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

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Namespace Documentation

4.1 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.

4.1.1 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.

4.1.2 Function Documentation

4.1.2.1 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

4.1.2.2 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

4.1.2.3 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

4.1.2.4 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_array	Pointer containing the destination of the string sequence

4.1.2.5 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

4.1.2.6 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_array	Pointer containing the destination of the u16string sequence

4.1.2.7 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_array	Pointer to the first original C-style string

Returns

A C++ vector of strings

4.1.2.8 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_array	Pointer to the first original C-style u16string

Returns

A C++ vector of u16string

4.1.2.9 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

4.1.2.10 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

4.1.2.11 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

4.1.2.12 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

4.2 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.

4.2.1 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.

4.2.2 Function Documentation

4.2.2.1 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

4.2.2.2 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

4.2.2.3 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

4.2.2.4 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

4.2.2.5 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

4.2.2.6 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 5

Class Documentation

5.1 DesertClient Class Reference

Public Member Functions

- DesertClient (std::string service_name, const rosidl_service_type_support_t *type_supports)

  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.

5.1.1 Constructor & Destructor Documentation

5.1.1.1 DesertClient()

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

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

5.1.2 Member Function Documentation

5.1.2.1 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

5.1.2.2 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

req	Pointer to the memory location used to store the reading
req_header	Pointer to the request header used to store the service sequence identifier

5.1.2.3 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

5.2 DesertNode Class Reference

Public Member Functions

- **DesertNode** (const char ∗name)
- const char ∗ **getName** ()

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

- src/desert_classes/DesertNode.h

5.3 DesertPublisher Class Reference

Public Member Functions

- DesertPublisher (std::string topic_name, const rosidl_message_type_support_t ∗type_supports)

  Create a publisher.
- void push (const void ∗msg)

  Send a publication on the topic.

5.3.1 Constructor & Destructor Documentation

5.3.1.1 DesertPublisher()

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

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

5.3.2 Member Function Documentation

5.3.2.1 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

5.4 DesertService Class Reference

Public Member Functions

- DesertService (std::string service_name, const rosidl_service_type_support_t *type_supports)
  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.

5.4.1 Constructor & Destructor Documentation

5.4.1.1 DesertService()

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

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

5.4.2 Member Function Documentation

5.4.2.1 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

5.4.2.2 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

5.4.2.3 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
sequence_id	Pointer to 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

5.5 DesertSubscriber Class Reference

Public Member Functions

- DesertSubscriber (std::string topic_name, const rosidl_message_type_support_t *type_supports)

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.

5.5.1 Constructor & Destructor Documentation

5.5.1.1 DesertSubscriber()

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

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

5.5.2 Member Function Documentation

5.5.2.1 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

5.5.2.2 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

5.6 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

5.7 GenericCSequence< T > Struct Template Reference

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

- src/desert_classes/macros.h

5.8 cbor::RxStream Class Reference

Public Member Functions

- RxStream (uint8_t stream_type, std::string stream_name)

  Create a reception stream.
- bool data_available (int64_t sequence_id=0)

  Check if there are data.
- 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.

Static Public Member Functions

- static void interpret_packets ()

  Interpret raw packets and splits them into different communication types.

5.8.1 Constructor & Destructor Documentation

5.8.1.1 RxStream()

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

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

5.8.2 Member Function Documentation

5.8.2.1 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

5.8.2.2 deserialize_integer()

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

Decode a generic integer.

Parameters

n	Field to decode

5.8.2.3 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

5.8.2.4 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.

5.8.2.5 operator>>() [1/16]

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

Decode bool.

Parameters

b	Field to decode

5.8.2.6 operator>>() [2/16]

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

Decode char.

Parameters

n	Field to decode

5.8.2.7 operator>>() [3/16]

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

Decode double.

Parameters

d	Field to decode

5.8.2.8 operator>>() [4/16]

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

Decode float.

Parameters

f	Field to decode

5.8.2.9 operator>>() [5/16]

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

Decode int16.

