RMW desert

1.0

Generated by Doxygen 1.9.8

Chapter 1

Namespace Index

1.1 Namespace List

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

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

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

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)

  Assing 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)

  Assing 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)

  Assing to a rosidl_runtime_c__String__Sequence the value contained in a vector of std::string.
- void assign_u16string (std::u16string str, void ∗field)

  Assing 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)

  Assing 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)

  Assing 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

str	C++ style string containing data
field	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

str	C++ style u16string containing data
field	Pointer containing the destination of the u16string

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

cpp_string_vector	C++ style vector of string containing data
str_array	Pointer containing the destination of the string sequence
size	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

cpp_string_vector	C++ style vector of string containing data
str_seq	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

cpp_string_vector	C++ style vector of u16strings containing data
str_array	Pointer containing the destination of the u16string sequence
size	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 )
```

Assing 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

cpp_string_vector	C++ style vector of u16strings containing data
str_seq	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

str_seq	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

str_seq	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

str	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

str	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

str_array	Pointer to the first original C-style string
size	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

str_array	Pointer to the first original C-style u16string
size	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

topic_name	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

topic_name	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

topic_name	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

topic_name	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

topic_name	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

topic_name	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

impl	The middleware comtext 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

impl	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

impl	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

name	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

name	Mangled topic name
prefix	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

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

Parameters

entity_gid	The global identifier of the entity
topic_name	The topic name
type_name	The topic type
disconnect	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

stream	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 an 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

msg	Pointer to the first byte of the message in memory
casted_members	Pointer to the member containing type support informations
stream	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

member	Pointer to the member containing type support informations
field	Pointer to the destination memory address of the elementary data
stream	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

member	Pointer to the member containing type support informations
field	Pointer to the destination memory address of the elementary data
stream	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

msg	Pointer to the first byte of the message in memory
casted_members	Pointer to the member containing type support informations
stream	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

member	Pointer to the member containing type support informations
field	Pointer to the origin memory address of the elementary data
stream	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

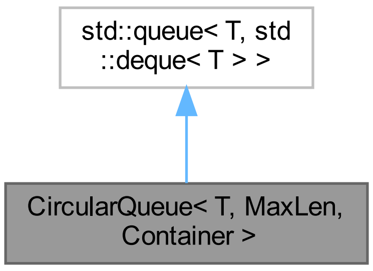
member	Pointer to the member containing type support informations
field	Pointer to the origin memory address of the elementary data
stream	The stream used to send data

Chapter 6

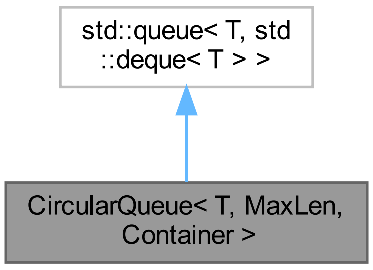
Class Documentation

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

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


std::queue< T, std ::deque< T > >
CircularQueue< T, MaxLen, Container >

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


std::queue< T, std ::deque< T > >
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 ()

  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.

6.2.1 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

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

6.2.2 Member Function Documentation

6.2.2.1 get_gid()

```
rmw_gid_t DesertClient::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

6.2.2.2 get_request_type_name()

```
std::string DesertClient::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

6.2.2.3 get_response_type_name()

```
std::string DesertClient::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

6.2.2.4 get_service_name()

```
std::string DesertClient::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

6.2.2.5 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

res	Pointer to the memory location used to store the reading
req_header	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

req	Pointer to the request to send
sequence←_id	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

name	Name of the node
namespace↩ _	Namespace of the node
gid	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

cli	Pointer to a DesertClient 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

pub	Pointer to a DesertPublisher 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

ser	Pointer to a DesertService 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

sub	Pointer to a DesertSubscriber 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

cli	Pointer to a DesertClient 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

pub	Pointer to a DesertPublisher 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

ser	Pointer to a DesertService 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

sub	Pointer to a DesertSubscriber 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

topic_name	Name of the topic used to push the messages
type_supports	Pointer to the message data structure coming from the ROS upper layers
gid	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

msg	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

service_name	Name of the service to receive requests and send responses
type_supports	Pointer to the message data structure coming from the ROS upper layers
gid	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

req	Pointer to the memory location used to store the request
req_header	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

res	Pointer to the response to send
req_header	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 ()

    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.

6.7.1 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

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

6.7.2 Member Function Documentation

6.7.2.1 get_gid()

```
rmw_gid_t DesertSubscriber::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

6.7.2.2 get_topic_name()

```
std::string DesertSubscriber::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

6.7.2.3 get_type_name()

```
std::string DesertSubscriber::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

6.7.2.4 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

msg	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

stream_type	Type of the object using the current instance
stream_name	Name of the topic or the service to which the communication belongs
stream_identifier	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

sequence←_id	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

n	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

items	Pointer to the first element
size	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

b	Field to decode

6.11.2.10 operator>>() [2/16]

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

Decode char.

Parameters

n	Field to decode

6.11.2.11 operator>>() [3/16]

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

Decode double.

Parameters

d	Field to decode

6.11.2.12 operator>>() [4/16]

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

Decode float.

Parameters

f	Field to decode

6.11.2.13 operator>>() [5/16]

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

Decode int16.

Parameters

n	Field to decode

6.11.2.14 operator>>() [6/16]

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

Decode int32.

Parameters

n	Field to decode

6.11.2.15 operator>>() [7/16]

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

Decode int64.

Parameters

n	Field to decode

6.11.2.16 operator>>() [8/16]

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

Decode int8.

Parameters

n	Field to decode

6.11.2.17 operator>>() [9/16]

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

Decode string.

Parameters

s	Field to decode

6.11.2.18 operator>>() [10/16]

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

Decode u16string.

Parameters

s	Field to decode

6.11.2.19 operator>>() [11/16]

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

Decode bool vector.

Parameters

v	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

v	Field to decode

6.11.2.21 operator>>() [13/16]

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

Decode uint16.

Parameters

n	Field to decode

6.11.2.22 operator>>() [14/16]

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

Decode uint32.

Parameters

n	Field to decode

6.11.2.23 operator>>() [15/16]

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

Decode uint64.

Parameters

n	Field to decode

6.11.2.24 operator>>() [16/16]

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

Decode uint8.

Parameters

n	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

packet	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

packet	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 estabilish a connection to the DESERT stack through a TCP socket.

Parameters

port	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

stream_type	Type of the object using the current instance
stream_name	Name of the topic or the service to which the communication belongs
stream_identifier	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

b	Field to encode

6.14.2.3 operator<<() [2/16]

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

Encode char.

Parameters

n	Field to encode

6.14.2.4 operator<<() [3/16]

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

Encode double.

Parameters

d	Field to encode

6.14.2.5 operator<<() [4/16]

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

Encode float.

Parameters

f	Field to encode

6.14.2.6 operator<<() [5/16]

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

Encode int16.

Parameters

n	Field to encode

6.14.2.7 operator<<() [6/16]

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

Encode int32.

Parameters

n	Field to encode

6.14.2.8 operator<<() [7/16]

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

Encode int64.

Parameters

n	Field to encode

6.14.2.9 operator<<() [8/16]

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

Encode int8.

Parameters

n	Field to encode

6.14.2.10 operator<<() [9/16]

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

Encode string.

Parameters

s	Field to encode

6.14.2.11 operator<<() [10/16]

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

Encode u16string.

Parameters

s	Field to encode

6.14.2.12 operator<<() [11/16]

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

Encode bool vector.

Parameters

v	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

v	Field to encode

6.14.2.14 operator<<() [13/16]

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

Encode uint16.

Parameters

n	Field to encode

6.14.2.15 operator<<() [14/16]

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

Encode uint32.

Parameters

n	Field to encode

6.14.2.16 operator<<() [15/16]

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

Encode uint64.

Parameters

n	Field to encode

6.14.2.17 operator<<() [16/16]

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

Encode uint8.

