C_MEMBER * member,
    void * field,
    cbor::TxStream & stream )
```

Serialize a C field.

The type support introspection information is used to know if a specific data type is a single item, a sequence or a variable length sequence. Based on this conclusion a specific interpretation is passed to the stream.

## Parameters

<i>member</i>	Pointer to the member containing type support informations
<i>field</i>	Pointer to the origin memory address of the elementary data
<i>stream</i>	The stream used to send data

5.3.2.6 `serialize_field()` [2/2]

```
template<typename T >
void MessageSerialization::serialize_field (
    const INTROSPECTION_CPP_MEMBER * member,
    void * field,
    cbor::TxStream & stream )
```

Serialize a C++ field.

The type support introspection information is used to know if a specific data type is a single item, a sequence or a vector. Based on this conclusion a specific interpretation is passed to the stream.

## Parameters

<i>member</i>	Pointer to the member containing type support informations
<i>field</i>	Pointer to the origin memory address of the elementary data
<i>stream</i>	The stream used to send data



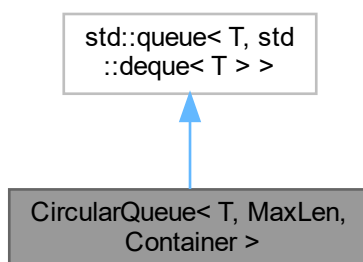


## Chapter 6

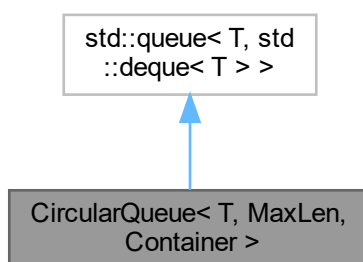
# Class Documentation

### 6.1 CircularQueue< T, MaxLen, Container > Class Template Reference

Inheritance diagram for CircularQueue< T, MaxLen, Container >:



Collaboration diagram for CircularQueue< T, MaxLen, Container >:



## Public Member Functions

- void **push** (const T &value)

The documentation for this class was generated from the following file:

- src/desert\_classes/[CBorStream.h](#)

## 6.2 DesertClient Class Reference

### Public Member Functions

- [DesertClient](#) (std::string service\_name, const rosidl\_service\_type\_support\_t \*type\_supports, rmw\_gid\_t gid)  
*Create a client.*
- bool [has\\_data](#) ()  
*Check if there is available data for the current client instance.*
- void [send\\_request](#) (const void \*req, int64\_t \*sequence\_id)  
*Send a request to the service.*
- void [read\\_response](#) (void \*res, rmw\_service\_info\_t \*req\_header)  
*Read a response from the service.*
- rmw\_gid\_t [get\\_gid](#) ()  
*Retrieve the gid of the current entity.*
- std::string [get\\_service\\_name](#) ()  
*Retrieve the service name of the current entity.*
- std::string [get\\_request\\_type\\_name](#) ()  
*Retrieve the request type of the current entity.*
- std::string [get\\_response\\_type\\_name](#) ()  
*Retrieve the response type of the current entity.*

### 6.2.1 Constructor & Destructor Documentation

#### 6.2.1.1 DesertClient()

```
DesertClient::DesertClient (
    std::string service_name,
    const rosidl_service_type_support_t * type_supports,
    rmw_gid_t gid )
```

Create a client.

#### Parameters

<i>service_name</i>	Name of the service to send requests and receive responses
<i>type_supports</i>	Pointer to the message data structure coming from the ROS upper layers
<i>gid</i>	Global identifier of the entity

## 6.2.2 Member Function Documentation

### 6.2.2.1 get\_gid()

```
rmw_gid_t DesertClient::get_gid ( )
```

Retrieve the gid of the current entity.

This function returns the global identifier of the current entity in the rmw format.

#### Returns

Global identifier of the entity

### 6.2.2.2 get\_request\_type\_name()

```
std::string DesertClient::get_request_type_name ( )
```

Retrieve the request type of the current entity.

This function returns a string containing the service request type name of the current entity.

#### Returns

Type of the service request

### 6.2.2.3 get\_response\_type\_name()

```
std::string DesertClient::get_response_type_name ( )
```

Retrieve the response type of the current entity.

This function returns a string containing the service response type name of the current entity.

#### Returns

Type of the service response

### 6.2.2.4 get\_service\_name()

```
std::string DesertClient::get_service_name ( )
```

Retrieve the service name of the current entity.

This function returns a string containing the service name of the current entity.

#### Returns

Name of the service

### 6.2.2.5 has\_data()

```
bool DesertClient::has_data ( )
```

Check if there is available data for the current client instance.

The `has_data` function calls the `interpret_packets` method in `RxStream` and then verifies if in the map of client packets there is a correspondence with the service name and the sequence identifier of the current instance.

#### Returns

True if data is present otherwise false

### 6.2.2.6 read\_response()

```
void DesertClient::read_response (
    void * res,
    rmw_service_info_t * req_header )
```

Read a response from the service.

The `read_response` function interprets a transmission with the current sequence identifier deserializing the message using the method from the [MessageSerialization](#) namespace. A discrimination is made between C members and C++ members based on the type support.

#### Parameters

<i>res</i>	Pointer to the memory location used to store the reading
<i>req_header</i>	Pointer to the request header used to store the service sequence identifier

### 6.2.2.7 send\_request()

```
void DesertClient::send_request (
    const void * req,
    int64_t * sequence_id )
```

Send a request to the service.

The `send_request` function starts a transmission with the current sequence identifier and then serializes the message using the method from the [MessageSerialization](#) namespace. A discrimination is made between C members and C++ members based on the type support.

#### Parameters

<i>req</i>	Pointer to the request to send
<i>sequence_id</i>	Pointer to the random service sequence identifier

The documentation for this class was generated from the following files:

- [src/desert\\_classes/DesertClient.h](#)
- [src/desert\\_classes/DesertClient.cpp](#)

## 6.3 DesertGuardCondition Class Reference

### Public Member Functions

- **DesertGuardCondition ()**  
*Create a guard condition.*
- void [trigger \(\)](#)  
*Trigger the guard condition.*
- bool [has\\_triggered \(\)](#)  
*Check if the guard condition has triggered.*
- bool [get\\_has\\_triggered \(\)](#)  
*Check if the guard condition has triggered.*

### 6.3.1 Member Function Documentation

#### 6.3.1.1 get\_has\_triggered()

```
bool DesertGuardCondition::get_has_triggered ( )
```

Check if the guard condition has triggered.

The `get_has_triggered` function returns a bool value with the status of the guard condition and resets the internal state to false.

#### Returns

True if the guard condition has triggered otherwise false

#### 6.3.1.2 has\_triggered()

```
bool DesertGuardCondition::has_triggered ( )
```

Check if the guard condition has triggered.

The `has_triggered` function returns a bool value with the status of the guard condition. Its internal state is not modified.

#### Returns

True if the guard condition has triggered otherwise false

### 6.3.1.3 trigger()

```
void DesertGuardCondition::trigger ( )
```

Trigger the guard condition.

The trigger function sets the status of the atomic bool variable `_has_triggered` to save the new status of the guard condition.

The documentation for this class was generated from the following files:

- [src/desert\\_classes/DesertGuardCondition.h](#)
- [src/desert\\_classes/DesertGuardCondition.cpp](#)

## 6.4 DesertNode Class Reference

### Public Member Functions

- [DesertNode](#) (std::string name, std::string namespace\_, rmw\_gid\_t gid)  
*Create a node.*
- void [add\\_publisher](#) ([DesertPublisher](#) \*pub)  
*Add a publisher to the current node.*
- void [add\\_subscriber](#) ([DesertSubscriber](#) \*sub)  
*Add a subscriber to the current node.*
- void [add\\_client](#) ([DesertClient](#) \*cli)  
*Add a client to the current node.*
- void [add\\_service](#) ([DesertService](#) \*ser)  
*Add a service to the current node.*
- void [remove\\_publisher](#) ([DesertPublisher](#) \*pub)  
*Remove a publisher from the current node.*
- void [remove\\_subscriber](#) ([DesertSubscriber](#) \*sub)  
*Remove a subscriber from the current node.*
- void [remove\\_client](#) ([DesertClient](#) \*cli)  
*Remove a client from the current node.*
- void [remove\\_service](#) ([DesertService](#) \*ser)  
*Remove a service from the current node.*
- rmw\_gid\_t [get\\_gid](#) ()  
*Retreive the gid of the current entity.*

### 6.4.1 Constructor & Destructor Documentation

#### 6.4.1.1 DesertNode()

```
DesertNode::DesertNode (
    std::string name,
    std::string namespace_,
    rmw_gid_t gid )
```

Create a node.

## Parameters

<i>name</i>	Name of the node
<i>namespace</i> ↔	Namespace of the node
—	
<i>gid</i>	Global identifier of the node

## 6.4.2 Member Function Documentation

### 6.4.2.1 add\_client()

```
void DesertNode::add_client (
    DesertClient * cli )
```

Add a client to the current node.

This function pushes the pointer to a client in a vector of all the registered clients related to the current node.

## Parameters

<i>cli</i>	Pointer to a <a href="#">DesertClient</a> instance
------------	--

### 6.4.2.2 add\_publisher()

```
void DesertNode::add_publisher (
    DesertPublisher * pub )
```

Add a publisher to the current node.

This function pushes the pointer to a publisher in a vector of all the registered publishers related to the current node.

## Parameters

<i>pub</i>	Pointer to a <a href="#">DesertPublisher</a> instance
------------	---

### 6.4.2.3 add\_service()

```
void DesertNode::add_service (
    DesertService * ser )
```

Add a service to the current node.

This function pushes the pointer to a service in a vector of all the registered services related to the current node.

## Parameters

<i>ser</i>	Pointer to a <a href="#">DesertService</a> instance
------------	---

#### 6.4.2.4 add\_subscriber()

```
void DesertNode::add_subscriber (
    DesertSubscriber * sub )
```

Add a subscriber to the current node.

This function pushes the pointer to a subscriber in a vector of all the registered subscribers related to the current node.

##### Parameters

<i>sub</i>	Pointer to a <a href="#">DesertSubscriber</a> instance
------------	--

#### 6.4.2.5 get\_gid()

```
rmw_gid_t DesertNode::get_gid ( )
```

Retreive the gid of the current entity.

This function returns the global identifier of the current entity in the rmw format.

##### Returns

Global identifier of the entity

#### 6.4.2.6 remove\_client()

```
void DesertNode::remove_client (
    DesertClient * cli )
```

Remove a client from the current node.

This function removes the pointer to a client from the vector of all the registered clients related to the current node.

##### Parameters

<i>cli</i>	Pointer to a <a href="#">DesertClient</a> instance
------------	--

#### 6.4.2.7 remove\_publisher()

```
void DesertNode::remove_publisher (
    DesertPublisher * pub )
```

Remove a publisher from the current node.

This function removes the pointer to a publisher from the vector of all the registered publishers related to the current node.



## Parameters

<i>pub</i>	Pointer to a <a href="#">DesertPublisher</a> instance
------------	---

**6.4.2.8 remove\_service()**

```
void DesertNode::remove_service (
    DesertService * ser )
```

Remove a service from the current node.

This function removes the pointer to a service from the vector of all the registered services related to the current node.

## Parameters

<i>ser</i>	Pointer to a <a href="#">DesertService</a> instance
------------	---

**6.4.2.9 remove\_subscriber()**

```
void DesertNode::remove_subscriber (
    DesertSubscriber * sub )
```

Remove a subscriber from the current node.

This function removes the pointer to a subscriber from the vector of all the registered subscribers related to the current node.

## Parameters

<i>sub</i>	Pointer to a <a href="#">DesertSubscriber</a> instance
------------	--

The documentation for this class was generated from the following files:

- [src/desert\\_classes/DesertNode.h](#)
- [src/desert\\_classes/DesertNode.cpp](#)

**6.5 DesertPublisher Class Reference****Public Member Functions**

- [DesertPublisher](#) (std::string topic\_name, const rosidl\_message\_type\_support\_t \*type\_supports, rmw\_gid\_t gid)  
*Create a publisher.*
- void [push](#) (const void \*msg)  
*Send a publication on the topic.*
- rmw\_gid\_t [get\\_gid](#) ()

*Retreive the gid of the current entity.*

- `std::string get_topic_name ()`

*Retreive the topic name of the current entity.*

- `std::string get_type_name ()`

*Retreive the message type of the current entity.*

## 6.5.1 Constructor & Destructor Documentation

### 6.5.1.1 DesertPublisher()

```
DesertPublisher::DesertPublisher (
    std::string topic_name,
    const rosidl_message_type_support_t * type_supports,
    rmw_gid_t gid )
```

Create a publisher.

#### Parameters

<i>topic_name</i>	Name of the topic used to push the messages
<i>type_supports</i>	Pointer to the message data structure coming from the ROS upper layers
<i>gid</i>	Global identifier of the entity

## 6.5.2 Member Function Documentation

### 6.5.2.1 get\_gid()

```
rmw_gid_t DesertPublisher::get_gid ( )
```

Retreive the gid of the current entity.

This function returns the global identifier of the current entity in the rmw format.

#### Returns

Global identifier of the entity

### 6.5.2.2 get\_topic\_name()

```
std::string DesertPublisher::get_topic_name ( )
```

Retreive the topic name of the current entity.

This function returns a string containing the topic name of the current entity.

#### Returns

Name of the topic

### 6.5.2.3 get\_type\_name()

```
std::string DesertPublisher::get_type_name ( )
```

Retreive the message type of the current entity.

This function returns a string containing the message type name of the current entity.

#### Returns

Type of the message

### 6.5.2.4 push()

```
void DesertPublisher::push (
    const void * msg )
```

Send a publication on the topic.

The push function starts a transmission with the topic name in the current instance and then serializes the message using the method from the [MessageSerialization](#) namespace. A discrimination is made between C members and C++ members based on the type support.

#### Parameters

<i>msg</i>	Pointer to the message to send
------------	--------------------------------

The documentation for this class was generated from the following files:

- src/desert\_classes/[DesertPublisher.h](#)
- src/desert\_classes/DesertPublisher.cpp

## 6.6 DesertService Class Reference

### Public Member Functions

- [DesertService](#) (std::string service\_name, const rosidl\_service\_type\_support\_t \*type\_supports, rmw\_gid\_t gid)  
*Create a service.*
- bool [has\\_data](#) ()  
*Check if there is available data for the current service instance.*
- void [read\\_request](#) (void \*req, rmw\_service\_info\_t \*req\_header)  
*Read a request from a client.*
- void [send\\_response](#) (void \*res, rmw\_request\_id\_t \*req\_header)  
*Send the response to a client.*
- rmw\_gid\_t [get\\_gid](#) ()  
*Retreive the gid of the current entity.*
- std::string [get\\_service\\_name](#) ()

- Retreive the service name of the current entity.
- `std::string get_request_type_name ()`  
Retreive the request type of the current entity.
- `std::string get_response_type_name ()`  
Retreive the response type of the current entity.

## 6.6.1 Constructor & Destructor Documentation

### 6.6.1.1 DesertService()

```
DesertService::DesertService (
    std::string service_name,
    const rosidl_service_type_support_t * type_supports,
    rmw_gid_t gid )
```

Create a service.

#### Parameters

<i>service_name</i>	Name of the service to receive requests and send responses
<i>type_supports</i>	Pointer to the message data structure coming from the ROS upper layers
<i>gid</i>	Global identifier of the entity

## 6.6.2 Member Function Documentation

### 6.6.2.1 get\_gid()

```
rmw_gid_t DesertService::get_gid ( )
```

Retreive the gid of the current entity.

This function returns the global identifier of the current entity in the rmw format.

#### Returns

Global identifier of the entity

### 6.6.2.2 get\_request\_type\_name()

```
std::string DesertService::get_request_type_name ( )
```

Retreive the request type of the current entity.

This function returns a string containing the service request type name of the current entity.

#### Returns

Type of the service request

### 6.6.2.3 get\_response\_type\_name()

```
std::string DesertService::get_response_type_name ( )
```

Retreive the response type of the current entity.

This function returns a string containing the service response type name of the current entity.

#### Returns

Type of the service response

### 6.6.2.4 get\_service\_name()

```
std::string DesertService::get_service_name ( )
```

Retreive the service name of the current entity.

This function returns a string containing the service name of the current entity.

#### Returns

Name of the service

### 6.6.2.5 has\_data()