Parameters

n	Field to decode

5.8.2.10 operator>>() [6/16]

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

Decode int32.

Parameters

n	Field to decode

5.8.2.11 operator>>() [7/16]

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

Decode int64.

Parameters

n	Field to decode

5.8.2.12 operator>>() [8/16]

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

Decode int8.

Parameters

n	Field to decode

5.8.2.13 operator>>() [9/16]

```
RxStream & cbor::RxStream::operator>> (
```

```
            std::string & s )
```

Decode string.

Parameters

s	Field to decode

5.8.2.14 operator>>() [10/16]

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

Decode u16string.

Parameters

s	Field to decode

5.8.2.15 operator>>() [11/16]

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

Decode bool vector.

Parameters

v	Field to decode

5.8.2.16 operator>>() [12/16]

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

Decode vector.

Parameters

v	Field to decode

5.8.2.17 operator>>() [13/16]

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

Decode uint16.

Parameters

n	Field to decode

5.8.2.18 operator>>() [14/16]

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

Decode uint32.

Parameters

n	Field to decode

5.8.2.19 operator>>() [15/16]

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

Decode uint64.

Parameters

n	Field to decode

5.8.2.20 operator>>() [16/16]

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

Decode uint8.

Parameters

n	Field to decode

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

- src/desert_classes/CBorStream.h
- src/desert_classes/CBorStream.cpp

5.9 TcpDaemon Class Reference

Public Member Functions

- bool init ()

  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.

5.9.1 Member Function Documentation

5.9.1.1 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

5.9.1.2 init()

```
bool TcpDaemon::init ( )
```

Initialize the socket communication.

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

5.9.1.3 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

5.10 cbor::TxStream Class Reference

Public Member Functions

- TxStream (uint8_t stream_type, std::string stream_name)

  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.

5.10.1 Constructor & Destructor Documentation

5.10.1.1 TxStream()

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

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

5.10.2 Member Function Documentation

5.10.2.1 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.

5.10.2.2 operator<<() [1/16]

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

Encode bool.

Parameters

b	Field to encode

5.10.2.3 operator<<() [2/16]

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

Encode char.

Parameters

n	Field to encode

5.10.2.4 operator<<() [3/16]

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

Encode double.

Parameters

d	Field to encode

5.10.2.5 operator<<() [4/16]

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

Encode float.

Parameters

f	Field to encode

5.10.2.6 operator<<() [5/16]

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

Encode int16.

Parameters

n	Field to encode

5.10.2.7 operator<<() [6/16]

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

Encode int32.

Parameters

n	Field to encode

5.10.2.8 operator<<() [7/16]

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

```
            const int64_t n )
```

Encode int64.

Parameters

n	Field to encode

5.10.2.9 operator<<() [8/16]

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

Encode int8.

Parameters

n	Field to encode

5.10.2.10 operator<<() [9/16]

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

Encode string.

Parameters

s	Field to encode

5.10.2.11 operator<<() [10/16]

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

Encode u16string.

Parameters

s	Field to encode

5.10.2.12 operator<<() [11/16]

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

Encode bool vector.

Parameters

v	Field to encode

5.10.2.13 operator<<() [12/16]

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

Encode vector.

Parameters

v	Field to encode

5.10.2.14 operator<<() [13/16]

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

Encode uint16.

Parameters

n	Field to encode

5.10.2.15 operator<<() [14/16]

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

Encode uint32.

Parameters

n	Field to encode

5.10.2.16 operator<<() [15/16]

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

Encode uint64.

Parameters

n	Field to encode

5.10.2.17 operator<<() [16/16]

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

Encode uint8.

Parameters

n	Field to encode

5.10.2.18 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

5.10.2.19 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.