Parameters

n	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

items	Pointer to the first element
size	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

sequence←_id	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

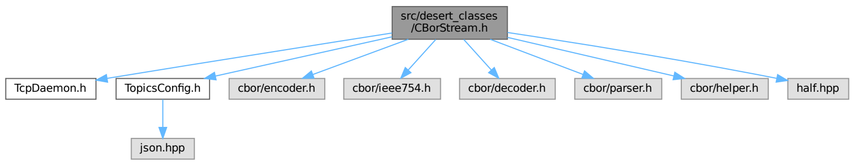
File Documentation

7.1 src/desert_classes/CBorStream.h File Reference

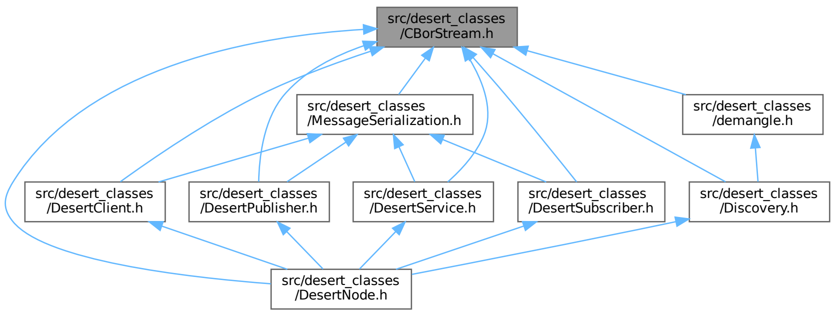
Classes used to convert data types into a CBOR encoded stream.

```
#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:



src/desert_classes /CBorStream.h
src/desert_classes /MessageSerialization.h
src/desert_classes /demangle.h
src/desert_classes /DesertClient.h
src/desert_classes /DesertPublisher.h
src/desert_classes /DesertService.h
src/desert_classes /DesertSubscriber.h
src/desert_classes /Discovery.h
src/desert_classes /DesertNode.h

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.

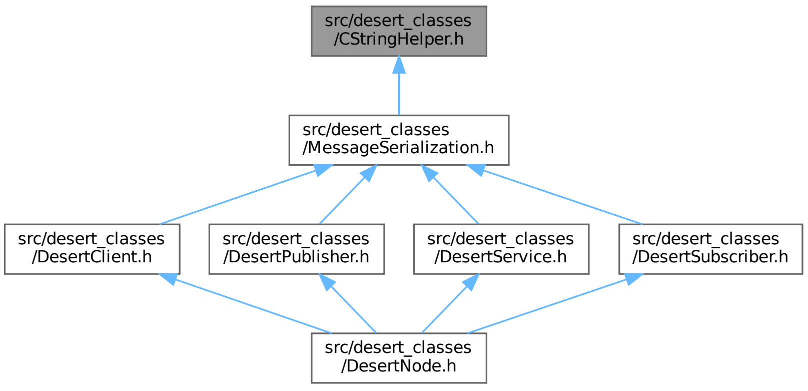
```
/*****************************************************************************
 * Copyright (C) 2024 Davide Costa                                          *
 *                                                                          *
 * This file is part of RMW desert.                                         *
 *                                                                          *
 *   RMW desert is free software: you can redistribute it and/or modify it  *
 *   under the terms of the GNU General Public License as published by the  *
 *   Free Software Foundation, either version 3 of the License, or any      *
 *   later version.                                                         *
 *                                                                          *
 *   RMW desert is distributed in the hope that it will be useful,          *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of         *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          *
 *   GNU General Public License for more details.                           *
 *                                                                          *
 *   You should have received a copy of the GNU General Public License      *
 *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.    *
 ****************************************************************************/

#ifndef CBORSTREAM_H_
#define CBORSTREAM_H_

#include "TcpDaemon.h"
#include "TopicsConfig.h"

#include <map>
#include <queue>
#include <utility>
#include <vector>
#include <string>
#include <locale>
#include <codecvt>
#include <cstdint>
```

```
#include <cstdio>
#include <mutex>

#include "cbor/encoder.h"
#include "cbor/ieee754.h"
#include "cbor/decoder.h"
#include "cbor/parser.h"
#include "cbor/helper.h"

#include "half.hpp"

#define PUBLISHER_TYPE  0
#define SUBSCRIBER_TYPE 1
#define CLIENT_TYPE     2
#define SERVICE_TYPE    3

#define MAX_BUFFER_CAPACITY 100

template <typename T, int MaxLen, typename Container=std::deque<T>>
class CircularQueue : public std::queue<T, Container> {
  public:
    void push(const T& value)
    {
        if (this->size() == MaxLen)
        {
           this->c.pop_front();
        }
        std::queue<T, Container>::push(value);
    }
};

namespace cbor
{

class TxStream
{
  public:
    TxStream(uint8_t stream_type, std::string stream_name, uint8_t stream_identifier);

    void start_transmission(uint64_t sequence_id);
    void start_transmission();
    void end_transmission();

    TxStream & operator<<(const uint64_t n);
    TxStream & operator<<(const uint32_t n);
    TxStream & operator<<(const uint16_t n);
    TxStream & operator<<(const uint8_t n);
    TxStream & operator<<(const int64_t n);
    TxStream & operator<<(const int32_t n);
    TxStream & operator<<(const int16_t n);
    TxStream & operator<<(const int8_t n);
    TxStream & operator<<(const char n);
    TxStream & operator<<(const float f);
    TxStream & operator<<(const double d);
    TxStream & operator<<(const std::string s);
    TxStream & operator<<(const std::u16string s);
    TxStream & operator<<(const bool b);

    template<typename T>
    inline TxStream & operator<<(const std::vector<T> v)
    {
      *this << static_cast<const uint32_t>(v.size());
      return serialize_sequence(v.data(), v.size());
    }

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

    template<typename T>
    inline TxStream & serialize_sequence(const T * items, size_t size)
    {
      for (size_t i = 0; i < size; ++i)
      {
        *this << items[i];
      }
      return *this;
    }

  private:
    uint8_t _stream_type;
    std::string _stream_name;
    uint8_t _stream_identifier;

    bool _overflow;
    uint8_t *  _packet;
    cbor_writer_t *  _writer;

    void new_packet();
    void handle_overrun(cbor_error_t result);

    std::string toUTF8(const std::u16string source);

};

class RxStream
{
  public:
    RxStream(uint8_t stream_type, std::string stream_name, uint8_t stream_identifier);

    ~RxStream();

    bool data_available(int64_t sequence_id = 0);

    void clear_buffer();

    RxStream & operator»(uint64_t & n);
    RxStream & operator»(uint32_t & n);
    RxStream & operator»(uint16_t & n);
    RxStream & operator»(uint8_t & n);
    RxStream & operator»(int64_t & n);
    RxStream & operator»(int32_t & n);
    RxStream & operator»(int16_t & n);
    RxStream & operator»(int8_t & n);

    template<typename T>
    RxStream & deserialize_integer(T & n);

    RxStream & operator»(char & n);
    RxStream & operator»(float & f);
    RxStream & operator»(double & d);
    RxStream & operator»(std::string & s);
    RxStream & operator»(std::u16string & s);
    RxStream & operator»(bool & b);

    template<typename T>
    inline RxStream & operator»(std::vector<T> & v)
    {
      uint32_t size;
      *this » size;
      v.resize(size);

      return deserialize_sequence(v.data(), size);
    }

    RxStream & operator»(std::vector<bool> & v);

    template<typename T>
    inline RxStream & deserialize_sequence(T * items, size_t size)
    {
      for (size_t i = 0; i < size; ++i)
      {
        *this » items[i];
      }
      return *this;
    }


    uint8_t get_type() const;
    std::string get_name() const;
    uint8_t get_identifier() const;

    void push_packet(std::vector<std::pair<void *, int» packet);

    static void interpret_packets();

  private:
    uint8_t _stream_type;
    std::string _stream_name;
    uint8_t _stream_identifier;

    size_t _buffered_iterator;

    // packets: <packet <field, field_type»
    std::vector<std::pair<void *, int» _buffered_packet;
    CircularQueue<std::vector<std::pair<void *, int>>, MAX_BUFFER_CAPACITY> _received_packets;

    static const std::map<int, int> _stream_type_match_map;
    static std::vector<RxStream *> _listening_streams;

    union _cbor_value {
    int8_t i8;
    int16_t i16;
    int32_t i32;
    int64_t i64;
    float f32;
    double f64;
    uint8_t *bin;
    char *str;
    uint8_t str_copy[128];
    };

    static std::mutex _rx_mutex;

    static std::pair<void *, int> interpret_field(cbor_item_t * items, size_t i, union _cbor_value &
val);
    std::u16string toUTF16(const std::string source);
};