```
bool DesertService::has_data ( )
```

Check if there is available data for the current service instance.

The has\_data function calls the interpret\_packets method in RxStream and then verifies if in the map of service packets there is a correspondence with the service name of the current instance.

#### Returns

True if data is present otherwise false

### 6.6.2.6 read\_request()

```
void DesertService::read_request (
    void * req,
    rmw_service_info_t * req_header )
```

Read a request from a client.

The read\_request function interprets a transmission with the service name in the current instance deserializing the message using the method from the [MessageSerialization](#) namespace. A discrimination is made between C members and C++ members based on the type support.

## Parameters

<i>req</i>	Pointer to the memory location used to store the request
<i>req_header</i>	Pointer to the request header used to store the service sequence identifier

**6.6.2.7 send\_response()**

```
void DesertService::send_response (
    void * res,
    rmw_request_id_t * req_header )
```

Send the response to a client.

The `send_response` function starts a transmission with the sequence identifier in `req_header` and then serializes the message using the method from the [MessageSerialization](#) namespace. A discrimination is made between C members and C++ members based on the type support.

## Parameters

<i>res</i>	Pointer to the response to send
<i>req_header</i>	Pointer to the request header used to store the service sequence identifier

The documentation for this class was generated from the following files:

- `src/desert_classes/DesertService.h`
- `src/desert_classes/DesertService.cpp`

**6.7 DesertSubscriber Class Reference****Public Member Functions**

- [DesertSubscriber](#) (std::string topic\_name, const rosidl\_message\_type\_support\_t \*type\_supports, rmw\_gid\_t gid)  
*Create a subscriber.*
- bool [has\\_data](#) ()  
*Check if there is available data for the registered topic.*
- void [read\\_data](#) (void \*msg)  
*Read a publication from the publisher.*
- rmw\_gid\_t [get\\_gid](#) ()  
*Retrieve the gid of the current entity.*
- std::string [get\\_topic\\_name](#) ()  
*Retrieve the topic name of the current entity.*
- std::string [get\\_type\\_name](#) ()  
*Retrieve the message type of the current entity.*

## 6.7.1 Constructor & Destructor Documentation

### 6.7.1.1 DesertSubscriber()

```
DesertSubscriber::DesertSubscriber (
    std::string topic_name,
    const rosidl_message_type_support_t * type_supports,
    rmw_gid_t gid )
```

Create a subscriber.

#### Parameters

<i>topic_name</i>	Name of the topic used for the registration
<i>type_supports</i>	Pointer to the message data structure coming from the ROS upper layers
<i>gid</i>	Global identifier of the entity

## 6.7.2 Member Function Documentation

### 6.7.2.1 get\_gid()

```
rmw_gid_t DesertSubscriber::get_gid ( )
```

Retrieve the gid of the current entity.

This function returns the global identifier of the current entity in the rmw format.

#### Returns

Global identifier of the entity

### 6.7.2.2 get\_topic\_name()

```
std::string DesertSubscriber::get_topic_name ( )
```

Retrieve the topic name of the current entity.

This function returns a string containing the topic name of the current entity.

#### Returns

Name of the topic

### 6.7.2.3 get\_type\_name()

```
std::string DesertSubscriber::get_type_name ( )
```

Retrieve the message type of the current entity.

This function returns a string containing the message type name of the current entity.

#### Returns

Type of the message

#### 6.7.2.4 has\_data()

```
bool DesertSubscriber::has_data ( )
```

Check if there is available data for the registered topic.

The has\_data function calls the interpret\_packets method in RxStream and then verifies if in the map of subscriber packets there is a correspondence with the topic name of the current instance.

##### Returns

True if data is present otherwise false

#### 6.7.2.5 read\_data()

```
void DesertSubscriber::read_data (
    void * msg )
```

Read a publication from the publisher.

The read\_data function interprets a transmission with the topic name present in the current instance deserializing the message using the method from the [MessageSerialization](#) namespace. A discrimination is made between C members and C++ members based on the type support.

##### Parameters

<i>msg</i>	Pointer to the memory location used to store the message
------------	--

The documentation for this class was generated from the following files:

- src/desert\_classes/[DesertSubscriber.h](#)
- src/desert\_classes/DesertSubscriber.cpp

## 6.8 DesertWaitset Class Reference

### Public Attributes

- std::mutex **lock**
- bool **inuse**

The documentation for this class was generated from the following file:

- src/desert\_classes/[DesertWaitSet.h](#)

## 6.9 GenericCSequence< T > Struct Template Reference

The documentation for this struct was generated from the following file:

- src/desert\_classes/[macros.h](#)



## 6.10 rmw\_context\_impl\_s Struct Reference

### Public Attributes

- rmw\_dds\_common::Context **common**
- bool **is\_shutdown** {false}

The documentation for this struct was generated from the following file:

- [src/desert\\_classes/rmw\\_context\\_impl\\_s.h](#)

## 6.11 cbor::RxStream Class Reference

### Public Member Functions

- [RxStream](#) (uint8\_t stream\_type, std::string stream\_name, uint8\_t stream\_identifier)  
*Create a reception stream.*
- [~RxStream](#) ()  
*Destroy the reception stream.*
- bool [data\\_available](#) (int64\_t sequence\_id=0)  
*Check if there are data.*
- void [clear\\_buffer](#) ()  
*Clear the currently buffered packet.*
- [RxStream & operator>>](#) (uint64\_t &n)  
*Decode uint64.*
- [RxStream & operator>>](#) (uint32\_t &n)  
*Decode uint32.*
- [RxStream & operator>>](#) (uint16\_t &n)  
*Decode uint16.*
- [RxStream & operator>>](#) (uint8\_t &n)  
*Decode uint8.*
- [RxStream & operator>>](#) (int64\_t &n)  
*Decode int64.*
- [RxStream & operator>>](#) (int32\_t &n)  
*Decode int32.*
- [RxStream & operator>>](#) (int16\_t &n)  
*Decode int16.*
- [RxStream & operator>>](#) (int8\_t &n)  
*Decode int8.*
- template<typename T >  
[RxStream & deserialize\\_integer](#) (T &n)  
*Decode a generic integer.*
- [RxStream & operator>>](#) (char &n)  
*Decode char.*
- [RxStream & operator>>](#) (float &f)  
*Decode float.*
- [RxStream & operator>>](#) (double &d)  
*Decode double.*
- [RxStream & operator>>](#) (std::string &s)

- Decode string.*
- [RxStream & operator>>](#) (std::u16string &s)
- Decode u16string.*
- [RxStream & operator>>](#) (bool &b)
- Decode bool.*
- template<typename T >  
[RxStream & operator>>](#) (std::vector< T > &v)
- Decode vector.*
- [RxStream & operator>>](#) (std::vector< bool > &v)
- Decode bool vector.*
- template<typename T >  
[RxStream & deserialize\\_sequence](#) (T \*items, size\_t size)
- Deserialize a sequence of uniform elements.*
- uint8\_t [get\\_type](#) () const
- Get the stream type of a specific instance.*
- std::string [get\\_name](#) () const
- Get the topic name of a specific instance.*
- uint8\_t [get\\_identifier](#) () const
- Get the stream identifier of a specific instance.*
- void [push\\_packet](#) (std::vector< std::pair< void \*, int > > packet)
- Add a packet to \_received\_packets.*

## Static Public Member Functions

- static void [interpret\\_packets](#) ()
- Interpret raw packets and splits them into different communication types.*

## 6.11.1 Constructor & Destructor Documentation

### 6.11.1.1 RxStream()

```
cbor::RxStream::RxStream (
    uint8_t stream_type,
    std::string stream_name,
    uint8_t stream_identifier )
```

Create a reception stream.

#### Parameters

<i>stream_type</i>	Type of the object using the current instance
<i>stream_name</i>	Name of the topic or the service to which the communication belongs
<i>stream_identifier</i>	Identifier of the topic or the service read from configuration

## 6.11.2 Member Function Documentation

### 6.11.2.1 clear\_buffer()

```
void cbor::RxStream::clear_buffer ( )
```

Clear the currently buffered packet.

When the packet is read by the entity, this function must be called to clear the buffer and allow [RxStream](#) to add the next one in the queue.

### 6.11.2.2 data\_available()

```
bool cbor::RxStream::data_available (
    int64_t sequence_id = 0 )
```

Check if there are data.

A map contains the information received for all topics and services, so using the name saved in the current instance as key it is possible to know if a message is arrived for a specific entity.

#### Parameters

<i>sequence_id</i>	The id of the client service communication
--------------------	--

### 6.11.2.3 deserialize\_integer()

```
template<typename T >
RxStream & cbor::RxStream::deserialize_integer (
    T & n )
```

Decode a generic integer.

#### Parameters

<i>n</i>	Field to decode
----------	-----------------

### 6.11.2.4 deserialize\_sequence()

```
template<typename T >
RxStream & cbor::RxStream::deserialize_sequence (
    T * items,
    size_t size ) [inline]
```

Deserialize a sequence of uniform elements.

#### Parameters

<i>items</i>	Pointer to the first element
<i>size</i>	Size of the items array

#### 6.11.2.5 get\_identifier()

```
uint8_t cbor::RxStream::get_identifier ( ) const
```

Get the stream identifier of a specific instance.

##### Returns

Topic identifier of the stream

#### 6.11.2.6 get\_name()

```
std::string cbor::RxStream::get_name ( ) const
```

Get the topic name of a specific instance.

##### Returns

Topic name of the stream

#### 6.11.2.7 get\_type()

```
uint8_t cbor::RxStream::get_type ( ) const
```

Get the stream type of a specific instance.

##### Returns

Type of the stream

#### 6.11.2.8 interpret\_packets()

```
void cbor::RxStream::interpret_packets ( ) [static]
```

Interpret raw packets and splits them into different communication types.

Raw packets from [TcpDaemon](#) are read and interpreted in order to put them in a map where the key allows to distinguish the topic name or the service name, and eventually the sequence identifier.

#### 6.11.2.9 operator>>() [1/16]

```
RxStream & cbor::RxStream::operator>> (
    bool & b )
```

Decode bool.

## Parameters

<i>b</i>	Field to decode
----------	-----------------

**6.11.2.10 operator>>() [2/16]**

```
RxStream & cbor::RxStream::operator>> (
    char & n )
```

Decode char.

## Parameters

<i>n</i>	Field to decode
----------	-----------------

**6.11.2.11 operator>>() [3/16]**

```
RxStream & cbor::RxStream::operator>> (
    double & d )
```

Decode double.

## Parameters

<i>d</i>	Field to decode
----------	-----------------

**6.11.2.12 operator>>() [4/16]**

```
RxStream & cbor::RxStream::operator>> (
    float & f )
```

Decode float.

## Parameters

<i>f</i>	Field to decode
----------	-----------------

**6.11.2.13 operator>>() [5/16]**

```
RxStream & cbor::RxStream::operator>> (
    int16_t & n )
```

Decode int16.

**Parameters**

<i>n</i>	Field to decode
----------	-----------------

**6.11.2.14 operator>>() [6/16]**

```
RxStream & cbor::RxStream::operator>> (  
    int32_t & n )
```

Decode int32.

**Parameters**

<i>n</i>	Field to decode
----------	-----------------

**6.11.2.15 operator>>() [7/16]**

```
RxStream & cbor::RxStream::operator>> (  
    int64_t & n )
```

Decode int64.

**Parameters**

<i>n</i>	Field to decode
----------	-----------------

**6.11.2.16 operator>>() [8/16]**

```
RxStream & cbor::RxStream::operator>> (  
    int8_t & n )
```

Decode int8.

**Parameters**

<i>n</i>	Field to decode
----------	-----------------

**6.11.2.17 operator>>() [9/16]**

```
RxStream & cbor::RxStream::operator>> (  
    std::string & s )
```

Decode string.

## Parameters

<i>s</i>	Field to decode
----------	-----------------

**6.11.2.18 operator>>()** [10/16]

```
RxStream & cbor::RxStream::operator>> (
    std::u16string & s )
```

Decode u16string.

## Parameters

<i>s</i>	Field to decode
----------	-----------------

**6.11.2.19 operator>>()** [11/16]

```
RxStream & cbor::RxStream::operator>> (
    std::vector< bool > & v )
```

Decode bool vector.

## Parameters

<i>v</i>	Field to decode
----------	-----------------

**6.11.2.20 operator>>()** [12/16]

```
template<typename T >
RxStream & cbor::RxStream::operator>> (
    std::vector< T > & v ) [inline]
```

Decode vector.

## Parameters

<i>v</i>	Field to decode
----------	-----------------

**6.11.2.21 operator>>()** [13/16]

```
RxStream & cbor::RxStream::operator>> (
    uint16_t & n )
```

Decode uint16.

## Parameters

<i>n</i>	Field to decode
----------	-----------------

**6.11.2.22 operator>>() [14/16]**

```
RxStream & cbor::RxStream::operator>> (
    uint32_t & n )
```

Decode uint32.

## Parameters

<i>n</i>	Field to decode
----------	-----------------

**6.11.2.23 operator>>() [15/16]**

```
RxStream & cbor::RxStream::operator>> (
    uint64_t & n )
```

Decode uint64.

## Parameters

<i>n</i>	Field to decode
----------	-----------------

**6.11.2.24 operator>>() [16/16]**

```
RxStream & cbor::RxStream::operator>> (
    uint8_t & n )
```

Decode uint8.

## Parameters

<i>n</i>	Field to decode
----------	-----------------

**6.11.2.25 push\_packet()**

```
void cbor::RxStream::push_packet (
    std::vector< std::pair< void *, int > > packet )
```

Add a packet to `_received_packets`.

When `interpret_packets()` is called is read all the currently active `RxStream` instances, and uses this function to enqueue a packet if the stream matches the type and the topic.



## Parameters

<i>packet</i>	The packet to add
---------------	-------------------

The documentation for this class was generated from the following files:

- [src/desert\\_classes/CBorStream.h](#)
- [src/desert\\_classes/CBorStream.cpp](#)

## 6.12 TcpDaemon Class Reference

### Public Member Functions

- [bool](#) [init](#) (int port)  
*Initialize the socket communication.*

### Static Public Member Functions

- static [std::vector< uint8\\_t >](#) [read\\_packet](#) ()  
*Read a packet from the `_rx_packets` member as vector of bytes.*
- static void [enqueue\\_packet](#) ([std::vector< uint8\\_t >](#) packet)  
*Enqueue a packet in the `_tx_packets` member as vector of bytes.*

### 6.12.1 Member Function Documentation

#### 6.12.1.1 enqueue\_packet()

```
void TcpDaemon::enqueue_packet (
    std::vector< uint8_t > packet ) [static]
```

Enqueue a packet in the `_tx_packets` member as vector of bytes.

This function is used by the various TxStream instances contained in publishers, clients and services.

## Parameters

<i>packet</i>	The packet that has to be sent through the DESERT stack
---------------	---

#### 6.12.1.2 init()

```
bool TcpDaemon::init (
    int port )
```

Initialize the socket communication.

This function allows the middleware to establish a connection to the DESERT stack through a TCP socket.

**Parameters**

<i>port</i>	The TCP port of the DESERT application layer
-------------	--

**6.12.1.3 read\_packet()**

```
std::vector< uint8_t > TcpDaemon::read_packet ( ) [static]
```

Read a packet from the `_rx_packets` member as vector of bytes.

This function is used by the various RxStream instances contained in subscribers, clients and services.

**Returns**

The packet that was read from the DESERT stack

The documentation for this class was generated from the following files:

- [src/desert\\_classes/TcpDaemon.h](#)
- [src/desert\\_classes/TcpDaemon.cpp](#)

**6.13 TopicsConfig Class Reference****Static Public Member Functions**

- static void [load\\_configuration](#) ()  
*Initialize the configuration.*
- static uint8\_t [get\\_topic\\_identifier](#) (std::string name)  
*Get topic's identifier from configuration.*
- static std::string [get\\_identifier\\_topic](#) (uint8\_t identifier)  
*Get identifier's topic from configuration.*

**6.13.1 Member Function Documentation****6.13.1.1 get\_identifier\_topic()**

```
std::string TopicsConfig::get_identifier_topic (
    uint8_t identifier ) [static]
```

Get identifier's topic from configuration.

This function returns the topic associated to an identifier if it exists, otherwise returns an empty string.

**Returns**

The identifier topic

### 6.13.1.2 get\_topic\_identifier()