5.10.2.20 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 6

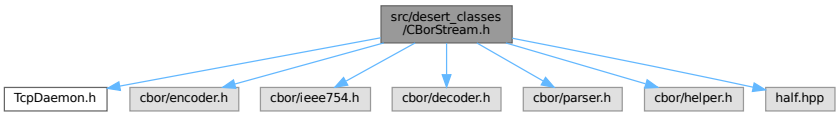
File Documentation

6.1 src/desert_classes/CBorStream.h File Reference

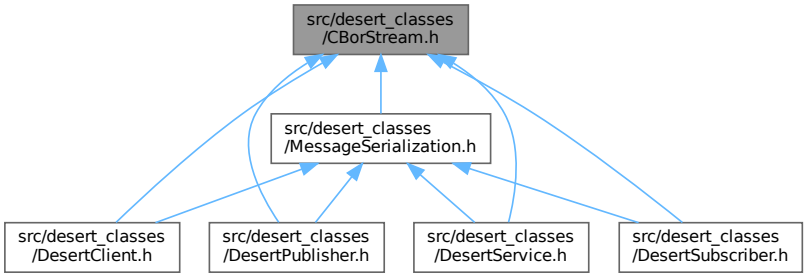
Classes used to convert data types into a CBOR encoded stream.

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

Include dependency graph for CBorStream.h:



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



Classes

- class cbor::TxStream
- class cbor::RxStream

Macros

- #define **PUBLISHER_TYPE** 0
- #define **SUBSCRIBER_TYPE** 1
- #define **CLIENT_TYPE** 2
- #define **SERVICE_TYPE** 3
- #define **MAX_PACKET_LENGTH** 512

6.1.1 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

6.2 CBorStream.h

Go to the documentation of this file.

```
00001 /*********************************************************************
00002  * Copyright (C) 2024 Davide Costa                                   *
00003  *                                                                   *
00004  * This file is part of RMW desert.                                  *
00005  *                                                                   *
00006  *   RMW desert is free software: you can redistribute it and/or modify it  *
00007  *   under the terms of the GNU General Public License as published by the  *
00008  *   Free Software Foundation, either version 3 of the License, or any      *
00009  *   later version.                                                  *
00010  *                                                                   *
00011  *   RMW desert is distributed in the hope that it will be useful,          *
00012  *   but WITHOUT ANY WARRANTY; without even the implied warranty of         *
00013  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          *
00014  *   GNU General Public License for more details.                    *
00015  *                                                                   *
00016  *   You should have received a copy of the GNU General Public License      *
00017  *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.    *
00018  *********************************************************************/
00019
00034 #ifndef CBORSTREAM_H_
00035 #define CBORSTREAM_H_
00036
00037 #include "TcpDaemon.h"
00038
00041 #include <map>
00042 #include <queue>
00043 #include <utility>
00044 #include <vector>
00045 #include <string>
00046 #include <locale>
00047 #include <codecvt>
00048 #include <cstdint>
00049 #include <cstdio>
00050
00053 #include "cbor/encoder.h"
```