}  // namespace cbor


#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:


src/desert_classes /CStringHelper.h
src/desert_classes /MessageSerialization.h
src/desert_classes /DesertClient.h
src/desert_classes /DesertPublisher.h
src/desert_classes /DesertService.h
src/desert_classes /DesertSubscriber.h
src/desert_classes /DesertNode.h

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)

    Assing 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)

    Assing 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)

    Assing 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)

    Assing 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)

    Assing 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)

    Assing 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.

```
/*****************************************************************************
 * Copyright (C) 2024 Davide Costa                                           *
 *                                                                           *
 * This file is part of RMW desert.                                          *
 *                                                                           *
 *   RMW desert is free software: you can redistribute it and/or modify it  *
 *   under the terms of the GNU General Public License as published by the  *
 *   Free Software Foundation, either version 3 of the License, or any      *
 *   later version.                                                          *
 *                                                                           *
 *   RMW desert is distributed in the hope that it will be useful,          *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of         *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          *
 *   GNU General Public License for more details.                           *
 *                                                                           *
 *   You should have received a copy of the GNU General Public License      *
 *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.    *
 *****************************************************************************/

#ifndef CSTRING_HELPER_H_
#define CSTRING_HELPER_H_

#include "rosidl_runtime_c/u16string.h"
```

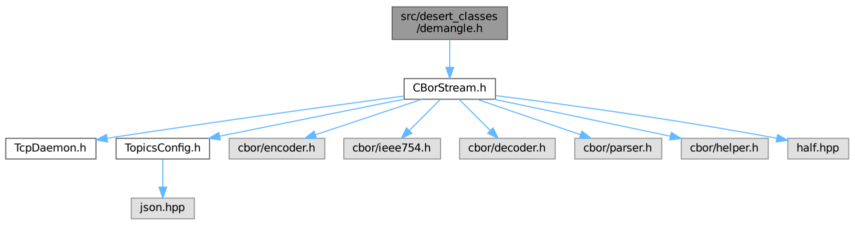
```
#include "rosidl_runtime_c/string.h"
#include "rosidl_runtime_c/u16string_functions.h"
#include "rosidl_runtime_c/string_functions.h"

#include <stdexcept>
#include <vector>
#include <string>

namespace CStringHelper
{
  std::string convert_to_std_string(void * str);
  std::vector<std::string> convert_to_std_vector_string(void * str_array, size_t size);
  std::vector<std::string> convert_sequence_to_std_vector_string(void * str_seq);

  std::u16string convert_to_std_u16string(void * str);
  std::vector<std::u16string> convert_to_std_vector_u16string(void * str_array, size_t size);
  std::vector<std::u16string> convert_sequence_to_std_vector_u16string(void * str_seq);

  void assign_string(std::string str, void * field);
  void assign_vector_string(std::vector<std::string> cpp_string_vector, void * str_array, size_t
size);
  void assign_vector_string_to_sequence(std::vector<std::string> cpp_string_vector, void * str_seq);

  void assign_u16string(std::u16string str, void * field);
  void assign_vector_u16string(std::vector<std::u16string> cpp_string_vector, void * str_array, size_t
size);
  void assign_vector_u16string_to_sequence(std::vector<std::u16string> cpp_string_vector, void *
str_seq);
}


#endif
```

7.5 src/desert_classes/demangle.h File Reference

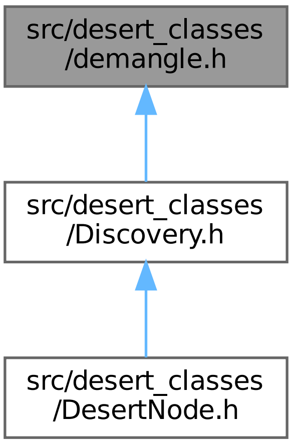
Functions used to demangle topic names during discovery operations.

`#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.

```
/*****************************************************************************
 * Copyright (C) 2024 Davide Costa                                           *
 *                                                                           *
 * This file is part of RMW desert.                                          *
 *                                                                           *
 *   RMW desert is free software: you can redistribute it and/or modify it   *
 *   under the terms of the GNU General Public License as published by the   *
 *   Free Software Foundation, either version 3 of the License, or any       *
 *   later version.                                                          *
 *                                                                           *
 *   RMW desert is distributed in the hope that it will be useful,           *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of          *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the           *
 *   GNU General Public License for more details.                            *
 *                                                                           *
 *   You should have received a copy of the GNU General Public License       *
 *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.     *
 ****************************************************************************/

#ifndef DEMANGLE_H_
#define DEMANGLE_H_

#include <algorithm>
#include <string>
#include <vector>

#include "rcpputils/find_and_replace.hpp"
#include "rcutils/logging_macros.h"
#include "rcutils/types.h"

#include "CBorStream.h"

namespace Discovery
{

  char * integer_to_string(int x);

  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);

  std::string resolve_prefix(const std::string & name, const std::string & prefix);

  std::string demangle_publisher_from_topic(const std::string & topic_name);

  std::string demangle_subscriber_from_topic(const std::string & topic_name);

```

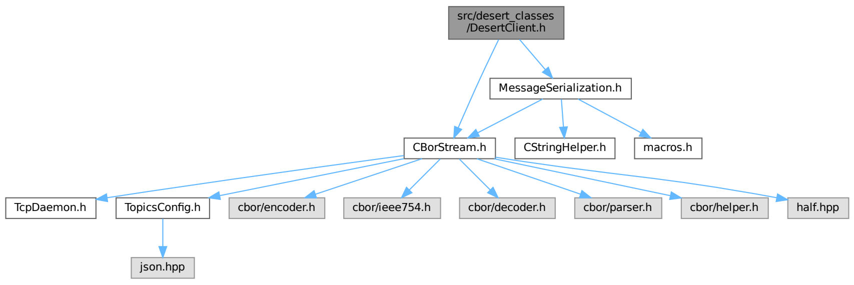
```
    std::string demangle_topic(const std::string & topic_name);

    std::string demangle_service_request_from_topic(const std::string & topic_name);

    std::string demangle_service_reply_from_topic(const std::string & topic_name);

    std::string demangle_service_from_topic(const std::string & topic_name);

    std::string identity_demangle(const std::string & name);

    using DemangleFunction = std::string (*)(const std::string &);

  }

  #endif  // DEMANGLE_H_
```

7.7 src/desert_classes/DesertClient.h File Reference

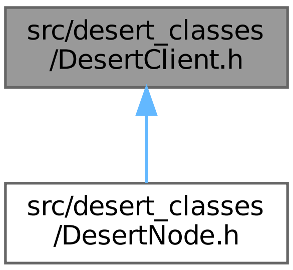
Implementation of the Client structure for DESERT.

```
#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.

```
/****************************************************************************
 * Copyright (C) 2024 Davide Costa                                          *
 *                                                                          *
 * This file is part of RMW desert.                                         *
 *                                                                          *
 *   RMW desert is free software: you can redistribute it and/or modify it  *
 *   under the terms of the GNU General Public License as published by the  *
 *   Free Software Foundation, either version 3 of the License, or any      *
 *   later version.                                                         *
 *                                                                          *
 *   RMW desert is distributed in the hope that it will be useful,          *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of         *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          *
 *   GNU General Public License for more details.                           *
 *                                                                          *
 *   You should have received a copy of the GNU General Public License      *
 *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.    *
 ****************************************************************************/

#ifndef DESERT_CLIENT_H_
#define DESERT_CLIENT_H_

#include "rosidl_typesupport_introspection_cpp/identifier.hpp"
#include "rosidl_typesupport_introspection_c/identifier.h"
#include "rosidl_typesupport_introspection_cpp/message_introspection.hpp"
#include "rosidl_typesupport_introspection_c/message_introspection.h"
#include "rosidl_typesupport_introspection_cpp/service_introspection.hpp"
#include "rosidl_typesupport_introspection_c/service_introspection.h"
#include "rosidl_typesupport_introspection_cpp/field_types.hpp"
#include "rosidl_typesupport_introspection_c/field_types.h"

#include "rosidl_runtime_c/service_type_support_struct.h"

#include "rmw/types.h"