```
uint8_t TopicsConfig::get_topic_identifier (
    std::string name ) [static]
```

Get topic's identifier from configuration.

This function returns the identifier associated to a topic if it exists, otherwise returns zero.

#### Returns

The topic identifier

### 6.13.1.3 load\_configuration()

```
void TopicsConfig::load_configuration ( ) [static]
```

Initialize the configuration.

This function reads the configuration file from `./ros_allowed_topics.conf`. If not present, a warning will be displayed.

The documentation for this class was generated from the following files:

- `src/desert_classes/TopicsConfig.h`
- `src/desert_classes/TopicsConfig.cpp`

## 6.14 cbor::TxStream Class Reference

### Public Member Functions

- [TxStream](#) (uint8\_t stream\_type, std::string stream\_name, uint8\_t stream\_identifier)  
*Create a transmission stream.*
- void [start\\_transmission](#) (uint64\_t sequence\_id)  
*Tell the stream to create a new packet.*
- void [start\\_transmission](#) ()  
*Tell the stream to create a new packet.*
- void [end\\_transmission](#) ()  
*Tell the stream to send down the packet.*
- [TxStream](#) & [operator<<](#) (const uint64\_t n)  
*Encode uint64.*
- [TxStream](#) & [operator<<](#) (const uint32\_t n)  
*Encode uint32.*
- [TxStream](#) & [operator<<](#) (const uint16\_t n)  
*Encode uint16.*
- [TxStream](#) & [operator<<](#) (const uint8\_t n)  
*Encode uint8.*
- [TxStream](#) & [operator<<](#) (const int64\_t n)  
*Encode int64.*
- [TxStream](#) & [operator<<](#) (const int32\_t n)

- Encode int32.*
- [TxStream](#) & [operator<<](#) (const int16\_t n)
- Encode int16.*
- [TxStream](#) & [operator<<](#) (const int8\_t n)
- Encode int8.*
- [TxStream](#) & [operator<<](#) (const char n)
- Encode char.*
- [TxStream](#) & [operator<<](#) (const float f)
- Encode float.*
- [TxStream](#) & [operator<<](#) (const double d)
- Encode double.*
- [TxStream](#) & [operator<<](#) (const std::string s)
- Encode string.*
- [TxStream](#) & [operator<<](#) (const std::u16string s)
- Encode u16string.*
- [TxStream](#) & [operator<<](#) (const bool b)
- Encode bool.*
- template<typename T >  
[TxStream](#) & [operator<<](#) (const std::vector< T > v)
- Encode vector.*
- [TxStream](#) & [operator<<](#) (const std::vector< bool > v)
- Encode bool vector.*
- template<typename T >  
[TxStream](#) & [serialize\\_sequence](#) (const T \*items, size\_t size)
- Serialize a sequence of uniform elements.*

## 6.14.1 Constructor & Destructor Documentation

### 6.14.1.1 TxStream()

```
cbor::TxStream::TxStream (
    uint8_t stream_type,
    std::string stream_name,
    uint8_t stream_identifier )
```

Create a transmission stream.

#### Parameters

<i>stream_type</i>	Type of the object using the current instance
<i>stream_name</i>	Name of the topic or the service to which the communication belongs
<i>stream_identifier</i>	Identifier of the topic or the service read from configuration

## 6.14.2 Member Function Documentation

### 6.14.2.1 end\_transmission()

```
void cbor::TxStream::end_transmission ( )
```

Tell the stream to send down the packet.

When the transmission is finished the packet is stored in the static member of [TcpDaemon](#) in order to be sent to DESERT.

#### 6.14.2.2 operator<<() [1/16]

```
TxStream & cbor::TxStream::operator<< (  
    const bool b )
```

Encode bool.

Parameters

<i>b</i>	Field to encode
----------	-----------------

#### 6.14.2.3 operator<<() [2/16]

```
TxStream & cbor::TxStream::operator<< (  
    const char n )
```

Encode char.

Parameters

<i>n</i>	Field to encode
----------	-----------------

#### 6.14.2.4 operator<<() [3/16]

```
TxStream & cbor::TxStream::operator<< (  
    const double d )
```

Encode double.

Parameters

<i>d</i>	Field to encode
----------	-----------------

#### 6.14.2.5 operator<<() [4/16]

```
TxStream & cbor::TxStream::operator<< (  
    const float f )
```

Encode float.

Parameters

<i>f</i>	Field to encode
----------	-----------------

#### 6.14.2.6 operator<<() [5/16]

```
TxStream & cbor::TxStream::operator<< (
    const int16_t n )
```

Encode int16.

Parameters

<i>n</i>	Field to encode
----------	-----------------

#### 6.14.2.7 operator<<() [6/16]

```
TxStream & cbor::TxStream::operator<< (
    const int32_t n )
```

Encode int32.

Parameters

<i>n</i>	Field to encode
----------	-----------------

#### 6.14.2.8 operator<<() [7/16]

```
TxStream & cbor::TxStream::operator<< (
    const int64_t n )
```

Encode int64.

Parameters

<i>n</i>	Field to encode
----------	-----------------

#### 6.14.2.9 operator<<() [8/16]

```
TxStream & cbor::TxStream::operator<< (
    const int8_t n )
```

Encode int8.

Parameters

<i>n</i>	Field to encode
----------	-----------------

#### 6.14.2.10 operator<<() [9/16]

```
TxStream & cbor::TxStream::operator<< (
```

```
const std::string s )
```

Encode string.

#### Parameters

<code>s</code>	Field to encode
----------------	-----------------

#### 6.14.2.11 operator<<() [10/16]

```
TxStream & cbor::TxStream::operator<< (
    const std::u16string s )
```

Encode u16string.

#### Parameters

<code>s</code>	Field to encode
----------------	-----------------

#### 6.14.2.12 operator<<() [11/16]

```
TxStream & cbor::TxStream::operator<< (
    const std::vector< bool > v )
```

Encode bool vector.

#### Parameters

<code>v</code>	Field to encode
----------------	-----------------

#### 6.14.2.13 operator<<() [12/16]

```
template<typename T >
TxStream & cbor::TxStream::operator<< (
    const std::vector< T > v ) [inline]
```

Encode vector.

#### Parameters

<code>v</code>	Field to encode
----------------	-----------------

#### 6.14.2.14 operator<<() [13/16]

```
TxStream & cbor::TxStream::operator<< (
    const uint16_t n )
```

Encode uint16.



## Parameters

<i>n</i>	Field to encode
----------	-----------------

**6.14.2.15 operator<<() [14/16]**

```
TxStream & cbor::TxStream::operator<< (  
    const uint32_t n )
```

Encode uint32.

## Parameters

<i>n</i>	Field to encode
----------	-----------------

**6.14.2.16 operator<<() [15/16]**

```
TxStream & cbor::TxStream::operator<< (  
    const uint64_t n )
```

Encode uint64.

## Parameters

<i>n</i>	Field to encode
----------	-----------------

**6.14.2.17 operator<<() [16/16]**

```
TxStream & cbor::TxStream::operator<< (  
    const uint8_t n )
```

Encode uint8.

## Parameters

<i>n</i>	Field to encode
----------	-----------------

**6.14.2.18 serialize\_sequence()**

```
template<typename T >  
TxStream & cbor::TxStream::serialize_sequence (  
    const T * items,  
    size_t size ) [inline]
```

Serialize a sequence of uniform elements.

## Parameters

<i>items</i>	Pointer to the first element
<i>size</i>	Size of the items array

**6.14.2.19 start\_transmission()** [1/2]

```
void cbor::TxStream::start_transmission ( )
```

Tell the stream to create a new packet.

Every time a transmission in started, a new empty packet must be generated and saved as a private member. Then type and topic name are put in front of the data.

**6.14.2.20 start\_transmission()** [2/2]

```
void cbor::TxStream::start_transmission (
    uint64_t sequence_id )
```

Tell the stream to create a new packet.

Every time a transmission in started, a new empty packet must be generated and saved as a private member. Then type, service name and sequence id are put in front of the data.

## Parameters

<i>sequence↵ _id</i>	The id of the client service communication
--------------------------	--

The documentation for this class was generated from the following files:

- src/desert\_classes/[CBorStream.h](#)
- src/desert\_classes/CBorStream.cpp

## Chapter 7

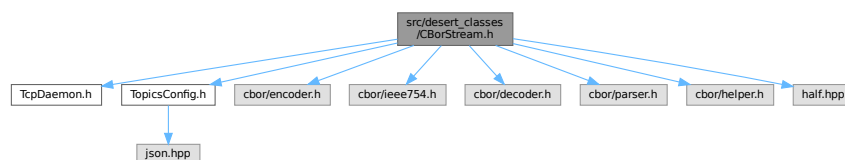
# File Documentation

### 7.1 src/desert\_classes/CBorStream.h File Reference

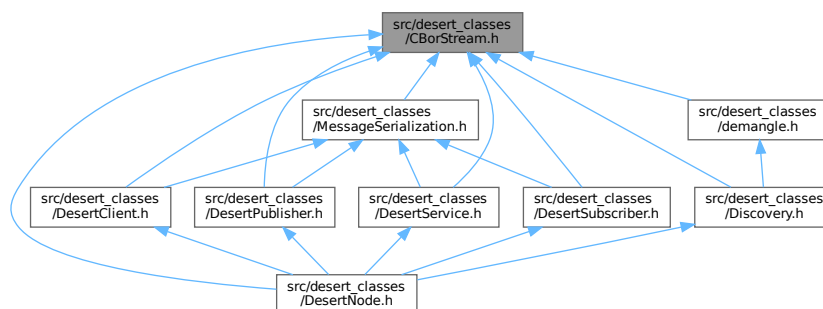
Classes used to convert data types into a CBOR encoded stream.

```
#include "TcpDaemon.h"
#include "TopicsConfig.h"
#include "cbor/encoder.h"
#include "cbor/ieee754.h"
#include "cbor/decoder.h"
#include "cbor/parser.h"
#include "cbor/helper.h"
#include "half.hpp"
```

Include dependency graph for CBorStream.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [CircularQueue< T, MaxLen, Container >](#)
- class [cbor::TxStream](#)
- class [cbor::RxStream](#)

## Macros

- `#define PUBLISHER_TYPE 0`
- `#define SUBSCRIBER_TYPE 1`
- `#define CLIENT_TYPE 2`
- `#define SERVICE_TYPE 3`
- `#define MAX_BUFFER_CAPACITY 100`

### 7.1.1 Detailed Description

Classes used to convert data types into a CBOR encoded stream.

In order to perform a socket communication different data types needs to be encoded into binary representations so they can be sent through the same channel. CBOR fits perfectly with the DESERT requirements because only a minimal overhead is introduced in the stream and all the data types are sent using only the minimal quantity of bytes possible.

#### Author

Prof. Davide Costa

## 7.2 CBorStream.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * Copyright (C) 2024 Davide Costa
00003  *
00004  * This file is part of RMW desert.
00005  *
00006  * RMW desert is free software: you can redistribute it and/or modify it
00007  * under the terms of the GNU General Public License as published by the
00008  * Free Software Foundation, either version 3 of the License, or any
00009  * later version.
00010  *
00011  * RMW desert is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with RMW desert. If not, see <http://www.gnu.org/licenses/>.
00018  *****/
00019
00034 #ifndef CBORSTREAM_H_
00035 #define CBORSTREAM_H_
00036
00037 #include "TcpDaemon.h"
00038 #include "TopicsConfig.h"
00039
00042 #include <map>
00043 #include <queue>
00044 #include <utility>
00045 #include <vector>
00046 #include <string>
00047 #include <locale>
00048 #include <codecvt>
00049 #include <cstdint>

```

```

00050 #include <cstdio>
00051 #include <mutex>
00052
00055 #include "cbor/encoder.h"
00056 #include "cbor/ieee754.h"
00057 #include "cbor/decoder.h"
00058 #include "cbor/parser.h"
00059 #include "cbor/helper.h"
00060
00061 #include "half.hpp"
00062
00063 #define PUBLISHER_TYPE 0
00064 #define SUBSCRIBER_TYPE 1
00065 #define CLIENT_TYPE 2
00066 #define SERVICE_TYPE 3
00067
00068 #define MAX_BUFFER_CAPACITY 100
00069
00070 template <typename T, int MaxLen, typename Container=std::deque<T>
00071 class CircularQueue : public std::queue<T, Container> {
00072 public:
00073     void push(const T& value)
00074     {
00075         if (this->size() == MaxLen)
00076         {
00077             this->c.pop_front();
00078         }
00079         std::queue<T, Container>::push(value);
00080     }
00081 };
00082
00083 namespace cbor
00084 {
00085
00086 class TxStream
00087 {
00088 public:
00089     TxStream(uint8_t stream_type, std::string stream_name, uint8_t stream_identififier);
00090
00091     void start_transmission(uint64_t sequence_id);
00092     void start_transmission();
00093     void end_transmission();
00094
00095     TxStream & operator<<(const uint64_t n);
00096     TxStream & operator<<(const uint32_t n);
00097     TxStream & operator<<(const uint16_t n);
00098     TxStream & operator<<(const uint8_t n);
00099     TxStream & operator<<(const int64_t n);
00100     TxStream & operator<<(const int32_t n);
00101     TxStream & operator<<(const int16_t n);
00102     TxStream & operator<<(const int8_t n);
00103     TxStream & operator<<(const char n);
00104     TxStream & operator<<(const float f);
00105     TxStream & operator<<(const double d);
00106     TxStream & operator<<(const std::string s);
00107     TxStream & operator<<(const std::ul64string s);
00108     TxStream & operator<<(const bool b);
00109
00110     template<typename T>
00111     inline TxStream & operator<<(const std::vector<T> v)
00112     {
00113         *this << static_cast<const uint32_t>(v.size());
00114         return serialize_sequence(v.data(), v.size());
00115     }
00116
00117     TxStream & operator<<(const std::vector<bool> v);
00118
00119     template<typename T>
00120     inline TxStream & serialize_sequence(const T * items, size_t size)
00121     {
00122         for (size_t i = 0; i < size; ++i)
00123         {
00124             *this << items[i];
00125         }
00126         return *this;
00127     }
00128 private:
00129     uint8_t _stream_type;
00130     std::string _stream_name;
00131     uint8_t _stream_identififier;
00132
00133     bool _overflow;
00134     uint8_t * _packet;
00135     cbor_writer_t * _writer;
00136
00137     void new_packet();

```

```

00237     void handle_oveerrun(cbor_error_t result);
00238
00239     std::string toUTF8(const std::ul6string source);
00240
00241 };
00242
00243 class RxStream
00244 {
00245     public:
00253     RxStream(uint8_t stream_type, std::string stream_name, uint8_t stream_identifiler);
00254
00258     ~RxStream();
00259
00269     bool data_available(int64_t sequence_id = 0);
00270
00277     void clear_buffer();
00278
00283     RxStream & operator»(uint64_t & n);
00288     RxStream & operator»(uint32_t & n);
00293     RxStream & operator»(uint16_t & n);
00298     RxStream & operator»(uint8_t & n);
00303     RxStream & operator»(int64_t & n);
00308     RxStream & operator»(int32_t & n);
00313     RxStream & operator»(int16_t & n);
00318     RxStream & operator»(int8_t & n);
00319
00324     template<typename T>
00325     RxStream & deserialize_integer(T & n);
00326
00331     RxStream & operator»(char & n);
00336     RxStream & operator»(float & f);
00341     RxStream & operator»(double & d);
00346     RxStream & operator»(std::string & s);
00351     RxStream & operator»(std::ul6string & s);
00356     RxStream & operator»(bool & b);
00357
00362     template<typename T>
00363     inline RxStream & operator»(std::vector<T> & v)
00364     {
00365         uint32_t size;
00366         *this » size;
00367         v.resize(size);
00368
00369         return deserialize_sequence(v.data(), size);
00370     }
00371
00376     RxStream & operator»(std::vector<bool> & v);
00377
00383     template<typename T>
00384     inline RxStream & deserialize_sequence(T * items, size_t size)
00385     {
00386         for (size_t i = 0; i < size; ++i)
00387         {
00388             *this » items[i];
00389         }
00390         return *this;
00391     }
00392
00398     uint8_t get_type() const;
00403     std::string get_name() const;
00408     uint8_t get_identifiler() const;
00409
00419     void push_packet(std::vector<std::pair<void *, int>» packet);
00420
00428     static void interpret_packets();
00429
00430     private:
00431     uint8_t _stream_type;
00432     std::string _stream_name;
00433     uint8_t _stream_identifiler;
00434
00435     size_t _buffered_iterator;
00436
00437     // packets: <packet <field, field_type>
00438     std::vector<std::pair<void *, int>» _buffered_packet;
00439     CircularQueue<std::vector<std::pair<void *, int>», MAX_BUFFER_CAPACITY> _received_packets;
00440
00441     static const std::map<int, int> _stream_type_match_map;
00442     static std::vector<RxStream *» _listening_streams;
00443
00444     union _cbor_value {
00445         int8_t i8;
00446         int16_t i16;
00447         int32_t i32;
00448         int64_t i64;
00449         float f32;

```

```

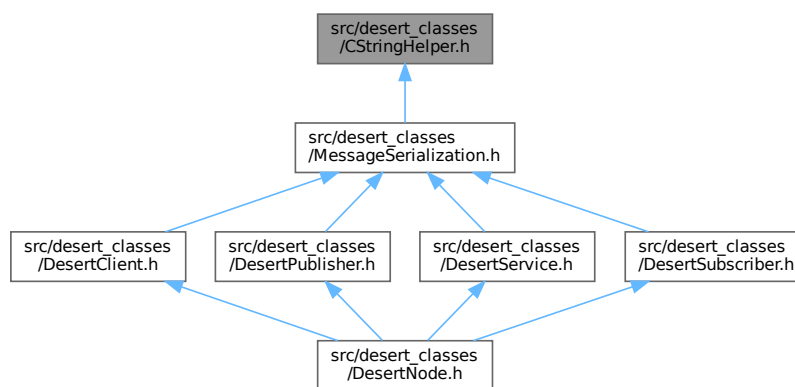
00450     double f64;
00451     uint8_t *bin;
00452     char *str;
00453     uint8_t str_copy[128];
00454 };
00455
00456     static std::mutex _rx_mutex;
00457
00458     static std::pair<void *, int> interpret_field(cbor_item_t * items, size_t i, union _cbor_value &
val);
00459     std::u16string toUTF16(const std::string source);
00460 };
00461
00462 } // namespace cbor
00463
00464
00465 #endif

```

## 7.3 src/desert\_classes/CStringHelper.h File Reference

Namespace containing C sequence handling functions.