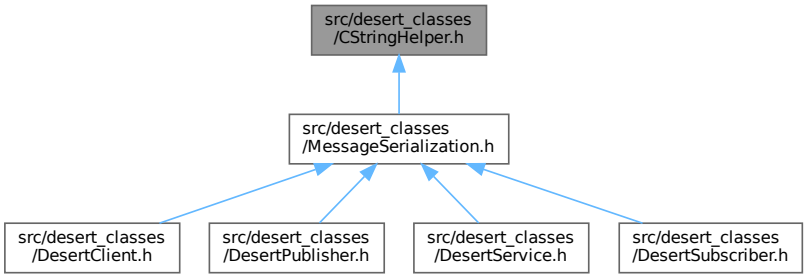
```
00054 #include "cbor/ieee754.h"
00055 #include "cbor/decoder.h"
00056 #include "cbor/parser.h"
00057 #include "cbor/helper.h"
00058
00059 #include "half.hpp"
00060
00061 #define PUBLISHER_TYPE  0
00062 #define SUBSCRIBER_TYPE 1
00063 #define CLIENT_TYPE     2
00064 #define SERVICE_TYPE    3
00065
00066 #define MAX_PACKET_LENGTH 512
00067
00068 namespace cbor
00069 {
00070
00071 class TxStream
00072 {
00073   public:
00080     TxStream(uint8_t stream_type, std::string stream_name);
00081
00091     void start_transmission(uint64_t sequence_id);
00099     void start_transmission();
00106     void end_transmission();
00107
00112     TxStream & operator«(const uint64_t n);
00117     TxStream & operator«(const uint32_t n);
00122     TxStream & operator«(const uint16_t n);
00127     TxStream & operator«(const uint8_t n);
00132     TxStream & operator«(const int64_t n);
00137     TxStream & operator«(const int32_t n);
00142     TxStream & operator«(const int16_t n);
00147     TxStream & operator«(const int8_t n);
00152     TxStream & operator«(const char n);
00157     TxStream & operator«(const float f);
00162     TxStream & operator«(const double d);
00167     TxStream & operator«(const std::string s);
00172     TxStream & operator«(const std::u16string s);
00177     TxStream & operator«(const bool b);
00178
00183     template<typename T>
00184     inline TxStream & operator«(const std::vector<T> v)
00185     {
00186       *this « static_cast<const uint32_t>(v.size());
00187       return serialize_sequence(v.data(), v.size());
00188     }
00189
00194     TxStream & operator«(const std::vector<bool> v);
00195
00201     template<typename T>
00202     inline TxStream & serialize_sequence(const T * items, size_t size)
00203     {
00204       for (size_t i = 0; i < size; ++i)
00205       {
00206         *this « items[i];
00207       }
00208       return *this;
00209     }
00210
00211   private:
00212     uint8_t _stream_type;
00213     std::string _stream_name;
00214
00215     bool _overflow;
00216     uint8_t *  _packet;
00217     cbor_writer_t *  _writer;
00218
00219     void new_packet();
00220     void handle_overrun(cbor_error_t result);
00221
00222     std::string toUTF8(const std::u16string source);
00223
00224 };
00225
00226 class RxStream
00227 {
00228   public:
00235     RxStream(uint8_t stream_type, std::string stream_name);
00236
00246     bool data_available(int64_t sequence_id = 0);
00247
00252     RxStream & operator»(uint64_t & n);
00257     RxStream & operator»(uint32_t & n);
00262     RxStream & operator»(uint16_t & n);
00267     RxStream & operator»(uint8_t & n);
00272     RxStream & operator»(int64_t & n);
```

```
00277     RxStream & operator»(int32_t & n);
00282     RxStream & operator»(int16_t & n);
00287     RxStream & operator»(int8_t & n);
00288
00293     template<typename T>
00294     RxStream & deserialize_integer(T & n);
00295
00300     RxStream & operator»(char & n);
00305     RxStream & operator»(float & f);
00310     RxStream & operator»(double & d);
00315     RxStream & operator»(std::string & s);
00320     RxStream & operator»(std::u16string & s);
00325     RxStream & operator»(bool & b);
00326
00331     template<typename T>
00332     inline RxStream & operator»(std::vector<T> & v)
00333     {
00334       uint32_t size;
00335       *this » size;
00336       v.resize(size);
00337
00338       return deserialize_sequence(v.data(), size);
00339     }
00340
00345     RxStream & operator»(std::vector<bool> & v);
00346
00352     template<typename T>
00353     inline RxStream & deserialize_sequence(T * items, size_t size)
00354     {
00355       for (size_t i = 0; i < size; ++i)
00356       {
00357         *this » items[i];
00358       }
00359       return *this;
00360     }
00361
00369     static void interpret_packets();
00370
00371   private:
00372     uint8_t _stream_type;
00373     std::string _stream_name;
00374
00375     int _buffered_iterator;
00376     std::vector<std::pair<void *, int» _buffered_packet;
00377
00378     // <topic, packets <packet <field, field_type»>
00379     static std::map<std::string, std::queue<std::vector<std::pair<void *, int»»
      _interpreted_publications;
00380     // <service, packets <packet <field, field_type»>
00381     static std::map<std::string, std::queue<std::vector<std::pair<void *, int»» _interpreted_requests;
00382     // <service + id, packets <packet <field, field_type»>
00383     static std::map<std::string, std::queue<std::vector<std::pair<void *, int»»
      _interpreted_responses;
00384
00385     union _cbor_value {
00386     int8_t i8;
00387     int16_t i16;
00388     int32_t i32;
00389     int64_t i64;
00390     float f32;
00391     double f64;
00392     uint8_t *bin;
00393     char *str;
00394     uint8_t str_copy[128];
00395     };
00396
00397     static std::pair<void *, int> interpret_field(cbor_item_t * items, size_t i, union _cbor_value &
      val);
00398     std::u16string toUTF16(const std::string source);
00399 };
00400
00401 }  // namespace cbor
00402
00403
00404 #endif
```