#include <vector>
#include <string>
#include <regex>

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

class DesertClient
{
  public:
    DesertClient(std::string service_name, const rosidl_service_type_support_t * type_supports,
rmw_gid_t gid);

    bool has_data();
    void send_request(const void * req, int64_t * sequence_id);
    void read_response(void * res, rmw_service_info_t * req_header);

    rmw_gid_t get_gid();
    std::string get_service_name();
    std::string get_request_type_name();
    std::string get_response_type_name();


  private:
    uint8_t _id;
    rmw_gid_t _gid;
    std::string _name;
    cbor::TxStream _request_data_stream;
    cbor::RxStream _response_data_stream;

    int64_t _sequence_id;

    int _c_cpp_identifier;
    const void * _service;

    const void * get_service(const rosidl_service_type_support_t * service_type_support);
    const rosidl_service_type_support_t * get_service_type_support(const rosidl_service_type_support_t
* type_supports);

};

#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.

```
/****************************************************************************
 * Copyright (C) 2024 Davide Costa                                          *
 *                                                                          *
 * This file is part of RMW desert.                                         *
 *                                                                          *
 *   RMW desert is free software: you can redistribute it and/or modify it  *
 *   under the terms of the GNU General Public License as published by the  *
 *   Free Software Foundation, either version 3 of the License, or any      *
 *   later version.                                                         *
 *                                                                          *
 *   RMW desert is distributed in the hope that it will be useful,          *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of         *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          *
 *   GNU General Public License for more details.                           *
 *                                                                          *
 *   You should have received a copy of the GNU General Public License      *
 *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.    *
 ****************************************************************************/

#ifndef DESERT_GUARD_CONDITION_H_
#define DESERT_GUARD_CONDITION_H_

#include <array>
```

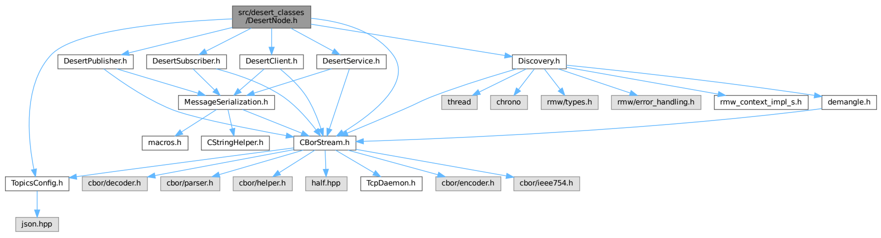
```
#include <atomic>
#include <cassert>
#include <condition_variable>
#include <mutex>
#include <utility>

#include "rcpputils/thread_safety_annotations.hpp"

class DesertGuardCondition
{
  public:
    DesertGuardCondition();

    void trigger();

    bool has_triggered();

    bool get_has_triggered();

  private:
    std::atomic_bool _has_triggered;
};

#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 af 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.

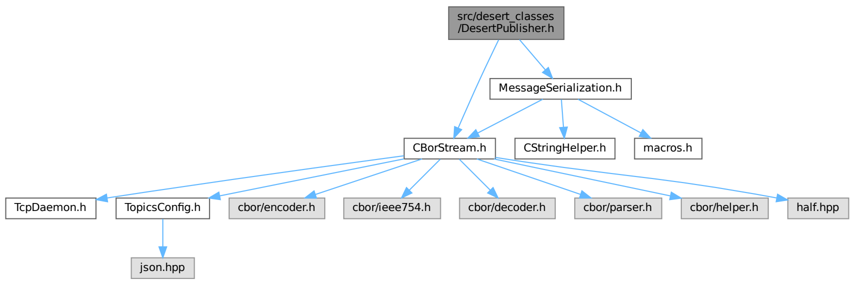
```
/****************************************************************************
 * Copyright (C) 2024 Davide Costa                                          *
 *                                                                          *
 * This file is part of RMW desert.                                         *
 *                                                                          *
 *   RMW desert is free software: you can redistribute it and/or modify it  *
 *   under the terms of the GNU General Public License as published by the  *
 *   Free Software Foundation, either version 3 of the License, or any      *
 *   later version.                                                         *
 *                                                                          *
 *   RMW desert is distributed in the hope that it will be useful,          *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of         *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          *
 *   GNU General Public License for more details.                           *
 *                                                                          *
 *   You should have received a copy of the GNU General Public License      *
 *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.    *
 ****************************************************************************/

#ifndef DESERT_NODE_H_
#define DESERT_NODE_H_

#include "rmw/rmw.h"
#include "rmw/types.h"

#include <vector>
#include <string>

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

class DesertNode
{
  public:
    DesertNode(std::string name, std::string namespace_, rmw_gid_t gid);
    ~DesertNode();

    void add_publisher(DesertPublisher * pub);

    void add_subscriber(DesertSubscriber * sub);

    void add_client(DesertClient * cli);

    void add_service(DesertService * ser);

    void remove_publisher(DesertPublisher * pub);

    void remove_subscriber(DesertSubscriber * sub);

    void remove_client(DesertClient * cli);

    void remove_service(DesertService * ser);

    rmw_gid_t get_gid();

  private:
    rmw_gid_t _gid;
    std::string _name;
    std::string _namespace;
    cbor::TxStream _discovery_beacon_data_stream;
    cbor::RxStream _discovery_request_data_stream;

    std::vector<DesertPublisher *> _publishers;
    std::vector<DesertSubscriber *> _subscribers;
    std::vector<DesertClient *> _clients;
    std::vector<DesertService *> _services;

    void publish_all_beacons();

    bool _discovery_done;
    std::thread _discovery_request_thread;

    void _discovery_request();

};

#endif
```

7.13 src/desert_classes/DesertPublisher.h File Reference

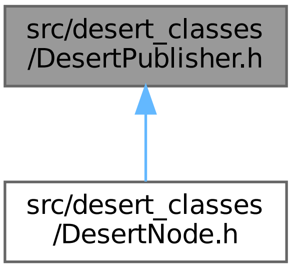
Implementation of the Publisher structure for DESERT.

```
#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.

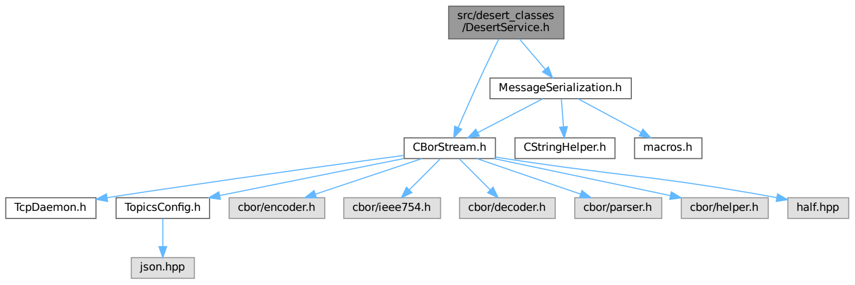
```
/****************************************************************************
 * Copyright (C) 2024 Davide Costa                                          *
 *                                                                          *
 * This file is part of RMW desert.                                         *
 *                                                                          *
 *   RMW desert is free software: you can redistribute it and/or modify it  *
 *   under the terms of the GNU General Public License as published by the  *
 *   Free Software Foundation, either version 3 of the License, or any      *
 *   later version.                                                         *
 *                                                                          *
 *   RMW desert is distributed in the hope that it will be useful,          *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of         *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          *
 *   GNU General Public License for more details.                           *
 *                                                                          *
 *   You should have received a copy of the GNU General Public License      *
 *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.    *
 ****************************************************************************/

#ifndef DESERT_PUBLISHER_H_
#define DESERT_PUBLISHER_H_

#include "rosidl_typesupport_introspection_cpp/identifier.hpp"
#include "rosidl_typesupport_introspection_c/identifier.h"
#include "rosidl_typesupport_introspection_cpp/message_introspection.hpp"
#include "rosidl_typesupport_introspection_c/message_introspection.h"
#include "rosidl_typesupport_introspection_cpp/service_introspection.hpp"
#include "rosidl_typesupport_introspection_c/service_introspection.h"
#include "rosidl_typesupport_introspection_cpp/field_types.hpp"
#include "rosidl_typesupport_introspection_c/field_types.h"

#include "rosidl_runtime_c/message_type_support_struct.h"

#include "rmw/types.h"

#include <vector>
#include <string>
#include <regex>

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

class DesertPublisher
{
  public:
    DesertPublisher(std::string topic_name, const rosidl_message_type_support_t * type_supports,
rmw_gid_t gid);

    void push(const void * msg);

    rmw_gid_t get_gid();
    std::string get_topic_name();
    std::string get_type_name();


  private:
    uint8_t _id;
    rmw_gid_t _gid;
    std::string _name;
    cbor::TxStream _data_stream;

    int _c_cpp_identifier;
    const void * _members;

    const void * get_members(const rosidl_message_type_support_t * type_support);
    const rosidl_message_type_support_t * get_type_support(const rosidl_message_type_support_t *
type_supports);

};

#endif
```