This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [CStringHelper](#)  
Namespace containing C sequence handling functions.

### Functions

- std::string [CStringHelper::convert\\_to\\_std\\_string](#) (void \*str)  
Convert a `rosidl_runtime_c_String` into `std::string`.
- std::vector< std::string > [CStringHelper::convert\\_to\\_std\\_vector\\_string](#) (void \*str\_array, size\_t size)  
Convert a `rosidl_runtime_c_String` into a vector of `std::string`.
- std::vector< std::string > [CStringHelper::convert\\_sequence\\_to\\_std\\_vector\\_string](#) (void \*str\_seq)  
Convert a `rosidl_runtime_c_String__Sequence` into a vector of `std::string`.
- std::u16string [CStringHelper::convert\\_to\\_std\\_u16string](#) (void \*str)  
Convert a `rosidl_runtime_c_U16String` into `std::u16string`.

- `std::vector< std::u16string > CStringHelper::convert_to_std_vector_u16string` (void \*str\_array, size\_t size)  
*Convert a `rosidl_runtime_c__U16String` into a vector of `std::u16string`.*
- `std::vector< std::u16string > CStringHelper::convert_sequence_to_std_vector_u16string` (void \*str\_seq)  
*Convert a `rosidl_runtime_c__U16String__Sequence` into a vector of `std::u16string`.*
- void `CStringHelper::assign_string` (std::string str, void \*field)  
*Assigning to a `rosidl_runtime_c__String` the value contained in a `std::string`.*
- void `CStringHelper::assign_vector_string` (std::vector< std::string > cpp\_string\_vector, void \*str\_array, size\_t size)  
*Assigning to a `rosidl_runtime_c__String` the value contained in a vector of `std::string`.*
- void `CStringHelper::assign_vector_string_to_sequence` (std::vector< std::string > cpp\_string\_vector, void \*str\_seq)  
*Assigning to a `rosidl_runtime_c__String__Sequence` the value contained in a vector of `std::string`.*
- void `CStringHelper::assign_u16string` (std::u16string str, void \*field)  
*Assigning to a `rosidl_runtime_c__U16String` the value contained in a `std::u16string`.*
- void `CStringHelper::assign_vector_u16string` (std::vector< std::u16string > cpp\_string\_vector, void \*str\_array, size\_t size)  
*Assigning to a `rosidl_runtime_c__U16String` the value contained in a vector of `std::u16string`.*
- void `CStringHelper::assign_vector_u16string_to_sequence` (std::vector< std::u16string > cpp\_string\_vector, void \*str\_seq)  
*Assigning to a `rosidl_runtime_c__U16String__Sequence` the value contained in a vector of `std::u16string`.*

### 7.3.1 Detailed Description

Namespace containing C sequence handling functions.

The C data type implementation is more complicated than the C++ one, because complex types like vectors have to be manually managed and this header contains functions to convert C strings and generic sequences into respectively C++ strings and vectors.

#### Author

Prof. Davide Costa

## 7.4 CStringHelper.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * Copyright (C) 2024 Davide Costa
00003  *
00004  * This file is part of RMW desert.
00005  *
00006  * RMW desert is free software: you can redistribute it and/or modify it
00007  * under the terms of the GNU General Public License as published by the
00008  * Free Software Foundation, either version 3 of the License, or any
00009  * later version.
00010  *
00011  * RMW desert is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with RMW desert. If not, see <http://www.gnu.org/licenses/>.
00018  *****/
00019
00032 #ifndef CSTRING_HELPER_H_
00033 #define CSTRING_HELPER_H_
00034
00037 #include "rosidl_runtime_c/u16string.h"

```



```

00038 #include "rosidl_runtime_c/string.h"
00039 #include "rosidl_runtime_c/ul6string_functions.h"
00040 #include "rosidl_runtime_c/string_functions.h"
00041
00042 #include <stdexcept>
00043 #include <vector>
00044 #include <string>
00045
00056 namespace CStringHelper
00057 {
00066     std::string convert_to_std_string(void * str);
00077     std::vector<std::string> convert_to_std_vector_string(void * str_array, size_t size);
00087     std::vector<std::string> convert_sequence_to_std_vector_string(void * str_seq);
00088
00097     std::ul6string convert_to_std_ul6string(void * str);
00108     std::vector<std::ul6string> convert_to_std_vector_ul6string(void * str_array, size_t size);
00118     std::vector<std::ul6string> convert_sequence_to_std_vector_ul6string(void * str_seq);
00119
00129     void assign_string(std::string str, void * field);
00140     void assign_vector_string(std::vector<std::string> cpp_string_vector, void * str_array, size_t
size);
00150     void assign_vector_string_to_sequence(std::vector<std::string> cpp_string_vector, void * str_seq);
00151
00161     void assign_ul6string(std::ul6string str, void * field);
00172     void assign_vector_ul6string(std::vector<std::ul6string> cpp_string_vector, void * str_array, size_t
size);
00182     void assign_vector_ul6string_to_sequence(std::vector<std::ul6string> cpp_string_vector, void *
str_seq);
00183 }
00184
00185
00186 #endif

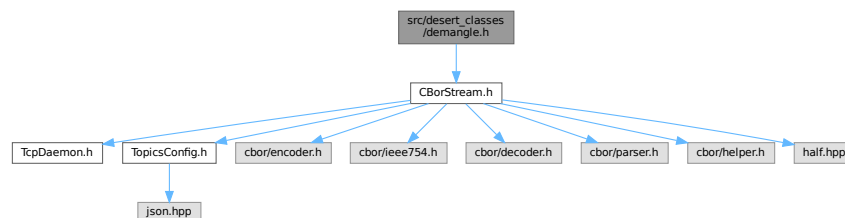
```

## 7.5 src/desert\_classes/demangle.h File Reference

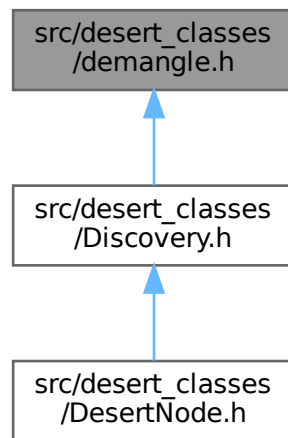
Functions used to demangle topic names during discovery operations.

```
#include "CBorStream.h"
```

Include dependency graph for demangle.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [Discovery](#)  
*Namespace containing discovery functions.*

## Typedefs

- using **Discovery::DemangleFunction** = std::string(\*)(const std::string &)

## Functions

- char \* **Discovery::integer\_to\_string** (int x)
- std::string [Discovery::resolve\\_prefix](#) (const std::string &name, const std::string &prefix)  
*Resolve a prefix.*
- std::string [Discovery::demangle\\_publisher\\_from\\_topic](#) (const std::string &topic\_name)  
*Demangle a publisher.*
- std::string [Discovery::demangle\\_subscriber\\_from\\_topic](#) (const std::string &topic\_name)  
*Demangle a subscriber.*
- std::string [Discovery::demangle\\_topic](#) (const std::string &topic\_name)  
*Demangle a topic.*
- std::string [Discovery::demangle\\_service\\_request\\_from\\_topic](#) (const std::string &topic\_name)  
*Demangle a service request.*
- std::string [Discovery::demangle\\_service\\_reply\\_from\\_topic](#) (const std::string &topic\_name)  
*Demangle a service reply.*
- std::string [Discovery::demangle\\_service\\_from\\_topic](#) (const std::string &topic\_name)  
*Demangle a service.*
- std::string [Discovery::identity\\_demangle](#) (const std::string &name)  
*No demangle.*

## Variables

- `const char *const Discovery::ros_topic_publisher_prefix = integer_to_string(PUBLISHER_TYPE)`
- `const char *const Discovery::ros_topic_subscriber_prefix = integer_to_string(SUBSCRIBER_TYPE)`
- `const char *const Discovery::ros_service_requester_prefix = integer_to_string(CLIENT_TYPE)`
- `const char *const Discovery::ros_service_response_prefix = integer_to_string(SERVICE_TYPE)`

## 7.5.1 Detailed Description

Functions used to demangle topic names during discovery operations.

Demangle functions allows to extract topic names and type names of each entity stored in the common context implementation. Since in this object the informations are divided in writers and readers, they must be converted in publishers, subscribers, clients and services.

## Author

Prof. Davide Costa

## 7.6 demangle.h

[Go to the documentation of this file.](#)

```
00001 /*****
00002  * Copyright (C) 2024 Davide Costa
00003  *
00004  * This file is part of RMW desert.
00005  *
00006  * RMW desert is free software: you can redistribute it and/or modify it
00007  * under the terms of the GNU General Public License as published by the
00008  * Free Software Foundation, either version 3 of the License, or any
00009  * later version.
00010  *
00011  * RMW desert is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with RMW desert. If not, see <http://www.gnu.org/licenses/>.
00018  *****/
00019
00034 #ifndef DEMANGLE_H_
00035 #define DEMANGLE_H_
00036
00039 #include <algorithm>
00040 #include <string>
00041 #include <vector>
00042
00043 #include "rcpputils/find_and_replace.hpp"
00044 #include "rcutils/logging_macros.h"
00045 #include "rcutils/types.h"
00046
00049 #include "CBorStream.h"
00050
00061 namespace Discovery
00062 {
00063
00064     char * integer_to_string(int x);
00065
00066     const char * const ros_topic_publisher_prefix = integer_to_string(PUBLISHER_TYPE);
00067     const char * const ros_topic_subscriber_prefix = integer_to_string(SUBSCRIBER_TYPE);
00068     const char * const ros_service_requester_prefix = integer_to_string(CLIENT_TYPE);
00069     const char * const ros_service_response_prefix = integer_to_string(SERVICE_TYPE);
00070
00080     std::string resolve_prefix(const std::string & name, const std::string & prefix);
00081
00090     std::string demangle_publisher_from_topic(const std::string & topic_name);
00091
00100     std::string demangle_subscriber_from_topic(const std::string & topic_name);
00101 }
```

```

00110     std::string demangle_topic(const std::string & topic_name);
00111
00120     std::string demangle_service_request_from_topic(const std::string & topic_name);
00121
00130     std::string demangle_service_reply_from_topic(const std::string & topic_name);
00131
00140     std::string demangle_service_from_topic(const std::string & topic_name);
00141
00150     std::string identity_demangle(const std::string & name);
00151
00152     using DemangleFunction = std::string (*) (const std::string &);
00153
00154 }
00155
00156 #endif // DEMANGLE_H_

```

## 7.7 src/desert\_classes/DesertClient.h File Reference

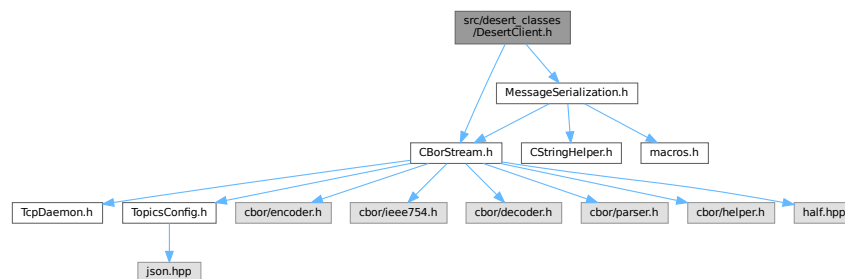
Implementation of the Client structure for DESERT.

```

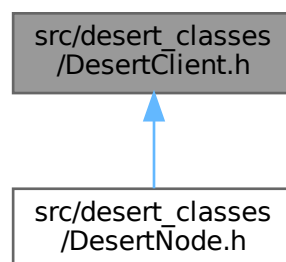
#include "CBorStream.h"
#include "MessageSerialization.h"

```

Include dependency graph for DesertClient.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [DesertClient](#)

### 7.7.1 Detailed Description

Implementation of the Client structure for DESERT.

The [DesertClient](#) class is used to create instances of the various clients registered by ROS. Each of them contains the informations needed to decode the data structure of the messages in the service and allows to send and receive data through specific public functions.

Author

Prof. Davide Costa

## 7.8 DesertClient.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * Copyright (C) 2024 Davide Costa
00003  *
00004  * This file is part of RMW desert.
00005  *
00006  * RMW desert is free software: you can redistribute it and/or modify it
00007  * under the terms of the GNU General Public License as published by the
00008  * Free Software Foundation, either version 3 of the License, or any
00009  * later version.
00010  *
00011  * RMW desert is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with RMW desert. If not, see <http://www.gnu.org/licenses/>.
00018  *****/
00019
00033 #ifndef DESERT_CLIENT_H_
00034 #define DESERT_CLIENT_H_
00035
00038 #include "rosidl_typesupport_introspection_cpp/identifier.hpp"
00039 #include "rosidl_typesupport_introspection_c/identifier.h"
00040 #include "rosidl_typesupport_introspection_cpp/message_introspection.hpp"
00041 #include "rosidl_typesupport_introspection_c/message_introspection.h"
00042 #include "rosidl_typesupport_introspection_cpp/service_introspection.hpp"
00043 #include "rosidl_typesupport_introspection_c/service_introspection.h"
00044 #include "rosidl_typesupport_introspection_cpp/field_types.hpp"
00045 #include "rosidl_typesupport_introspection_c/field_types.h"
00046
00047 #include "rosidl_runtime_c/service_type_support_struct.h"
00048
00049 #include "rmw/types.h"
00050
00051 #include <vector>
00052 #include <string>
00053 #include <regex>
00054
00057 #include "CBorStream.h"
00058 #include "MessageSerialization.h"
00059
00060 class DesertClient
00061 {
00062 public:
00070 DesertClient(std::string service_name, const rosidl_service_type_support_t * type_supports,
rmw_gid_t gid);
00071
00081 bool has_data();
00092 void send_request(const void * req, int64_t * sequence_id);
00103 void read_response(void * res, rmw_service_info_t * req_header);
00104
00112 rmw_gid_t get_gid();
00120 std::string get_service_name();
00128 std::string get_request_type_name();
00136 std::string get_response_type_name();
00137
00138 private:
00139 uint8_t _id;
00140

```

```

00141     rmw_gid_t _gid;
00142     std::string _name;
00143     cbor::TxStream _request_data_stream;
00144     cbor::RxStream _response_data_stream;
00145
00146     int64_t _sequence_id;
00147
00148     int _c_cpp_identifier;
00149     const void * _service;
00150
00151     const void * get_service(const rosidl_service_type_support_t * service_type_support);
00152     const rosidl_service_type_support_t * get_service_type_support(const rosidl_service_type_support_t
* type_supports);
00153
00154 };
00155
00156 #endif

```

## 7.9 src/desert\_classes/DesertGuardCondition.h File Reference

Implementation of the GuardCondition structure for DESERT.