6.3 src/desert_classes/CStringHelper.h File Reference

Namespace containing C sequence handling functions.

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



Namespaces

- namespace CStringHelper

  Namespace containing C sequence handling functions.

Functions

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

  Convert a rosidl_runtime_c__String into std::string.
- std::vector< std::string > CStringHelper::convert_to_std_vector_string (void *str_array, size_t size)

  Convert a rosidl_runtime_c__String into a vector of std::string.
- std::vector< std::string > CStringHelper::convert_sequence_to_std_vector_string (void *str_seq)

  Convert a rosidl_runtime_c__String__Sequence into a vector of std::string.
- std::u16string CStringHelper::convert_to_std_u16string (void *str)

  Convert a rosidl_runtime_c__U16String into std::u16string.
- std::vector< std::u16string > CStringHelper::convert_to_std_vector_u16string (void *str_array, size_t size)

  Convert a rosidl_runtime_c__U16String into a vector of std::u16string.
- std::vector< std::u16string > CStringHelper::convert_sequence_to_std_vector_u16string (void *str_seq)

  Convert a rosidl_runtime_c__U16String__Sequence into a vector of std::u16string.
- void CStringHelper::assign_string (std::string str, void *field)

  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.

6.3.1 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

6.4 CStringHelper.h

Go to the documentation of this file.

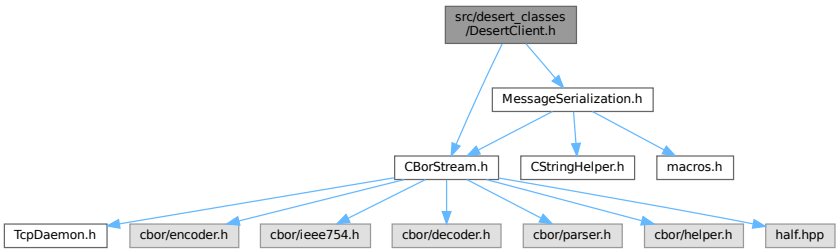
```
00001 /********************************************************************************
00002  * Copyright (C) 2024 Davide Costa                                              *
00003  *                                                                              *
00004  * This file is part of RMW desert.                                             *
00005  *                                                                              *
00006  *   RMW desert is free software: you can redistribute it and/or modify it      *
00007  *   under the terms of the GNU General Public License as published by the      *
00008  *   Free Software Foundation, either version 3 of the License, or any          *
00009  *   later version.                                                             *
00010  *                                                                              *
00011  *   RMW desert is distributed in the hope that it will be useful,              *
00012  *   but WITHOUT ANY WARRANTY; without even the implied warranty of             *
00013  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the              *
00014  *   GNU General Public License for more details.                               *
00015  *                                                                              *
00016  *   You should have received a copy of the GNU General Public License          *
00017  *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.        *
00018  ********************************************************************************/
00019
00032 #ifndef CSTRING_HELPER_H_
00033 #define CSTRING_HELPER_H_
00034
00037 #include "rosidl_runtime_c/u16string.h"
00038 #include "rosidl_runtime_c/string.h"
00039 #include "rosidl_runtime_c/u16string_functions.h"
00040 #include "rosidl_runtime_c/string_functions.h"
00041
00042 #include <stdexcept>
00043 #include <vector>
00044 #include <string>
00045
00056 namespace CStringHelper
00057 {
00066   std::string convert_to_std_string(void * str);
00077   std::vector<std::string> convert_to_std_vector_string(void * str_array, size_t size);
00087   std::vector<std::string> convert_sequence_to_std_vector_string(void * str_seq);
00088
00097   std::u16string convert_to_std_u16string(void * str);
00108   std::vector<std::u16string> convert_to_std_vector_u16string(void * str_array, size_t size);
00118   std::vector<std::u16string> convert_sequence_to_std_vector_u16string(void * str_seq);
00119
00129   void assign_string(std::string str, void * field);
00140   void assign_vector_string(std::vector<std::string> cpp_string_vector, void * str_array, size_t
      size);
00150   void assign_vector_string_to_sequence(std::vector<std::string> cpp_string_vector, void * str_seq);
00151
00161   void assign_u16string(std::u16string str, void * field);
00172   void assign_vector_u16string(std::vector<std::u16string> cpp_string_vector, void * str_array, size_t
      size);
00182   void assign_vector_u16string_to_sequence(std::vector<std::u16string> cpp_string_vector, void *
      str_seq);
00183 }
00184
00185
00186 #endif
```