7.15 src/desert_classes/DesertService.h File Reference

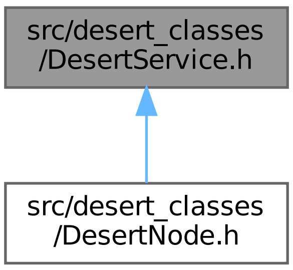
Implementation of the Service structure for DESERT.

```
#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.

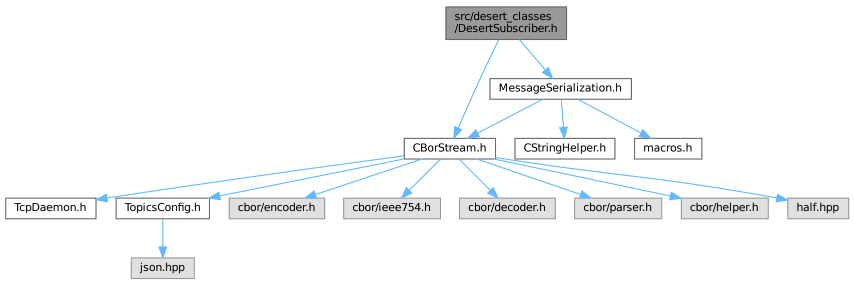
```
/****************************************************************************
 * Copyright (C) 2024 Davide Costa                                          *
 *                                                                          *
 * This file is part of RMW desert.                                         *
 *                                                                          *
 *   RMW desert is free software: you can redistribute it and/or modify it  *
 *   under the terms of the GNU General Public License as published by the  *
 *   Free Software Foundation, either version 3 of the License, or any      *
 *   later version.                                                         *
 *                                                                          *
 *   RMW desert is distributed in the hope that it will be useful,          *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of         *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          *
 *   GNU General Public License for more details.                           *
 *                                                                          *
 *   You should have received a copy of the GNU General Public License      *
 *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.    *
 ****************************************************************************/

#ifndef DESERT_SERVICE_H_
#define DESERT_SERVICE_H_

#include "rosidl_typesupport_introspection_cpp/identifier.hpp"
#include "rosidl_typesupport_introspection_c/identifier.h"
#include "rosidl_typesupport_introspection_cpp/message_introspection.hpp"
#include "rosidl_typesupport_introspection_c/message_introspection.h"
#include "rosidl_typesupport_introspection_cpp/service_introspection.hpp"
#include "rosidl_typesupport_introspection_c/service_introspection.h"
#include "rosidl_typesupport_introspection_cpp/field_types.hpp"
#include "rosidl_typesupport_introspection_c/field_types.h"

#include "rosidl_runtime_c/service_type_support_struct.h"

#include "rmw/types.h"

#include <vector>
#include <string>
#include <regex>

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

class DesertService
{
  public:
    DesertService(std::string service_name, const rosidl_service_type_support_t * type_supports, rmw_gid_t gid);

    bool has_data();
    void read_request(void * req, rmw_service_info_t * req_header);
    void send_response(void * res, rmw_request_id_t * req_header);

    rmw_gid_t get_gid();
    std::string get_service_name();
    std::string get_request_type_name();
    std::string get_response_type_name();


  private:
    uint8_t _id;
    rmw_gid_t _gid;
    std::string _name;
    cbor::RxStream _request_data_stream;
    cbor::TxStream _response_data_stream;

    int64_t _sequence_id;

    int _c_cpp_identifier;
    const void * _service;

    const void * get_service(const rosidl_service_type_support_t * service_type_support);
    const rosidl_service_type_support_t * get_service_type_support(const rosidl_service_type_support_t * type_supports);

};

#endif
```

7.17 src/desert_classes/DesertSubscriber.h File Reference

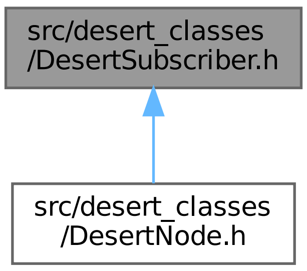
Implementation of the Subscriber structure for DESERT.

```
#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.

```
/****************************************************************************
 * Copyright (C) 2024 Davide Costa                                          *
 *                                                                          *
 * This file is part of RMW desert.                                         *
 *                                                                          *
 *   RMW desert is free software: you can redistribute it and/or modify it  *
 *   under the terms of the GNU General Public License as published by the  *
 *   Free Software Foundation, either version 3 of the License, or any      *
 *   later version.                                                         *
 *                                                                          *
 *   RMW desert is distributed in the hope that it will be useful,          *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of         *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          *
 *   GNU General Public License for more details.                           *
 *                                                                          *
 *   You should have received a copy of the GNU General Public License      *
 *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.    *
 ****************************************************************************/

#ifndef DESERT_SUBSCRIBER_H_
#define DESERT_SUBSCRIBER_H_

#include "rosidl_typesupport_introspection_cpp/identifier.hpp"
#include "rosidl_typesupport_introspection_c/identifier.h"
#include "rosidl_typesupport_introspection_cpp/message_introspection.hpp"
#include "rosidl_typesupport_introspection_c/message_introspection.h"
#include "rosidl_typesupport_introspection_cpp/service_introspection.hpp"
#include "rosidl_typesupport_introspection_c/service_introspection.h"
#include "rosidl_typesupport_introspection_cpp/field_types.hpp"
#include "rosidl_typesupport_introspection_c/field_types.h"

#include "rosidl_runtime_c/message_type_support_struct.h"

#include "rmw/types.h"

#include <vector>
#include <string>
#include <regex>

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

class DesertSubscriber
{
  public:
    DesertSubscriber(std::string topic_name, const rosidl_message_type_support_t * type_supports,
rmw_gid_t gid);

    bool has_data();
    void read_data(void * msg);

    rmw_gid_t get_gid();
    std::string get_topic_name();
    std::string get_type_name();

  private:
    uint8_t _id;
    rmw_gid_t _gid;
    std::string _name;
    cbor::RxStream _data_stream;

    int _c_cpp_identifier;
    const void * _members;

    const void * get_members(const rosidl_message_type_support_t * type_support);
    const rosidl_message_type_support_t * get_type_support(const rosidl_message_type_support_t *
type_supports);

};

#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.

```
/****************************************************************************
 * Copyright (C) 2024 Davide Costa                                          *
 *                                                                          *
 * This file is part of RMW desert.                                         *
 *                                                                          *
 *   RMW desert is free software: you can redistribute it and/or modify it  *
 *   under the terms of the GNU General Public License as published by the  *
 *   Free Software Foundation, either version 3 of the License, or any      *
 *   later version.                                                         *
 *                                                                          *
 *   RMW desert is distributed in the hope that it will be useful,          *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of         *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          *
 *   GNU General Public License for more details.                           *
 *                                                                          *
 *   You should have received a copy of the GNU General Public License      *
 *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.    *
 ****************************************************************************/

#ifndef DESERT_WAIT_SET_H_
#define DESERT_WAIT_SET_H_

class DesertWaitset
{
  public:
    DesertWaitset()
    {}

    std::mutex lock;
    bool inuse;
};