### Classes

- class [DesertGuardCondition](#)

### 7.9.1 Detailed Description

Implementation of the GuardCondition structure for DESERT.

The [DesertGuardCondition](#) class is used to handle trigger signals sent from rclcpp in order to break rcl\_wait, usually when dealing with multithreading executors.

### Author

Prof. Davide Costa

## 7.10 DesertGuardCondition.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * Copyright (C) 2024 Davide Costa
00003  *
00004  * This file is part of RMW desert.
00005  *
00006  * RMW desert is free software: you can redistribute it and/or modify it
00007  * under the terms of the GNU General Public License as published by the
00008  * Free Software Foundation, either version 3 of the License, or any
00009  * later version.
00010  *
00011  * RMW desert is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with RMW desert. If not, see <http://www.gnu.org/licenses/>.
00018  *****/
00019
00031 #ifndef DESERT_GUARD_CONDITION_H_
00032 #define DESERT_GUARD_CONDITION_H_
00033
00036 #include <array>

```

```

00037 #include <atomic>
00038 #include <cassert>
00039 #include <condition_variable>
00040 #include <mutex>
00041 #include <utility>
00042
00043 #include "rcpputils/thread_safety_annotations.hpp"
00044
00047 class DesertGuardCondition
00048 {
00049 public:
00053     DesertGuardCondition();
00054
00061     void trigger();
00062
00071     bool has_triggered();
00072
00081     bool get_has_triggered();
00082
00083 private:
00084     std::atomic_bool _has_triggered;
00085 };
00086
00087 #endif

```

## 7.11 src/desert\_classes/DesertNode.h File Reference

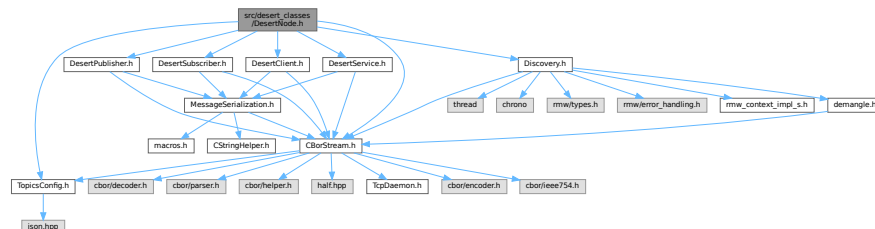
Implementation of the Node structure for DESERT.

```

#include "CBorStream.h"
#include "DesertPublisher.h"
#include "DesertSubscriber.h"
#include "DesertClient.h"
#include "DesertService.h"
#include "Discovery.h"
#include "TopicsConfig.h"

```

Include dependency graph for DesertNode.h:



### Classes

- class [DesertNode](#)

### 7.11.1 Detailed Description

Implementation of the Node structure for DESERT.

The [DesertNode](#) class is used to keep track of all the entities created by a specific node. Each of them is stored in a vector of pointers to the original memory locations mainly used to provide discovery functionalities.

### Author

Prof. Davide Costa

## 7.12 DesertNode.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * Copyright (C) 2024 Davide Costa
00003  *
00004  * This file is part of RMW desert.
00005  *
00006  * RMW desert is free software: you can redistribute it and/or modify it
00007  * under the terms of the GNU General Public License as published by the
00008  * Free Software Foundation, either version 3 of the License, or any
00009  * later version.
00010  *
00011  * RMW desert is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with RMW desert. If not, see <http://www.gnu.org/licenses/>.
00018  *****/
00019
00032 #ifndef DESERT_NODE_H_
00033 #define DESERT_NODE_H_
00034
00037 #include "rmw/rmw.h"
00038 #include "rmw/types.h"
00039
00040 #include <vector>
00041 #include <string>
00042
00045 #include "CBorStream.h"
00046 #include "DesertPublisher.h"
00047 #include "DesertSubscriber.h"
00048 #include "DesertClient.h"
00049 #include "DesertService.h"
00050 #include "Discovery.h"
00051 #include "TopicsConfig.h"
00052
00053 class DesertNode
00054 {
00055 public:
00063 DesertNode(std::string name, std::string namespace_, rmw_gid_t gid);
00064 ~DesertNode();
00065
00074 void add_publisher(DesertPublisher * pub);
00075
00084 void add_subscriber(DesertSubscriber * sub);
00085
00094 void add_client(DesertClient * cli);
00095
00104 void add_service(DesertService * ser);
00105
00114 void remove_publisher(DesertPublisher * pub);
00115
00124 void remove_subscriber(DesertSubscriber * sub);
00125
00134 void remove_client(DesertClient * cli);
00135
00144 void remove_service(DesertService * ser);
00145
00153 rmw_gid_t get_gid();
00154
00155 private:
00156 rmw_gid_t _gid;
00157 std::string _name;
00158 std::string _namespace;
00159 cbor::TxStream _discovery_beacon_data_stream;
00160 cbor::RxStream _discovery_request_data_stream;
00161
00162 std::vector<DesertPublisher *> _publishers;
00163 std::vector<DesertSubscriber *> _subscribers;
00164 std::vector<DesertClient *> _clients;
00165 std::vector<DesertService *> _services;
00166
00167 void publish_all_beacons();
00168
00169 bool _discovery_done;
00170 std::thread _discovery_request_thread;
00171
00172 void _discovery_request();
00173
00174 };
00175
00176 #endif

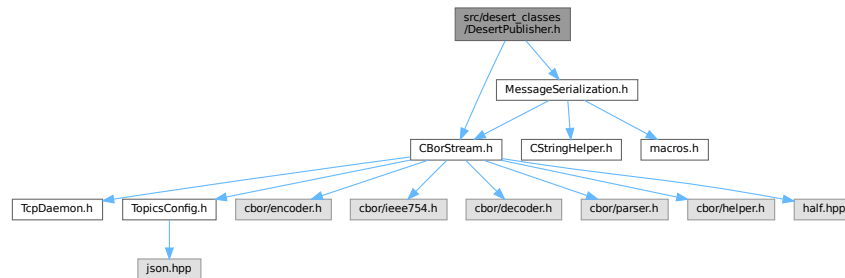
```



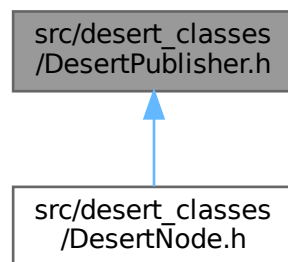
## 7.13 src/desert\_classes/DesertPublisher.h File Reference

Implementation of the Publisher structure for DESERT.

```
#include "CBorStream.h"
#include "MessageSerialization.h"
Include dependency graph for DesertPublisher.h:
```



This graph shows which files directly or indirectly include this file:



### Classes

- class [DesertPublisher](#)

#### 7.13.1 Detailed Description

Implementation of the Publisher structure for DESERT.

The [DesertPublisher](#) class is used to create instances of the various publishers registered by ROS. Each of them contains the informations needed to encode the data structure of the messages in the topic and send them to the stream through specific public functions.

#### Author

Prof. Davide Costa

## 7.14 DesertPublisher.h

[Go to the documentation of this file.](#)

```

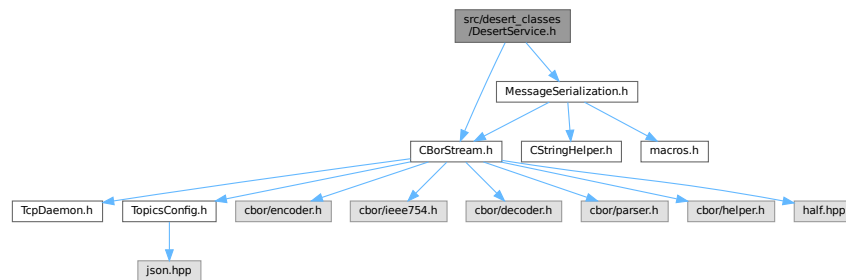
00001 /*****
00002  * Copyright (C) 2024 Davide Costa
00003  *
00004  * This file is part of RMW desert.
00005  *
00006  * RMW desert is free software: you can redistribute it and/or modify it
00007  * under the terms of the GNU General Public License as published by the
00008  * Free Software Foundation, either version 3 of the License, or any
00009  * later version.
00010  *
00011  * RMW desert is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with RMW desert. If not, see <http://www.gnu.org/licenses/>.
00018  *****/
00019
00033 #ifndef DESERT_PUBLISHER_H_
00034 #define DESERT_PUBLISHER_H_
00035
00038 #include "rosidl_typesupport_introspection_cpp/identifier.hpp"
00039 #include "rosidl_typesupport_introspection_c/identifier.h"
00040 #include "rosidl_typesupport_introspection_cpp/message_introspection.hpp"
00041 #include "rosidl_typesupport_introspection_c/message_introspection.h"
00042 #include "rosidl_typesupport_introspection_cpp/service_introspection.hpp"
00043 #include "rosidl_typesupport_introspection_c/service_introspection.h"
00044 #include "rosidl_typesupport_introspection_cpp/field_types.hpp"
00045 #include "rosidl_typesupport_introspection_c/field_types.h"
00046
00047 #include "rosidl_runtime_c/message_type_support_struct.h"
00048
00049 #include "rmw/types.h"
00050
00051 #include <vector>
00052 #include <string>
00053 #include <regex>
00054
00057 #include "CBorStream.h"
00058 #include "MessageSerialization.h"
00059
00060 class DesertPublisher
00061 {
00062 public:
00070 DesertPublisher(std::string topic_name, const rosidl_message_type_support_t * type_supports,
00071 rmw_gid_t gid);
00072
00081 void push(const void * msg);
00082
00090 rmw_gid_t get_gid();
00098 std::string get_topic_name();
00106 std::string get_type_name();
00107
00108 private:
00110 uint8_t _id;
00111 rmw_gid_t _gid;
00112 std::string _name;
00113 cbor::TxStream _data_stream;
00114
00115 int _c_cpp_identifier;
00116 const void * _members;
00117
00118 const void * get_members(const rosidl_message_type_support_t * type_support);
00119 const rosidl_message_type_support_t * get_type_support(const rosidl_message_type_support_t *
00120 type_supports);
00121 };
00122
00123 #endif

```

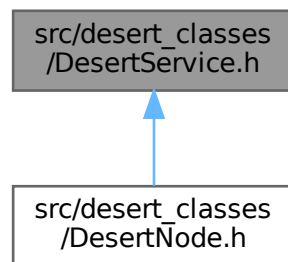
## 7.15 src/desert\_classes/DesertService.h File Reference

Implementation of the Service structure for DESERT.

```
#include "CBorStream.h"
#include "MessageSerialization.h"
Include dependency graph for DesertService.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [DesertService](#)

### 7.15.1 Detailed Description

Implementation of the Service structure for DESERT.

The [DesertService](#) class is used to create instances of the various services registered by ROS. Each of them contains the informations needed to decode the data structure of the messages in the stream and allows to send and receive data through specific public functions.

## Author

Prof. Davide Costa

## 7.16 DesertService.h

[Go to the documentation of this file.](#)

```

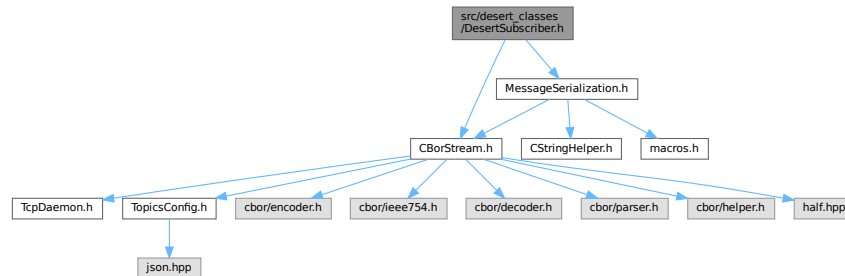
00001 /*****
00002  * Copyright (C) 2024 Davide Costa
00003  *
00004  * This file is part of RMW desert.
00005  *
00006  * RMW desert is free software: you can redistribute it and/or modify it
00007  * under the terms of the GNU General Public License as published by the
00008  * Free Software Foundation, either version 3 of the License, or any
00009  * later version.
00010  *
00011  * RMW desert is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with RMW desert. If not, see <http://www.gnu.org/licenses/>.
00018  *****/
00019
00033 #ifndef DESERT_SERVICE_H_
00034 #define DESERT_SERVICE_H_
00035
00038 #include "rosidl_typesupport_introspection_cpp/identifier.hpp"
00039 #include "rosidl_typesupport_introspection_c/identifier.h"
00040 #include "rosidl_typesupport_introspection_cpp/message_introspection.hpp"
00041 #include "rosidl_typesupport_introspection_c/message_introspection.h"
00042 #include "rosidl_typesupport_introspection_cpp/service_introspection.hpp"
00043 #include "rosidl_typesupport_introspection_c/service_introspection.h"
00044 #include "rosidl_typesupport_introspection_cpp/field_types.hpp"
00045 #include "rosidl_typesupport_introspection_c/field_types.h"
00046
00047 #include "rosidl_runtime_c/service_type_support_struct.h"
00048
00049 #include "rmw/types.h"
00050
00051 #include <vector>
00052 #include <string>
00053 #include <regex>
00054
00057 #include "CBorStream.h"
00058 #include "MessageSerialization.h"
00059
00060 class DesertService
00061 {
00062 public:
00070 DesertService(std::string service_name, const rosidl_service_type_support_t * type_supports,
00071 rmw_gid_t gid);
00072
00081 bool has_data();
00092 void read_request(void * req, rmw_service_info_t * req_header);
00103 void send_response(void * res, rmw_request_id_t * req_header);
00104
00112 rmw_gid_t get_gid();
00120 std::string get_service_name();
00128 std::string get_request_type_name();
00136 std::string get_response_type_name();
00137
00138
00139 private:
00140 uint8_t _id;
00141 rmw_gid_t _gid;
00142 std::string _name;
00143 cbor::RxStream _request_data_stream;
00144 cbor::TxStream _response_data_stream;
00145
00146 int64_t _sequence_id;
00147
00148 int _c_cpp_identifier;
00149 const void * _service;
00150
00151 const void * get_service(const rosidl_service_type_support_t * service_type_support);
00152 const rosidl_service_type_support_t * get_service_type_support(const rosidl_service_type_support_t
00153 * type_supports);
00154 };
00155
00156 #endif

```

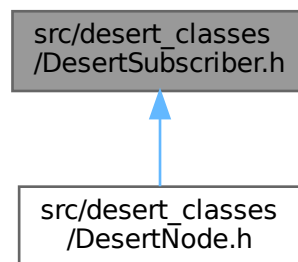
## 7.17 src/desert\_classes/DesertSubscriber.h File Reference

Implementation of the Subscriber structure for DESERT.