#endif
```

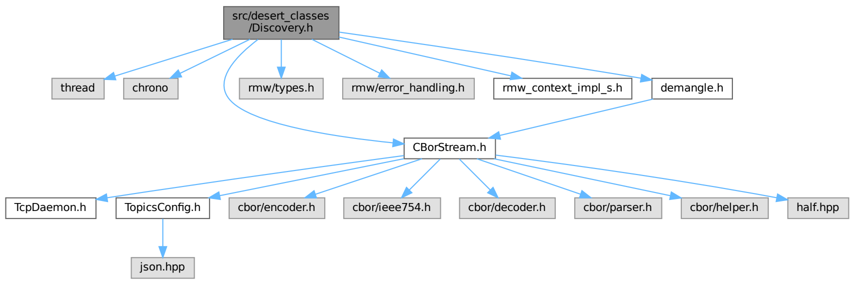
7.21 src/desert_classes/Discovery.h File Reference

Namespace used to provide discovery functionalities.

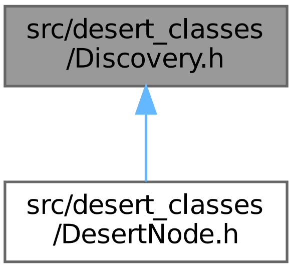
```
#include <thread>
#include <chrono>
#include "CBorStream.h"
#include "rmw/types.h"
#include "rmw/error_handling.h"
#include "rmw_context_impl_s.h"
```

```
#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

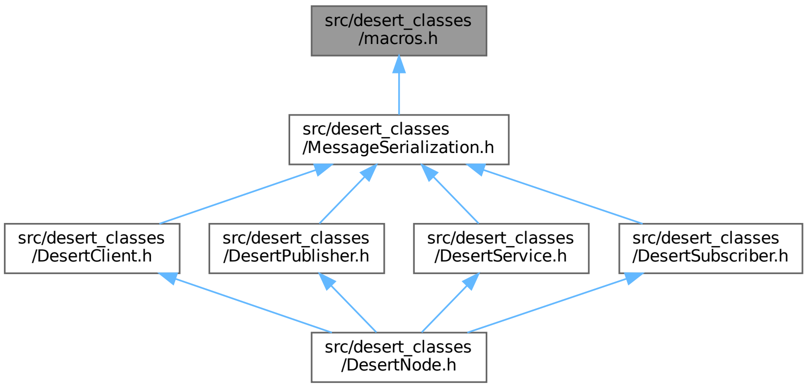
Go to the documentation of this file.

```
/****************************************************************************
 * Copyright (C) 2024 Davide Costa                                          *
 *                                                                          *
 * This file is part of RMW desert.                                         *
 *                                                                          *
 *   RMW desert is free software: you can redistribute it and/or modify it  *
 *   under the terms of the GNU General Public License as published by the  *
 *   Free Software Foundation, either version 3 of the License, or any      *
 *   later version.                                                         *
 *                                                                          *
 *   RMW desert is distributed in the hope that it will be useful,          *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of         *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          *
 *   GNU General Public License for more details.                           *
 *                                                                          *
 *   You should have received a copy of the GNU General Public License      *
 *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.    *
 ****************************************************************************/

#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"

#ifndef DISCOVERY_H_
#define DISCOVERY_H_

namespace Discovery
{

  void discovery_thread(rmw_context_impl_t * impl);

  rmw_ret_t discovery_thread_start(rmw_context_impl_t * impl);

  rmw_ret_t discovery_thread_stop(rmw_context_impl_t * impl);

  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);

  void send_discovery_request(cbor::TxStream stream);

}

#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.

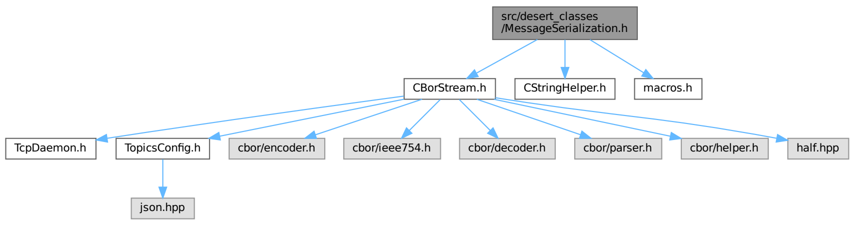
```
/*****************************************************************************
 * Copyright (C) 2024 Davide Costa                                           *
 *                                                                           *
 * This file is part of RMW desert.                                          *
 *                                                                           *
 *   RMW desert is free software: you can redistribute it and/or modify it  *
 *   under the terms of the GNU General Public License as published by the  *
 *   Free Software Foundation, either version 3 of the License, or any      *
 *   later version.                                                          *
 *                                                                           *
 *   RMW desert is distributed in the hope that it will be useful,          *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of         *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          *
 *   GNU General Public License for more details.                           *
 *                                                                           *
 *   You should have received a copy of the GNU General Public License      *
 *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.    *
 *****************************************************************************/

#ifndef MACROS_H_
#define MACROS_H_

#include "rosidl_runtime_c/primitives_sequence.h"
#include "rosidl_runtime_c/primitives_sequence_functions.h"

#define SPECIALIZE_GENERIC_C_SEQUENCE(C_NAME, C_TYPE) \
  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); \
    } \
  };

template<typename T>
struct GenericCSequence;

// multiple definitions of ambiguous primitive types
SPECIALIZE_GENERIC_C_SEQUENCE(bool, bool)
SPECIALIZE_GENERIC_C_SEQUENCE(byte, uint8_t)
SPECIALIZE_GENERIC_C_SEQUENCE(char, char)
SPECIALIZE_GENERIC_C_SEQUENCE(float32, float)
SPECIALIZE_GENERIC_C_SEQUENCE(float64, double)
SPECIALIZE_GENERIC_C_SEQUENCE(int8, int8_t)
SPECIALIZE_GENERIC_C_SEQUENCE(int16, int16_t)
SPECIALIZE_GENERIC_C_SEQUENCE(uint16, uint16_t)
SPECIALIZE_GENERIC_C_SEQUENCE(int32, int32_t)
SPECIALIZE_GENERIC_C_SEQUENCE(uint32, uint32_t)
SPECIALIZE_GENERIC_C_SEQUENCE(int64, int64_t)
SPECIALIZE_GENERIC_C_SEQUENCE(uint64, uint64_t)

#endif  // MACROS_HPP_
```

7.25 src/desert_classes/MessageSerialization.h File Reference

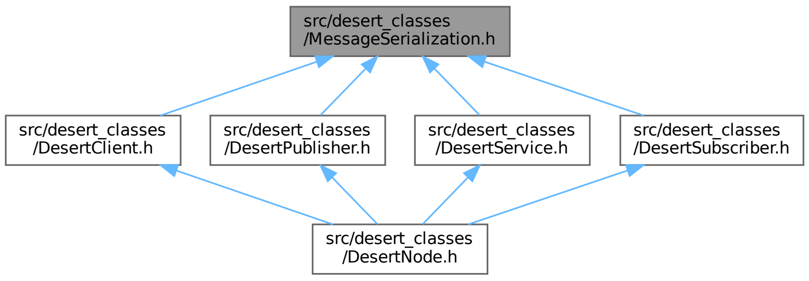
Namespace containing serialization functions.

```
#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__Service↩ Members
- #define **INTROSPECTION_CPP_SERVICE_MEMBERS** rosidl_typesupport_introspection_cpp::Service↩ Members

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 an 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"
```

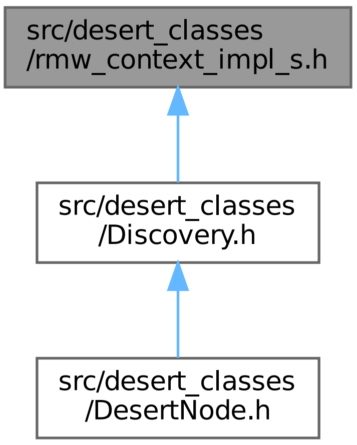
```
#include "CStringHelper.h"
#include "macros.h"

#include <vector>
#include <string>

#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

namespace MessageSerialization
{

  template<typename T>
  void serialize_field(const INTROSPECTION_CPP_MEMBER * member, void * field, cbor::TxStream & stream)
  {
    if (!member->is_array_)
    {
      stream « * static_cast<T *>(field);
    }
    else if (member->array_size_ && !member->is_upper_bound_)
    {
      stream.serialize_sequence(static_cast<T *>(field), member->array_size_);
    }
    else
    {
      std::vector<T> & data = *reinterpret_cast<std::vector<T> *>(field);
      stream « data;
    }
  }

  template<typename T>
  void serialize_field(const INTROSPECTION_C_MEMBER * member, void * field, cbor::TxStream & stream)
  {
    // String specific implementation
    if constexpr(std::is_same_v<T, std::string>)
    {
      if (!member->is_array_)
      {
        stream « CStringHelper::convert_to_std_string(field);
      }
      else if (member->array_size_ && !member->is_upper_bound_)
      {
        stream « CStringHelper::convert_to_std_vector_string(field, member->array_size_);
      }
      else
      {
        printf("WARNING: non-fixed size sequences are currently sperimental\n");
        stream « CStringHelper::convert_sequence_to_std_vector_string(field);
      }
    }
    // U16string specific implementation
    else if constexpr(std::is_same_v<T, std::u16string>)
    {
      if (!member->is_array_)
      {
        stream « CStringHelper::convert_to_std_u16string(field);
      }
      else if (member->array_size_ && !member->is_upper_bound_)
      {
        stream « CStringHelper::convert_to_std_vector_u16string(field, member->array_size_);
      }
      else
      {
        printf("WARNING: non-fixed size sequences are currently sperimental\n");
        stream « CStringHelper::convert_sequence_to_std_vector_u16string(field);
      }
    }
    // Generic implementation
    else
    {
      if (!member->is_array_)
      {
        stream « * static_cast<T *>(field);
      }
      else if (member->array_size_ && !member->is_upper_bound_)
      {
        stream.serialize_sequence(static_cast<T *>(field), member->array_size_);
      }
      else
      {
        printf("WARNING: non-fixed size sequences are currently sperimental\n");
        auto & data = *reinterpret_cast<typename GenericCSequence<T>::type *>(field);

        // Serialize length
        stream « (uint32_t)data.size;

        stream.serialize_sequence(reinterpret_cast<T *>(data.data), data.size);
      }
    }
  }

  template<typename MembersType>
  void serialize(const void * msg, const MembersType * casted_members, cbor::TxStream & stream)
  {
    for (uint32_t i = 0; i < casted_members->member_count_; ++i) {
      const auto member = casted_members->members_ + i;
      void * field = const_cast<char *>(static_cast<const char *>(msg)) + member->offset_;
      switch (member->type_id_) {
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_MESSAGE:
          {
            auto sub_members = static_cast<const MembersType *>(member->members_->data);
            if (!member->is_array_) {
              serialize(field, sub_members, stream);
            }
            else if (member->array_size_ && !member->is_upper_bound_)
            {
              for (size_t index = 0; index < member->array_size_; ++index) {
                serialize(member->get_function(field, index), sub_members, stream);
              }
            }
            else
            {
              size_t array_size = member->size_function(field);

              if (member->is_upper_bound_ && array_size > member->array_size_)
              {
                throw std::runtime_error("Sequence overcomes the maximum length");
              }

              // Serialize length
              stream « (uint32_t)array_size;

              for (size_t index = 0; index < array_size; ++index) {
                serialize(member->get_function(field, index), sub_members, stream);
              }
            }
          }
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_BOOLEAN:
          if (!member->is_array_)
          {
            // Don't cast to bool here because if the bool is uninitialized the random value can't be deserialized
            stream « (*static_cast<uint8_t *>(field) ? true : false);
          }
          else
          {
            serialize_field<bool>(member, field, stream);
          }
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_OCTET:
          //throw std::runtime_error("OCTET type unsupported");
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT8:
          serialize_field<uint8_t>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_CHAR:
          serialize_field<char>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT8:
          serialize_field<int8_t>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_FLOAT:
          serialize_field<float>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_DOUBLE:
          serialize_field<double>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT16:
          serialize_field<int16_t>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT16:
          serialize_field<uint16_t>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT32:
          serialize_field<int32_t>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT32:
          serialize_field<uint32_t>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT64:
          serialize_field<int64_t>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT64:
          serialize_field<uint64_t>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_STRING:
          serialize_field<std::string>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_WSTRING:
          serialize_field<std::u16string>(member, field, stream);
          break;
        default:
          throw std::runtime_error("unknown type");
      }
    }
  }

  template<typename T>
  void deserialize_field(const INTROSPECTION_CPP_MEMBER * member, void * field, cbor::RxStream &
stream)
  {
    if (!member->is_array_) {
      stream » *static_cast<T *>(field);
    }
    else if (member->array_size_ && !member->is_upper_bound_)
    {
      stream.deserialize_sequence(static_cast<T *>(field), member->array_size_);
    }
    else
    {
      auto & vector = *reinterpret_cast<std::vector<T> *>(field);
      new(&vector) std::vector<T>;
      stream » vector;
    }
  }

  template<typename T>
  void deserialize_field(const INTROSPECTION_C_MEMBER * member, void * field, cbor::RxStream & stream)
  {
    // String specific implementation
    if constexpr(std::is_same_v<T, std::string>)
    {
      if (!member->is_array_)
      {
        std::string str;
        stream » str;
        CStringHelper::assign_string(str, field);
      }
      else if (member->array_size_ && !member->is_upper_bound_)
      {
        std::vector<std::string> cpp_string_vector;
        stream » cpp_string_vector;

        CStringHelper::assign_vector_string(cpp_string_vector, field, member->array_size_);
      }
      else
      {
        printf("WARNING: non-fixed size sequences are currently sperimental\n");
        std::vector<std::string> cpp_string_vector;
        stream » cpp_string_vector;

        CStringHelper::assign_vector_string_to_sequence(cpp_string_vector, field);
      }
    }
    // U16string specific implementation
    else if constexpr(std::is_same_v<T, std::u16string>)
    {
      if (!member->is_array_)
      {
        std::u16string str;
        stream » str;
        CStringHelper::assign_u16string(str, field);
      }
      else if (member->array_size_ && !member->is_upper_bound_)
      {
        std::vector<std::u16string> cpp_string_vector;
        stream » cpp_string_vector;

        CStringHelper::assign_vector_u16string(cpp_string_vector, field, member->array_size_);
      }
      else
      {
        printf("WARNING: non-fixed size sequences are currently sperimental\n");
        std::vector<std::u16string> cpp_string_vector;
```

```
        stream » cpp_string_vector;

        CStringHelper::assign_vector_u16string_to_sequence(cpp_string_vector, field);
      }
    }
    // Generic implementation
    else
    {
      if (!member->is_array_)
      {
        stream » * static_cast<T *>(field);
      }
      else if (member->array_size_ && !member->is_upper_bound_)
      {
        stream.deserialize_sequence(static_cast<T *>(field), member->array_size_);
      }
      else
      {
        printf("WARNING: non-fixed size sequences are currently sperimental\n");
        auto & data = *reinterpret_cast<typename GenericCSequence<T>::type *>(field);
        uint32_t size = 0;
        stream » size;
        size_t dsize = static_cast<size_t>(size);

        if (!GenericCSequence<T>::init(&data, dsize))
        {
          throw std::runtime_error("unable to initialize GenericCSequence");
        }

        stream.deserialize_sequence(reinterpret_cast<T *>(data.data), dsize);
      }
    }
  }

  template<typename MembersType>
  void deserialize(void * msg, const MembersType * casted_members, cbor::RxStream & stream)
  {
    for (uint32_t i = 0; i < casted_members->member_count_; ++i) {
      const auto member = casted_members->members_ + i;
      void * field = static_cast<char *>(msg) + member->offset_;
      switch (member->type_id_) {
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_MESSAGE:
          {
            auto sub_members = static_cast<const MembersType *>(member->members_->data);
            if (!member->is_array_) {
              deserialize(field, sub_members, stream);
            }
            else if (member->array_size_ && !member->is_upper_bound_)
            {
              for (size_t index = 0; index < member->array_size_; ++index) {
                deserialize(member->get_function(field, index), sub_members, stream);
              }
            }
            else
            {
              // Deserialize length
              uint32_t array_size = 0;
              stream » array_size;

              auto vector = reinterpret_cast<std::vector<unsigned char> *>(field);
              new(vector) std::vector<unsigned char>;
              member->resize_function(field, array_size);

              for (size_t index = 0; index < array_size; ++index) {
                deserialize(member->get_function(field, index), sub_members, stream);
              }
            }
          }
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_BOOLEAN:
          deserialize_field<bool>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_OCTET:
          //throw std::runtime_error("OCTET type unsupported");
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT8:
          deserialize_field<uint8_t>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_CHAR:
          deserialize_field<char>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT8:
          deserialize_field<int8_t>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_FLOAT:
          deserialize_field<float>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_DOUBLE:
          deserialize_field<double>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT16:
          deserialize_field<int16_t>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT16:
          deserialize_field<uint16_t>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT32:
          deserialize_field<int32_t>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT32:
          deserialize_field<uint32_t>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT64:
          deserialize_field<int64_t>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT64:
          deserialize_field<uint64_t>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_STRING:
          deserialize_field<std::string>(member, field, stream);
          break;
        case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_WSTRING:
          deserialize_field<std::u16string>(member, field, stream);
          break;
        default:
          throw std::runtime_error("unknown type");
      }
    }
  }