```
#include "CBorStream.h"
#include "MessageSerialization.h"
Include dependency graph for DesertSubscriber.h:
```



This graph shows which files directly or indirectly include this file:



### Classes

- class [DesertSubscriber](#)

### 7.17.1 Detailed Description

Implementation of the Subscriber structure for DESERT.

The [DesertSubscriber](#) class is used to create instances of the various subscribers registered by ROS. Each of them contains the informations needed to decode the data structure of the messages in the topic through specific public functions.

### Author

Prof. Davide Costa

## 7.18 DesertSubscriber.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * Copyright (C) 2024 Davide Costa
00003  *
00004  * This file is part of RMW desert.
00005  *
00006  * RMW desert is free software: you can redistribute it and/or modify it
00007  * under the terms of the GNU General Public License as published by the
00008  * Free Software Foundation, either version 3 of the License, or any
00009  * later version.
00010  *
00011  * RMW desert is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with RMW desert. If not, see <http://www.gnu.org/licenses/>.
00018  *****/
00019
00032 #ifndef DESERT_SUBSCRIBER_H_
00033 #define DESERT_SUBSCRIBER_H_
00034
00037 #include "rosidl_typesupport_introspection_cpp/identifier.hpp"
00038 #include "rosidl_typesupport_introspection_c/identifier.h"
00039 #include "rosidl_typesupport_introspection_cpp/message_introspection.hpp"
00040 #include "rosidl_typesupport_introspection_c/message_introspection.h"
00041 #include "rosidl_typesupport_introspection_cpp/service_introspection.hpp"
00042 #include "rosidl_typesupport_introspection_c/service_introspection.h"
00043 #include "rosidl_typesupport_introspection_cpp/field_types.hpp"
00044 #include "rosidl_typesupport_introspection_c/field_types.h"
00045
00046 #include "rosidl_runtime_c/message_type_support_struct.h"
00047
00048 #include "rmw/types.h"
00049
00050 #include <vector>
00051 #include <string>
00052 #include <regex>
00053
00056 #include "CBorStream.h"
00057 #include "MessageSerialization.h"
00058
00059 class DesertSubscriber
00060 {
00061 public:
00062     DesertSubscriber(std::string topic_name, const rosidl_message_type_support_t * type_supports,
00063                     rmw_gid_t gid);
00070
00080     bool has_data();
00090     void read_data(void * msg);
00091
00099     rmw_gid_t get_gid();
00107     std::string get_topic_name();
00115     std::string get_type_name();
00116
00117 private:
00118     uint8_t _id;
00119     rmw_gid_t _gid;
00120     std::string _name;
00121     cbor::RxStream _data_stream;
00122
00123     int _c_cpp_identifier;
00124     const void * _members;
00125
00126     const void * get_members(const rosidl_message_type_support_t * type_support);
00127     const rosidl_message_type_support_t * get_type_support(const rosidl_message_type_support_t *
00128 type_supports);
00129 };
00130
00131 #endif

```

## 7.19 src/desert\_classes/DesertWaitSet.h File Reference

Implementation of the WaitSet structure for DESERT.

**Classes**

- class [DesertWaitset](#)

**7.19.1 Detailed Description**

Implementation of the WaitSet structure for DESERT.

Unimplemented class included for future expansions

**Author**

Prof. Davide Costa

**7.20 DesertWaitSet.h**

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * Copyright (C) 2024 Davide Costa
00003  *
00004  * This file is part of RMW desert.
00005  *
00006  * RMW desert is free software: you can redistribute it and/or modify it
00007  * under the terms of the GNU General Public License as published by the
00008  * Free Software Foundation, either version 3 of the License, or any
00009  * later version.
00010  *
00011  * RMW desert is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with RMW desert. If not, see <http://www.gnu.org/licenses/>.
00018  *****/
00019
00030 #ifndef DESERT_WAIT_SET_H_
00031 #define DESERT_WAIT_SET_H_
00032
00033 class DesertWaitset
00034 {
00035 public:
00036     DesertWaitset ()
00037     {}
00038
00039     std::mutex lock;
00040     bool inuse;
00041 };
00042
00043 #endif

```

**7.21 src/desert\_classes/Discovery.h File Reference**

Namespace used to provide discovery functionalities.

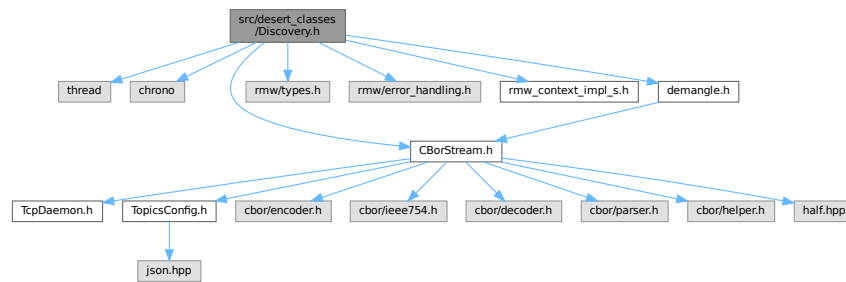
```

#include <thread>
#include <chrono>
#include "CBorStream.h"
#include "rmw/types.h"
#include "rmw/error_handling.h"
#include "rmw_context_impl_s.h"

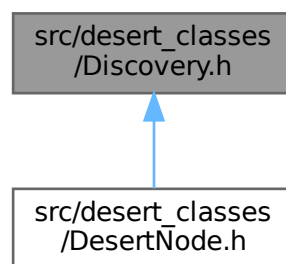
```

```
#include "demangle.h"
```

Include dependency graph for Discovery.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [Discovery](#)  
*Namespace containing discovery functions.*

## Functions

- void [Discovery::discovery\\_thread](#) (rmw\_context\_impl\_t \*impl)  
*Thread handling discovery beacons.*
- rmw\_ret\_t [Discovery::discovery\\_thread\\_start](#) (rmw\_context\_impl\_t \*impl)  
*Initialize the discovery thread.*
- rmw\_ret\_t [Discovery::discovery\\_thread\\_stop](#) (rmw\_context\_impl\_t \*impl)  
*Stop the discovery thread.*
- void [Discovery::send\\_discovery\\_beacon](#) (cbor::TxStream stream, std::string node\_name, std::string node\_namespace, int entity\_type, rmw\_gid\_t entity\_gid, std::string topic\_name, std::string type\_name, bool disconnect)  
*Send a discovery beacon.*
- void [Discovery::send\\_discovery\\_request](#) (cbor::TxStream stream)  
*Send a discovery request.*



### 7.21.1 Detailed Description

Namespace used to provide discovery functionalities.

The middleware layer of a ROS stack must implement functionalities used to inform each node of the network structure of the other nodes connected, with their names and topics. Since this operation is quite resource-consuming and the underwater channel has a limited bandwidth, it is possible to disable it.

#### Author

Prof. Davide Costa

## 7.22 Discovery.h

[Go to the documentation of this file.](#)

```

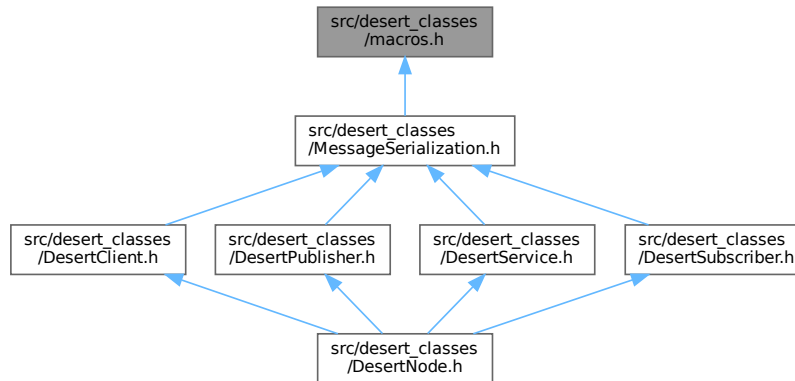
00001 /*****
00002  * Copyright (C) 2024 Davide Costa
00003  *
00004  * This file is part of RMW desert.
00005  *
00006  * RMW desert is free software: you can redistribute it and/or modify it
00007  * under the terms of the GNU General Public License as published by the
00008  * Free Software Foundation, either version 3 of the License, or any
00009  * later version.
00010  *
00011  * RMW desert is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with RMW desert. If not, see <http://www.gnu.org/licenses/>.
00018  *****/
00019
00034 #include <thread>
00035 #include <chrono>
00036
00037 #include "CBorStream.h"
00038
00039 #include "rmw/types.h"
00040 #include "rmw/error_handling.h"
00041
00042 #include "rmw_context_impl_s.h"
00043
00044 #include "demangle.h"
00045
00046 #ifndef DISCOVERY_H_
00047 #define DISCOVERY_H_
00048
00059 namespace Discovery
00060 {
00061
00070 void discovery_thread(rmw_context_impl_t * impl);
00071
00081 rmw_ret_t discovery_thread_start(rmw_context_impl_t * impl);
00082
00092 rmw_ret_t discovery_thread_stop(rmw_context_impl_t * impl);
00093
00109 void send_discovery_beacon(cbor::TxStream stream, std::string node_name, std::string node_namespace,
int entity_type, rmw_gid_t entity_gid, std::string topic_name, std::string type_name, bool
disconnect);
00110
00119 void send_discovery_request(cbor::TxStream stream);
00120
00121 }
00122
00123 #endif

```

## 7.23 src/desert\_classes/macros.h File Reference

Header containing C sequence macros.

This graph shows which files directly or indirectly include this file:



### Macros

- `#define SPECIALIZE_GENERIC_C_SEQUENCE(C_NAME, C_TYPE)`

### 7.23.1 Detailed Description

Header containing C sequence macros.

The C data type implementation is more complicated than the C++ one, because complex types like vectors have to be manually managed and this header contains definitions used to create dynamic element sequences.

#### Author

Prof. Davide Costa

### 7.23.2 Macro Definition Documentation

#### 7.23.2.1 SPECIALIZE\_GENERIC\_C\_SEQUENCE

```

#define SPECIALIZE_GENERIC_C_SEQUENCE (
    C_NAME,
    C_TYPE )

```

#### Value:

```

template<> \
struct GenericCSequence<C_TYPE> \
{ \
    using type = rosidl_runtime_c__ ## C_NAME ## __Sequence; \
    \
    static void fini(type * sequence) { \
        rosidl_runtime_c__ ## C_NAME ## __Sequence__fini(sequence); \
    } \
    \
    static bool init(type * sequence, size_t size) { \
        return rosidl_runtime_c__ ## C_NAME ## __Sequence__init(sequence, size); \
    } \
};

```

## 7.24 macros.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * Copyright (C) 2024 Davide Costa
00003  *
00004  * This file is part of RMW desert.
00005  *
00006  * RMW desert is free software: you can redistribute it and/or modify it
00007  * under the terms of the GNU General Public License as published by the
00008  * Free Software Foundation, either version 3 of the License, or any
00009  * later version.
00010  *
00011  * RMW desert is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with RMW desert. If not, see <http://www.gnu.org/licenses/>.
00018  *****/
00019
00032 #ifndef MACROS_H_
00033 #define MACROS_H_
00034
00037 #include "rosidl_runtime_c/primitives_sequence.h"
00038 #include "rosidl_runtime_c/primitives_sequence_functions.h"
00039
00042 #define SPECIALIZE_GENERIC_C_SEQUENCE(C_NAME, C_TYPE) \
00043     template<> \
00044     struct GenericCSequence<C_TYPE> \
00045     { \
00046         using type = rosidl_runtime_c_ ## C_NAME ## __Sequence; \
00047     \
00048         static void fini(type * sequence) { \
00049             rosidl_runtime_c_ ## C_NAME ## __Sequence__fini(sequence); \
00050         } \
00051     \
00052         static bool init(type * sequence, size_t size) { \
00053             return rosidl_runtime_c_ ## C_NAME ## __Sequence__init(sequence, size); \
00054         } \
00055     };
00056
00057 template<typename T>
00058 struct GenericCSequence;
00059
00060 // multiple definitions of ambiguous primitive types
00061 SPECIALIZE_GENERIC_C_SEQUENCE(bool, bool)
00062 SPECIALIZE_GENERIC_C_SEQUENCE(byte, uint8_t)
00063 SPECIALIZE_GENERIC_C_SEQUENCE(char, char)
00064 SPECIALIZE_GENERIC_C_SEQUENCE(float32, float)
00065 SPECIALIZE_GENERIC_C_SEQUENCE(float64, double)
00066 SPECIALIZE_GENERIC_C_SEQUENCE(int8, int8_t)
00067 SPECIALIZE_GENERIC_C_SEQUENCE(int16, int16_t)
00068 SPECIALIZE_GENERIC_C_SEQUENCE(uint16, uint16_t)
00069 SPECIALIZE_GENERIC_C_SEQUENCE(int32, int32_t)
00070 SPECIALIZE_GENERIC_C_SEQUENCE(uint32, uint32_t)
00071 SPECIALIZE_GENERIC_C_SEQUENCE(int64, int64_t)
00072 SPECIALIZE_GENERIC_C_SEQUENCE(uint64, uint64_t)
00073
00074 #endif // MACROS_HPP_

```

## 7.25 src/desert\_classes/MessageSerialization.h File Reference

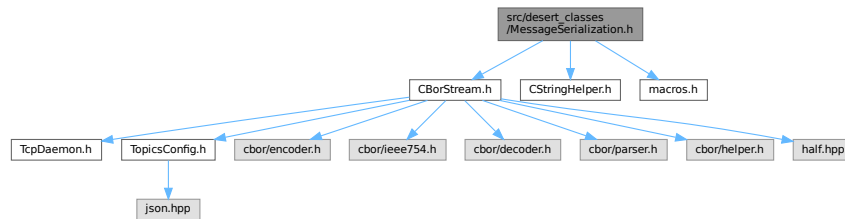
Namespace containing serialization functions.

```

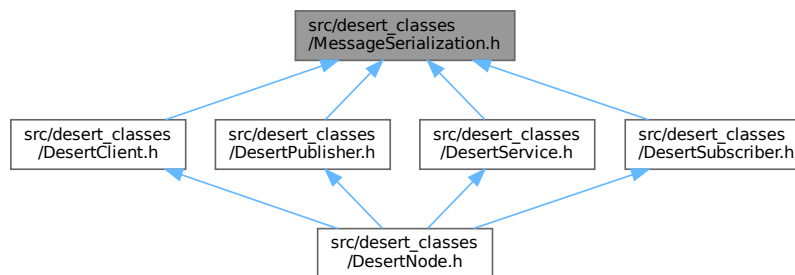
#include "CBorStream.h"
#include "CStringHelper.h"
#include "macros.h"

```

Include dependency graph for MessageSerialization.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [MessageSerialization](#)  
*Namespace containing serialization functions.*

## Macros

- `#define INTROSPECTION_C_MEMBER` `rosidl_typesupport_introspection_c__MessageMember`
- `#define INTROSPECTION_CPP_MEMBER` `rosidl_typesupport_introspection_cpp::MessageMember`
- `#define INTROSPECTION_C_MEMBERS` `rosidl_typesupport_introspection_c__MessageMembers`
- `#define INTROSPECTION_CPP_MEMBERS` `rosidl_typesupport_introspection_cpp::MessageMembers`
- `#define INTROSPECTION_C_SERVICE_MEMBERS` `rosidl_typesupport_introspection_c__ServiceMembers`
- `#define INTROSPECTION_CPP_SERVICE_MEMBERS` `rosidl_typesupport_introspection_cpp::ServiceMembers`

## Functions

- `template<typename T >`  
`void MessageSerialization::serialize\_field (const INTROSPECTION_CPP_MEMBER *member, void *field, cbor::TxStream &stream)`  
*Serialize a C++ field.*

- `template<typename T >`  
`void MessageSerialization::serialize_field (const INTROSPECTION_C_MEMBER *member, void *field, cbor::TxStream &stream)`  
*Serialize a C field.*
- `template<typename MembersType >`  
`void MessageSerialization::serialize (const void *msg, const MembersType *casted_members, cbor::TxStream &stream)`  
*Serialize a ROS message, request or response.*
- `template<typename T >`  
`void MessageSerialization::deserialize_field (const INTROSPECTION_CPP_MEMBER *member, void *field, cbor::RxStream &stream)`  
*Deserialize a C++ field.*
- `template<typename T >`  
`void MessageSerialization::deserialize_field (const INTROSPECTION_C_MEMBER *member, void *field, cbor::RxStream &stream)`  
*Deserialize a C field.*
- `template<typename MembersType >`  
`void MessageSerialization::deserialize (void *msg, const MembersType *casted_members, cbor::RxStream &stream)`  
*Deserialize a ROS message, request or response.*

### 7.25.1 Detailed Description

Namespace containing serialization functions.

The message data structure coming from upper layers is interpreted using type support informations passed by ROS2 during the creation of publishers, subscribers, clients and services. Those functions are used to compute the exact position that every data type must assume in memory and then calls TxStream or RxStream to receive or write them in the assigned location.

#### Author

Prof. Davide Costa

## 7.26 MessageSerialization.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * Copyright (C) 2024 Davide Costa
00003  *
00004  * This file is part of RMW desert.
00005  *
00006  * RMW desert is free software: you can redistribute it and/or modify it
00007  * under the terms of the GNU General Public License as published by the
00008  * Free Software Foundation, either version 3 of the License, or any
00009  * later version.
00010  *
00011  * RMW desert is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with RMW desert. If not, see <http://www.gnu.org/licenses/>.
00018  *****/
00019
00034 #ifndef MESSAGE_SERIALIZATION_H_
00035 #define MESSAGE_SERIALIZATION_H_
00036
00037 #include "CBorStream.h"

```

```

00038 #include "CStringHelper.h"
00039 #include "macros.h"
00040
00043 #include <vector>
00044 #include <string>
00045
00048 #define INTROSPECTION_C_MEMBER rosidl_typesupport_introspection_c__MessageMember
00049 #define INTROSPECTION_CPP_MEMBER rosidl_typesupport_introspection_cpp::MessageMember
00050
00051 #define INTROSPECTION_C_MEMBERS rosidl_typesupport_introspection_c__MessageMembers
00052 #define INTROSPECTION_CPP_MEMBERS rosidl_typesupport_introspection_cpp::MessageMembers
00053
00054 #define INTROSPECTION_C_SERVICE_MEMBERS rosidl_typesupport_introspection_c__ServiceMembers
00055 #define INTROSPECTION_CPP_SERVICE_MEMBERS rosidl_typesupport_introspection_cpp::ServiceMembers
00056
00067 namespace MessageSerialization
00068 {
00069
00081     template<typename T>
00082     void serialize_field(const INTROSPECTION_CPP_MEMBER * member, void * field, cbor::TxStream & stream)
00083     {
00084         if (!member->is_array_)
00085         {
00086             stream << * static_cast<T *>(field);
00087         }
00088         else if (member->array_size_ && !member->is_upper_bound_)
00089         {
00090             stream.serialize_sequence(static_cast<T *>(field), member->array_size_);
00091         }
00092         else
00093         {
00094             std::vector<T> & data = *reinterpret_cast<std::vector<T> *>(field);
00095             stream << data;
00096         }
00097     }
00098
00110     template<typename T>
00111     void serialize_field(const INTROSPECTION_C_MEMBER * member, void * field, cbor::TxStream & stream)
00112     {
00113         // String specific implementation
00114         if constexpr(std::is_same_v<T, std::string>)
00115         {
00116             if (!member->is_array_)
00117             {
00118                 stream << CStringHelper::convert_to_std_string(field);
00119             }
00120             else if (member->array_size_ && !member->is_upper_bound_)
00121             {
00122                 stream << CStringHelper::convert_to_std_vector_string(field, member->array_size_);
00123             }
00124             else
00125             {
00126                 printf("WARNING: non-fixed size sequences are currently sperimental\n");
00127                 stream << CStringHelper::convert_sequence_to_std_vector_string(field);
00128             }
00129         }
00130         // U16string specific implementation
00131         else if constexpr(std::is_same_v<T, std::u16string>)
00132         {
00133             if (!member->is_array_)
00134             {
00135                 stream << CStringHelper::convert_to_std_u16string(field);
00136             }
00137             else if (member->array_size_ && !member->is_upper_bound_)
00138             {
00139                 stream << CStringHelper::convert_to_std_vector_u16string(field, member->array_size_);
00140             }
00141             else
00142             {
00143                 printf("WARNING: non-fixed size sequences are currently sperimental\n");
00144                 stream << CStringHelper::convert_sequence_to_std_vector_u16string(field);
00145             }
00146         }
00147         // Generic implementation
00148         else
00149         {
00150             if (!member->is_array_)
00151             {
00152                 stream << * static_cast<T *>(field);
00153             }
00154             else if (member->array_size_ && !member->is_upper_bound_)
00155             {
00156                 stream.serialize_sequence(static_cast<T *>(field), member->array_size_);
00157             }
00158             else
00159             {
00160                 printf("WARNING: non-fixed size sequences are currently sperimental\n");

```

```

00161         auto & data = *reinterpret_cast<typename GenericCSequence<T>::type *>(field);
00162
00163         // Serialize length
00164         stream << (uint32_t)data.size;
00165
00166         stream.serialize_sequence(reinterpret_cast<T *>(data.data), data.size);
00167     }
00168 }
00169 }
00170
00171 template<typename MembersType>
00172 void serialize(const void * msg, const MembersType * casted_members, cbor::TxStream & stream)
00173 {
00174     for (uint32_t i = 0; i < casted_members->member_count_; ++i) {
00175         const auto member = casted_members->members_ + i;
00176         void * field = const_cast<char *>(static_cast<const char *>(msg)) + member->offset_;
00177         switch (member->type_id_) {
00178             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_MESSAGE:
00179             {
00180                 auto sub_members = static_cast<const MembersType *>(member->members->data);
00181                 if (!member->is_array_) {
00182                     serialize(field, sub_members, stream);
00183                 }
00184                 else if (member->array_size_ && !member->is_upper_bound_)
00185                 {
00186                     for (size_t index = 0; index < member->array_size_; ++index) {
00187                         serialize(member->get_function(field, index), sub_members, stream);
00188                     }
00189                 }
00190                 else
00191                 {
00192                     size_t array_size = member->size_function(field);
00193
00194                     if (member->is_upper_bound_ && array_size > member->array_size_)
00195                     {
00196                         throw std::runtime_error("Sequence overcomes the maximum length");
00197                     }
00198
00199                     // Serialize length
00200                     stream << (uint32_t)array_size;
00201
00202                     for (size_t index = 0; index < array_size; ++index) {
00203                         serialize(member->get_function(field, index), sub_members, stream);
00204                     }
00205                 }
00206             }
00207             break;
00208             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_BOOLEAN:
00209             {
00210                 if (!member->is_array_)
00211                 {
00212                     // Don't cast to bool here because if the bool is uninitialized the random value can't be
00213                     deserialized
00214                     stream << (*static_cast<uint8_t *>(field) ? true : false);
00215                 }
00216                 else
00217                 {
00218                     serialize_field<bool>(member, field, stream);
00219                 }
00220             }
00221             break;
00222             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_OCTET:
00223             {
00224                 //throw std::runtime_error("OCTET type unsupported");
00225             }
00226             break;
00227             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT8:
00228             {
00229                 serialize_field<uint8_t>(member, field, stream);
00230             }
00231             break;
00232             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_CHAR:
00233             {
00234                 serialize_field<char>(member, field, stream);
00235             }
00236             break;
00237             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT8:
00238             {
00239                 serialize_field<int8_t>(member, field, stream);
00240             }
00241             break;
00242             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_FLOAT:
00243             {
00244                 serialize_field<float>(member, field, stream);
00245             }
00246             break;
00247             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_DOUBLE:
00248             {
00249                 serialize_field<double>(member, field, stream);
00250             }
00251             break;
00252             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT16:
00253             {
00254                 serialize_field<int16_t>(member, field, stream);
00255             }
00256             break;
00257             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT16:
00258             {
00259                 serialize_field<uint16_t>(member, field, stream);
00260             }
00261             break;
00262             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT32:
00263             {
00264                 serialize_field<int32_t>(member, field, stream);
00265             }
00266             break;
00267             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT32:
00268             {
00269                 serialize_field<uint32_t>(member, field, stream);
00270             }
00271             break;
00272         }
00273     }
00274 }

```

```

00259         serialize_field<uint32_t>(member, field, stream);
00260         break;
00261     case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT64:
00262         serialize_field<int64_t>(member, field, stream);
00263         break;
00264     case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT64:
00265         serialize_field<uint64_t>(member, field, stream);
00266         break;
00267     case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_STRING:
00268         serialize_field<std::string>(member, field, stream);
00269         break;
00270     case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_WSTRING:
00271         serialize_field<std::u16string>(member, field, stream);
00272         break;
00273     default:
00274         throw std::runtime_error("unknown type");
00275     }
00276 }
00277 }
00278
00290 template<typename T>
00291 void deserialize_field(const INTROSPECTION_CPP_MEMBER * member, void * field, cbor::RxStream &
stream)
00292 {
00293     if (!member->is_array_) {
00294         stream » *static_cast<T *>(field);
00295     }
00296     else if (member->array_size_ && !member->is_upper_bound_)
00297     {
00298         stream.deserialize_sequence(static_cast<T *>(field), member->array_size_);
00299     }
00300     else
00301     {
00302         auto & vector = *reinterpret_cast<std::vector<T *>(field);
00303         new(&vector) std::vector<T>;
00304         stream » vector;
00305     }
00306 }
00307
00319 template<typename T>
00320 void deserialize_field(const INTROSPECTION_C_MEMBER * member, void * field, cbor::RxStream & stream)
00321 {
00322     // String specific implementation
00323     if constexpr(std::is_same_v<T, std::string>)
00324     {
00325         if (!member->is_array_)
00326         {
00327             std::string str;
00328             stream » str;
00329             CStringHelper::assign_string(str, field);
00330         }
00331         else if (member->array_size_ && !member->is_upper_bound_)
00332         {
00333             std::vector<std::string> cpp_string_vector;
00334             stream » cpp_string_vector;
00335             CStringHelper::assign_vector_string(cpp_string_vector, field, member->array_size_);
00336         }
00337         else
00338         {
00339             printf("WARNING: non-fixed size sequences are currently sperimental\n");
00340             std::vector<std::string> cpp_string_vector;
00341             stream » cpp_string_vector;
00342             CStringHelper::assign_vector_string_to_sequence(cpp_string_vector, field);
00343         }
00344     }
00345 }
00346
00347 // U16string specific implementation
00348 else if constexpr(std::is_same_v<T, std::u16string>)
00349 {
00350     if (!member->is_array_)
00351     {
00352         std::u16string str;
00353         stream » str;
00354         CStringHelper::assign_u16string(str, field);
00355     }
00356     else if (member->array_size_ && !member->is_upper_bound_)
00357     {
00358         std::vector<std::u16string> cpp_string_vector;
00359         stream » cpp_string_vector;
00360         CStringHelper::assign_vector_u16string(cpp_string_vector, field, member->array_size_);
00361     }
00362 }
00363 else
00364 {
00365     printf("WARNING: non-fixed size sequences are currently sperimental\n");
00366     std::vector<std::u16string> cpp_string_vector;

```



```

00367         stream >> cpp_string_vector;
00368
00369         CStringHelper::assign_vector_ul6string_to_sequence(cpp_string_vector, field);
00370     }
00371 }
00372 // Generic implementation
00373 else
00374 {
00375     if (!member->is_array_)
00376     {
00377         stream >> *static_cast<T*>(field);
00378     }
00379     else if (member->array_size_ && !member->is_upper_bound_)
00380     {
00381         stream.deserialize_sequence(static_cast<T*>(field), member->array_size_);
00382     }
00383     else
00384     {
00385         printf("WARNING: non-fixed size sequences are currently sperimental\n");
00386         auto & data = *reinterpret_cast<typename GenericCSequence<T>::type*>(field);
00387         uint32_t size = 0;
00388         stream >> size;
00389         size_t dsize = static_cast<size_t>(size);
00390
00391         if (!GenericCSequence<T>::init(&data, dsize))
00392         {
00393             throw std::runtime_error("unable to initialize GenericCSequence");
00394         }
00395
00396         stream.deserialize_sequence(reinterpret_cast<T*>(data.data), dsize);
00397     }
00398 }
00399 }
00400
00414 template<typename MembersType>
00415 void deserialize(void * msg, const MembersType * casted_members, cbor::RxStream & stream)
00416 {
00417     for (uint32_t i = 0; i < casted_members->member_count_; ++i) {
00418         const auto member = casted_members->members_ + i;
00419         void * field = static_cast<char*>(msg) + member->offset_;
00420         switch (member->type_id_) {
00421             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_MESSAGE:
00422             {
00423                 auto sub_members = static_cast<const MembersType*>(member->members_->data);
00424                 if (!member->is_array_) {
00425                     deserialize(field, sub_members, stream);
00426                 }
00427                 else if (member->array_size_ && !member->is_upper_bound_)
00428                 {
00429                     for (size_t index = 0; index < member->array_size_; ++index) {
00430                         deserialize(member->get_function(field, index), sub_members, stream);
00431                     }
00432                 }
00433                 else
00434                 {
00435                     // Deserialize length
00436                     uint32_t array_size = 0;
00437                     stream >> array_size;
00438
00439                     auto vector = reinterpret_cast<std::vector<unsigned char*>*>(field);
00440                     new(vector) std::vector<unsigned char>;
00441                     member->resize_function(field, array_size);
00442
00443                     for (size_t index = 0; index < array_size; ++index) {
00444                         deserialize(member->get_function(field, index), sub_members, stream);
00445                     }
00446                 }
00447             }
00448             break;
00449             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_BOOLEAN:
00450                 deserialize_field<bool>(member, field, stream);
00451                 break;
00452             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_OCTET:
00453                 //throw std::runtime_error("OCTET type unsupported");
00454                 break;
00455             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT8:
00456                 deserialize_field<uint8_t>(member, field, stream);
00457                 break;
00458             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_CHAR:
00459                 deserialize_field<char>(member, field, stream);
00460                 break;
00461             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT8:
00462                 deserialize_field<int8_t>(member, field, stream);
00463                 break;
00464             case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_FLOAT:
00465                 deserialize_field<float>(member, field, stream);
00466                 break;

```

```

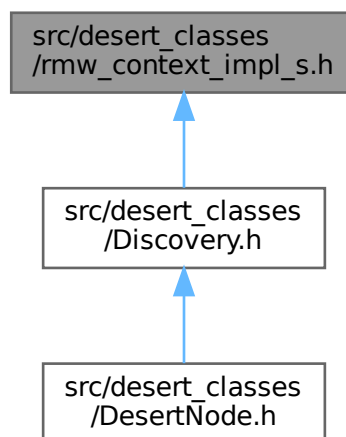
00467     case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_DOUBLE:
00468         deserialize_field<double>(member, field, stream);
00469         break;
00470     case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT16:
00471         deserialize_field<int16_t>(member, field, stream);
00472         break;
00473     case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT16:
00474         deserialize_field<uint16_t>(member, field, stream);
00475         break;
00476     case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT32:
00477         deserialize_field<int32_t>(member, field, stream);
00478         break;
00479     case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT32:
00480         deserialize_field<uint32_t>(member, field, stream);
00481         break;
00482     case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT64:
00483         deserialize_field<int64_t>(member, field, stream);
00484         break;
00485     case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT64:
00486         deserialize_field<uint64_t>(member, field, stream);
00487         break;
00488     case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_STRING:
00489         deserialize_field<std::string>(member, field, stream);
00490         break;
00491     case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_WSTRING:
00492         deserialize_field<std::wstring>(member, field, stream);
00493         break;
00494     default:
00495         throw std::runtime_error("unknown type");
00496 }
00497 }
00498 }
00499
00500 }
00501
00502
00503 #endif

```

## 7.27 src/desert\_classes/rmw\_context\_impl\_s.h File Reference

Implementation for the context variable.

This graph shows which files directly or indirectly include this file:



### Classes

- struct [rmw\\_context\\_impl\\_s](#)

### 7.27.1 Detailed Description

Implementation for the context variable.

Context is used to store informations about the current network structure using variables included from `rmw_dds_↵_common`. This struct provides an implementation for `rmw_context_impl_t` and it must be present to avoid compile errors.

Author

Prof. Davide Costa

## 7.28 rmw\_context\_impl\_s.h

[Go to the documentation of this file.](#)