}


#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:


src/desert_classes/rmw_context_impl_s.h
src/desert_classes/Discovery.h
src/desert_classes/DesertNode.h

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

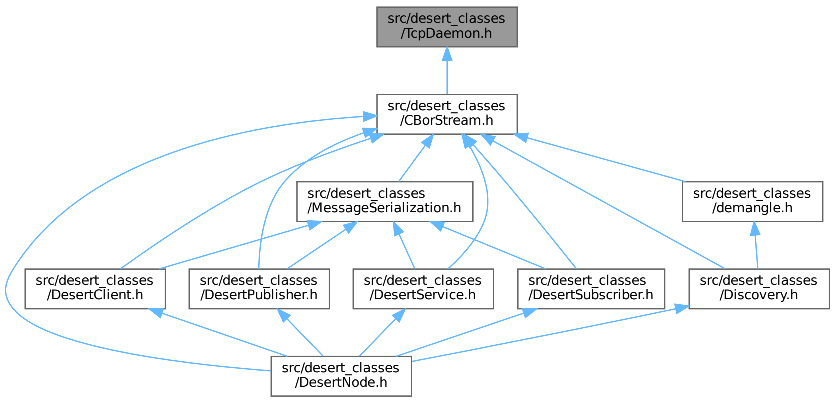
Go to the documentation of this file.

```
/*****************************************************************************
 * Copyright (C) 2024 Davide Costa                                          *
 *                                                                          *
 * This file is part of RMW desert.                                         *
 *                                                                          *
 *   RMW desert is free software: you can redistribute it and/or modify it  *
 *   under the terms of the GNU General Public License as published by the  *
 *   Free Software Foundation, either version 3 of the License, or any      *
 *   later version.                                                         *
 *                                                                          *
 *   RMW desert is distributed in the hope that it will be useful,          *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of         *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          *
 *   GNU General Public License for more details.                           *
 *                                                                          *
 *   You should have received a copy of the GNU General Public License      *
 *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.    *
 *****************************************************************************/

#include "rcpputils/scope_exit.hpp"
#include "rmw_dds_common/context.hpp"
#include "rmw_dds_common/graph_cache.hpp"
#include "rmw_dds_common/msg/participant_entities_info.hpp"
#include "rmw_dds_common/qos.hpp"
#include "rmw_dds_common/security.hpp"

#ifndef RMW_CONTEXT_IMPL_H_
#define RMW_CONTEXT_IMPL_H_

struct rmw_context_impl_s
{
  rmw_dds_common::Context common;
  bool is_shutdown{false};

  rmw_context_impl_s()
  : common()
  {
    /* destructor relies on these being initialized properly */
    common.thread_is_running.store(false);
    common.graph_guard_condition = nullptr;
    common.pub = nullptr;
    common.sub = nullptr;
  }

  ~rmw_context_impl_s()
  {
  }
};

#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:


src/desert_classes /TcpDaemon.h
src/desert_classes /CBorStream.h
src/desert_classes /MessageSerialization.h
src/desert_classes /demangle.h
src/desert_classes /DesertClient.h
src/desert_classes /DesertPublisher.h
src/desert_classes /DesertService.h
src/desert_classes /DesertSubscriber.h
src/desert_classes /Discovery.h
src/desert_classes /DesertNode.h

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 continously to store and send packets in the static members rx_packets and tx_packets

Author

Prof. Davide Costa

7.30 TcpDaemon.h

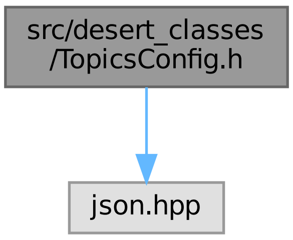
Go to the documentation of this file.

```
/******************************************************************************
 * Copyright (C) 2024 Davide Costa                                           *
 *                                                                           *
 * This file is part of RMW desert.                                          *
 *                                                                           *
 *   RMW desert is free software: you can redistribute it and/or modify it  *
 *   under the terms of the GNU General Public License as published by the  *
 *   Free Software Foundation, either version 3 of the License, or any      *
 *   later version.                                                          *
 *                                                                           *
 *   RMW desert is distributed in the hope that it will be useful,          *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of         *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          *
 *   GNU General Public License for more details.                           *
 *                                                                           *
 *   You should have received a copy of the GNU General Public License      *
 *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.    *
 ******************************************************************************/

#ifndef TCP_DAEMON_H_
#define TCP_DAEMON_H_

#include <queue>
#include <vector>
#include <cstdint>
#include <cstdio>
#include <cstring>
#include <thread>
#include <chrono>

#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/poll.h>
#include <unistd.h>

#include "rmw/error_handling.h"

#define MAX_PACKET_LENGTH 512

#define ADDRESS "127.0.0.1"

#define START_MARKER 0b10011001
#define END_MARKER   0b01010101
#define BYTE_MASK    0b11111111

class TcpDaemon
{
  public:
    TcpDaemon();

    bool init(int port);
    static std::vector<uint8_t> read_packet();
    static void enqueue_packet(std::vector<uint8_t> packet);


  private:
    static int _client_fd;
    static std::queue<std::vector<uint8_t>> _rx_packets;
    static std::queue<std::vector<uint8_t>> _tx_packets;

    void socket_rx_communication();
    void socket_tx_communication();

};

#endif
```

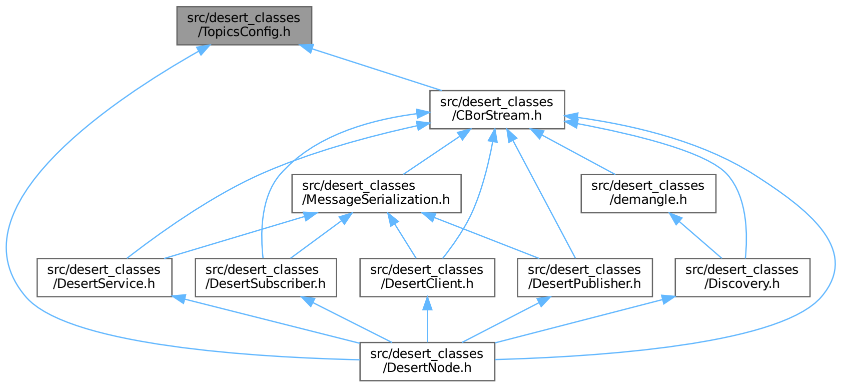
7.31 src/desert_classes/TopicsConfig.h File Reference

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.

```
/*******************************************************************************
 * Copyright (C) 2024 Davide Costa                                             *
 *                                                                             *
 * This file is part of RMW desert.                                            *
 *                                                                             *
 *   RMW desert is free software: you can redistribute it and/or modify it    *
 *   under the terms of the GNU General Public License as published by the    *
 *   Free Software Foundation, either version 3 of the License, or any        *
 *   later version.                                                            *
 *                                                                             *
 *   RMW desert is distributed in the hope that it will be useful,            *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of           *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the            *
 *   GNU General Public License for more details.                             *
 *                                                                             *
 *   You should have received a copy of the GNU General Public License        *
 *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.      *
 *******************************************************************************/

#ifndef TOPICS_CONFIG_H_
#define TOPICS_CONFIG_H_

#include <map>
#include <string>
#include <cstdint>
#include <fstream>

#include "json.hpp"

using namespace nlohmann::json_abi_v3_11_3;

class TopicsConfig
{
  public:

    static void load_configuration();
    static uint8_t get_topic_identifier(std::string name);
    static std::string get_identifier_topic(uint8_t identifier);

  private:
    static std::map<std::string, uint8_t> _topics_list;
    static std::map<uint8_t, std::string> _identifiers_list;
};

#endif  // MACROS_HPP_
```

Index