```

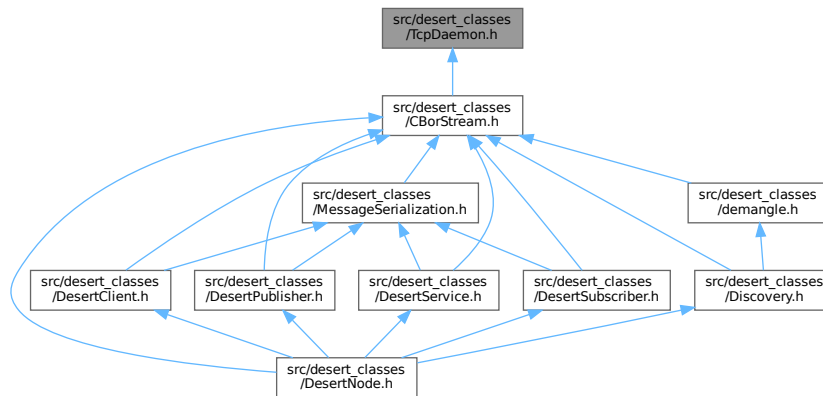
00001 /*****
00002  * Copyright (C) 2024 Davide Costa
00003  *
00004  * This file is part of RMW desert.
00005  *
00006  * RMW desert is free software: you can redistribute it and/or modify it
00007  * under the terms of the GNU General Public License as published by the
00008  * Free Software Foundation, either version 3 of the License, or any
00009  * later version.
00010  *
00011  * RMW desert is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with RMW desert. If not, see <http://www.gnu.org/licenses/>.
00018  *****/
00019
00034 #include "rcpputils/scope_exit.hpp"
00035 #include "rmw_dds_common/context.hpp"
00036 #include "rmw_dds_common/graph_cache.hpp"
00037 #include "rmw_dds_common/msg/participant_entities_info.hpp"
00038 #include "rmw_dds_common/qos.hpp"
00039 #include "rmw_dds_common/security.hpp"
00040
00043 #ifndef RMW_CONTEXT_IMPL_H_
00044 #define RMW_CONTEXT_IMPL_H_
00045
00046 struct rmw_context_impl_s
00047 {
00048     rmw_dds_common::Context common;
00049     bool is_shutdown{false};
00050
00051     rmw_context_impl_s()
00052     : common()
00053     {
00054         /* destructor relies on these being initialized properly */
00055         common.thread_is_running.store(false);
00056         common.graph_guard_condition = nullptr;
00057         common.pub = nullptr;
00058         common.sub = nullptr;
00059     }
00060
00061     ~rmw_context_impl_s()
00062     {
00063     }
00064 };
00065
00066 #endif

```

## 7.29 src/desert\_classes/TcpDaemon.h File Reference

Class used to send and receive data from the DESERT socket.

This graph shows which files directly or indirectly include this file:



## Classes

- class [TcpDaemon](#)

## Macros

- #define **MAX\_PACKET\_LENGTH** 512
- #define **ADDRESS** "127.0.0.1"
- #define **START\_MARKER** 0b10011001
- #define **END\_MARKER** 0b01010101
- #define **BYTE\_MASK** 0b11111111

### 7.29.1 Detailed Description

Class used to send and receive data from the DESERT socket.

The DESERT protocol stack interacts with the application level through a socket, used to send and receive a binary stream containing packets. This class connects to the socket and creates two threads, that run continuously to store and send packets in the static members rx\_packets and tx\_packets

## Author

Prof. Davide Costa

## 7.30 TcpDaemon.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * Copyright (C) 2024 Davide Costa
00003  *
00004  * This file is part of RMW desert.
00005  *
00006  * RMW desert is free software: you can redistribute it and/or modify it
00007  * under the terms of the GNU General Public License as published by the
00008  * Free Software Foundation, either version 3 of the License, or any
00009  * later version.
00010  *
00011  * RMW desert is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with RMW desert. If not, see <http://www.gnu.org/licenses/>.
00018  *****/
00019
00033 #ifndef TCP_DAEMON_H_
00034 #define TCP_DAEMON_H_
00035
00038 #include <queue>
00039 #include <vector>
00040 #include <stdint>
00041 #include <stdio>
00042 #include <cstring>
00043 #include <thread>
00044 #include <chrono>
00045
00046 #include <arpa/inet.h>
00047 #include <sys/socket.h>
00048 #include <sys/poll.h>
00049 #include <unistd.h>
00050
00051 #include "rmw/error_handling.h"
00052
00055 #define MAX_PACKET_LENGTH 512
00056
00057 #define ADDRESS "127.0.0.1"
00058
00059 #define START_MARKER 0b10011001
00060 #define END_MARKER 0b01010101
00061 #define BYTE_MASK 0b11111111
00062
00063 class TcpDaemon
00064 {
00065 public:
00066     TcpDaemon();
00067
00076     bool init(int port);
00085     static std::vector<uint8_t> read_packet();
00094     static void enqueue_packet(std::vector<uint8_t> packet);
00095
00096 private:
00097     static int _client_fd;
00098     static std::queue<std::vector<uint8_t>> _rx_packets;
00099     static std::queue<std::vector<uint8_t>> _tx_packets;
00100
00101     void socket_rx_communication();
00102     void socket_tx_communication();
00103
00104 };
00105
00106
00107 #endif

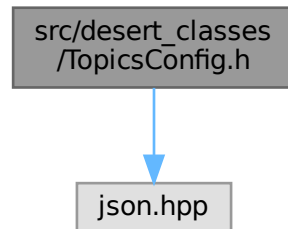
```

## 7.31 src/desert\_classes/TopicsConfig.h File Reference

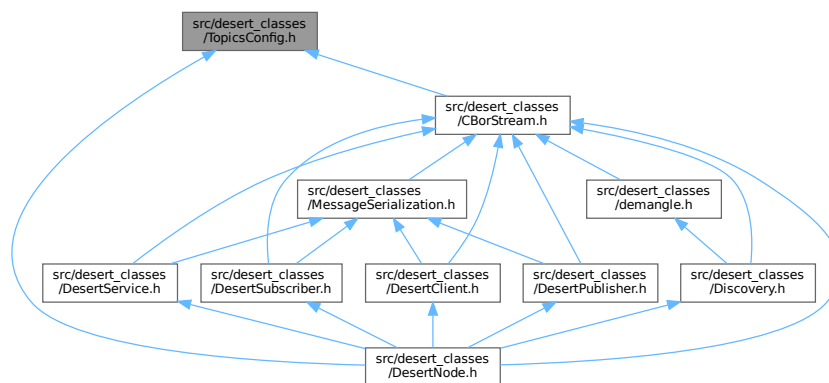
Class used to store configurations.

```
#include "json.hpp"
```

Include dependency graph for TopicsConfig.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [TopicsConfig](#)

### 7.31.1 Detailed Description

Class used to store configurations.

In order to prevent sending a string containing the topic name for each packet, a configuration file is used to associate each topic string to an integer that uses much less bandwidth and blocks topics that are not in this list.

## Author

Prof. Davide Costa

## 7.32 TopicsConfig.h

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * Copyright (C) 2024 Davide Costa
00003  *
00004  * This file is part of RMW desert.
00005  *
00006  * RMW desert is free software: you can redistribute it and/or modify it
00007  * under the terms of the GNU General Public License as published by the
00008  * Free Software Foundation, either version 3 of the License, or any
00009  * later version.
00010  *
00011  * RMW desert is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with RMW desert. If not, see <http://www.gnu.org/licenses/>.
00018  *****/
00019
00032 #ifndef TOPICS_CONFIG_H_
00033 #define TOPICS_CONFIG_H_
00034
00037 #include <map>
00038 #include <string>
00039 #include <cstdint>
00040 #include <fstream>
00041
00044 #include "json.hpp"
00045
00046 using namespace nlohmann::json_abi_v3_l1_3;
00047
00048 class TopicsConfig
00049 {
00050 public:
00051
00058 static void load_configuration();
00067 static uint8_t get_topic_identifier(std::string name);
00076 static std::string get_identifier_topic(uint8_t identifier);
00077
00078 private:
00079 static std::map<std::string, uint8_t> _topics_list;
00080 static std::map<uint8_t, std::string> _identifiers_list;
00081 };
00082
00083 #endif // MACROS_HPP_

```





# Index

- add\_client
  - DesertNode, [31](#)
- add\_publisher
  - DesertNode, [31](#)
- add\_service
  - DesertNode, [31](#)
- add\_subscriber
  - DesertNode, [32](#)
- assign\_string
  - CStringHelper, [10](#)
- assign\_u16string
  - CStringHelper, [10](#)
- assign\_vector\_string
  - CStringHelper, [10](#)
- assign\_vector\_string\_to\_sequence
  - CStringHelper, [11](#)
- assign\_vector\_u16string
  - CStringHelper, [11](#)
- assign\_vector\_u16string\_to\_sequence
  - CStringHelper, [11](#)
- cbor::RxStream, [41](#)
  - clear\_buffer, [42](#)
  - data\_available, [43](#)
  - deserialize\_integer, [43](#)
  - deserialize\_sequence, [43](#)
  - get\_identifier, [43](#)
  - get\_name, [44](#)
  - get\_type, [44](#)
  - interpret\_packets, [44](#)
  - operator>>, [44–48](#)
  - push\_packet, [48](#)
  - RxStream, [42](#)
- cbor::TxStream, [51](#)
  - end\_transmission, [52](#)
  - operator<<, [53–55](#), [57](#)
  - serialize\_sequence, [57](#)
  - start\_transmission, [58](#)
  - TxStream, [52](#)
- CircularQueue< T, MaxLen, Container >, [25](#)
- clear\_buffer
  - cbor::RxStream, [42](#)
- convert\_sequence\_to\_std\_vector\_string
  - CStringHelper, [12](#)
- convert\_sequence\_to\_std\_vector\_u16string
  - CStringHelper, [12](#)
- convert\_to\_std\_string
  - CStringHelper, [12](#)
- convert\_to\_std\_u16string
  - CStringHelper, [13](#)
- convert\_to\_std\_vector\_string
  - CStringHelper, [13](#)
- convert\_to\_std\_vector\_u16string
  - CStringHelper, [14](#)
- CStringHelper, [9](#)
  - assign\_string, [10](#)
  - assign\_u16string, [10](#)
  - assign\_vector\_string, [10](#)
  - assign\_vector\_string\_to\_sequence, [11](#)
  - assign\_vector\_u16string, [11](#)
  - assign\_vector\_u16string\_to\_sequence, [11](#)
  - convert\_sequence\_to\_std\_vector\_string, [12](#)
  - convert\_sequence\_to\_std\_vector\_u16string, [12](#)
  - convert\_to\_std\_string, [12](#)
  - convert\_to\_std\_u16string, [13](#)
  - convert\_to\_std\_vector\_string, [13](#)
  - convert\_to\_std\_vector\_u16string, [14](#)
- data\_available
  - cbor::RxStream, [43](#)
- demangle\_publisher\_from\_topic
  - Discovery, [15](#)
- demangle\_service\_from\_topic
  - Discovery, [15](#)
- demangle\_service\_reply\_from\_topic
  - Discovery, [16](#)
- demangle\_service\_request\_from\_topic
  - Discovery, [16](#)
- demangle\_subscriber\_from\_topic
  - Discovery, [17](#)
- demangle\_topic
  - Discovery, [17](#)
- deserialize
  - MessageSerialization, [21](#)
- deserialize\_field
  - MessageSerialization, [21](#)
- deserialize\_integer
  - cbor::RxStream, [43](#)
- deserialize\_sequence
  - cbor::RxStream, [43](#)
- DesertClient, [26](#)
  - DesertClient, [26](#)
  - get\_gid, [27](#)
  - get\_request\_type\_name, [27](#)
  - get\_response\_type\_name, [27](#)
  - get\_service\_name, [27](#)
  - has\_data, [27](#)
  - read\_response, [28](#)
  - send\_request, [28](#)
- DesertGuardCondition, [29](#)

- get\_has\_triggered, 29
  - has\_triggered, 29
  - trigger, 29
- DesertNode, 30
  - add\_client, 31
  - add\_publisher, 31
  - add\_service, 31
  - add\_subscriber, 32
  - DesertNode, 30
  - get\_gid, 32
  - remove\_client, 32
  - remove\_publisher, 32
  - remove\_service, 33
  - remove\_subscriber, 33
- DesertPublisher, 33
  - DesertPublisher, 34
  - get\_gid, 34
  - get\_topic\_name, 34
  - get\_type\_name, 34
  - push, 35
- DesertService, 35
  - DesertService, 36
  - get\_gid, 36
  - get\_request\_type\_name, 36
  - get\_response\_type\_name, 36
  - get\_service\_name, 37
  - has\_data, 37
  - read\_request, 37
  - send\_response, 38
- DesertSubscriber, 38
  - DesertSubscriber, 39
  - get\_gid, 39
  - get\_topic\_name, 39
  - get\_type\_name, 39
  - has\_data, 39
  - read\_data, 40
- DesertWaitset, 40
- Discovery, 14
  - demangle\_publisher\_from\_topic, 15
  - demangle\_service\_from\_topic, 15
  - demangle\_service\_reply\_from\_topic, 16
  - demangle\_service\_request\_from\_topic, 16
  - demangle\_subscriber\_from\_topic, 17
  - demangle\_topic, 17
  - discovery\_thread, 17
  - discovery\_thread\_start, 18
  - discovery\_thread\_stop, 18
  - identity\_demangle, 18
  - resolve\_prefix, 19
  - send\_discovery\_beacon, 19
  - send\_discovery\_request, 20
- discovery\_thread
  - Discovery, 17
- discovery\_thread\_start
  - Discovery, 18
- discovery\_thread\_stop
  - Discovery, 18
- end\_transmission
  - cbor::TxStream, 52
  - enqueue\_packet
    - TcpDaemon, 49
  - GenericCSequence< T >, 40
  - get\_gid
    - DesertClient, 27
    - DesertNode, 32
    - DesertPublisher, 34
    - DesertService, 36
    - DesertSubscriber, 39
  - get\_has\_triggered
    - DesertGuardCondition, 29
  - get\_identifier
    - cbor::RxStream, 43
  - get\_identifier\_topic
    - TopicsConfig, 50
  - get\_name
    - cbor::RxStream, 44
  - get\_request\_type\_name
    - DesertClient, 27
    - DesertService, 36
  - get\_response\_type\_name
    - DesertClient, 27
    - DesertService, 36
  - get\_service\_name
    - DesertClient, 27
    - DesertService, 37
  - get\_topic\_identifier
    - TopicsConfig, 50
  - get\_topic\_name
    - DesertPublisher, 34
    - DesertSubscriber, 39
  - get\_type
    - cbor::RxStream, 44
  - get\_type\_name
    - DesertPublisher, 34
    - DesertSubscriber, 39
  - has\_data
    - DesertClient, 27
    - DesertService, 37
    - DesertSubscriber, 39
  - has\_triggered
    - DesertGuardCondition, 29
  - identity\_demangle
    - Discovery, 18
  - init
    - TcpDaemon, 49
  - interpret\_packets
    - cbor::RxStream, 44
  - load\_configuration
    - TopicsConfig, 51
  - macros.h
    - SPECIALIZE\_GENERIC\_C\_SEQUENCE, 82
  - MessageSerialization, 20

- deserialize, 21
- deserialize\_field, 21
- serialize, 22
- serialize\_field, 22, 23
- operator<<
  - cbor::TxStream, 53–55, 57
- operator>>
  - cbor::RxStream, 44–48
- push
  - DesertPublisher, 35
- push\_packet
  - cbor::RxStream, 48
- read\_data
  - DesertSubscriber, 40
- read\_packet
  - TcpDaemon, 50
- read\_request
  - DesertService, 37
- read\_response
  - DesertClient, 28
- remove\_client
  - DesertNode, 32
- remove\_publisher
  - DesertNode, 32
- remove\_service
  - DesertNode, 33
- remove\_subscriber
  - DesertNode, 33
- resolve\_prefix
  - Discovery, 19
- rmw\_context\_impl\_s, 41
- RxStream
  - cbor::RxStream, 42
- send\_discovery\_beacon
  - Discovery, 19
- send\_discovery\_request
  - Discovery, 20
- send\_request
  - DesertClient, 28
- send\_response
  - DesertService, 38
- serialize
  - MessageSerialization, 22
- serialize\_field
  - MessageSerialization, 22, 23
- serialize\_sequence
  - cbor::TxStream, 57
- SPECIALIZE\_GENERIC\_C\_SEQUENCE
  - macros.h, 82
- src/desert\_classes/CBorStream.h, 59, 60
- src/desert\_classes/CStringHelper.h, 63, 64
- src/desert\_classes/demangle.h, 65, 67
- src/desert\_classes/DesertClient.h, 68, 69
- src/desert\_classes/DesertGuardCondition.h, 70
- src/desert\_classes/DesertNode.h, 71, 72
- src/desert\_classes/DesertPublisher.h, 73, 74
- src/desert\_classes/DesertService.h, 74, 76
- src/desert\_classes/DesertSubscriber.h, 77, 78
- src/desert\_classes/DesertWaitSet.h, 78, 79
- src/desert\_classes/Discovery.h, 79, 81
- src/desert\_classes/macros.h, 82, 83
- src/desert\_classes/MessageSerialization.h, 83, 85
- src/desert\_classes/rmw\_context\_impl\_s.h, 90, 91
- src/desert\_classes/TcpDaemon.h, 91, 93
- src/desert\_classes/TopicsConfig.h, 93, 95
- start\_transmission
  - cbor::TxStream, 58
- TcpDaemon, 49
  - enqueue\_packet, 49
  - init, 49
  - read\_packet, 50
- TopicsConfig, 50
  - get\_identifier\_topic, 50
  - get\_topic\_identifier, 50
  - load\_configuration, 51
- trigger
  - DesertGuardCondition, 29
- TxStream
  - cbor::TxStream, 52