6.5 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:



Classes

- class DesertClient

6.5.1 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

6.6 DesertClient.h

Go to the documentation of this file.

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

```
00019
00033 #ifndef DESERT_CLIENT_H_
00034 #define DESERT_CLIENT_H_
00035
00038 #include "rosidl_typesupport_introspection_cpp/identifier.hpp"
00039 #include "rosidl_typesupport_introspection_c/identifier.h"
00040 #include "rosidl_typesupport_introspection_cpp/message_introspection.hpp"
00041 #include "rosidl_typesupport_introspection_c/message_introspection.h"
00042 #include "rosidl_typesupport_introspection_cpp/service_introspection.hpp"
00043 #include "rosidl_typesupport_introspection_c/service_introspection.h"
00044 #include "rosidl_typesupport_introspection_cpp/field_types.hpp"
00045 #include "rosidl_typesupport_introspection_c/field_types.h"
00046
00047 #include "rosidl_typesupport_cpp/identifier.hpp"
00048 #include "rosidl_typesupport_c/type_support_map.h"
00049 #include "rosidl_typesupport_c/identifier.h"
00050
00051 #include "rosidl_runtime_c/service_type_support_struct.h"
00052
00053 #include "rmw/types.h"
00054
00055 #include <vector>
00056 #include <string>
00057
00060 #include "CBorStream.h"
00061 #include "MessageSerialization.h"
00062
00063 class DesertClient
00064 {
00065   public:
00072     DesertClient(std::string service_name, const rosidl_service_type_support_t * type_supports);
00073
00083     bool has_data();
00094     void send_request(const void * req, int64_t * sequence_id);
00105     void read_response(void * res, rmw_service_info_t * req_header);
00106
00107
00108   private:
00109     cbor::TxStream _request_data_stream;
00110     cbor::RxStream _response_data_stream;
00111     std::string _name;
00112     int64_t _sequence_id;
00113
00114     int _c_cpp_identifier;
00115     const void * _service;
00116
00117     const void * get_service(const rosidl_service_type_support_t * service_type_support);
00118     const rosidl_service_type_support_t * get_service_type_support(const rosidl_service_type_support_t
      * type_supports);
00119
00120 };
00121
00122 #endif
```

6.7 src/desert_classes/DesertNode.h File Reference

Implementation of the Node structure for DESERT.

Classes

- class DesertNode

6.7.1 Detailed Description

Implementation of the Node structure for DESERT.

Unimplemented class included for future expansions

Author

Prof. Davide Costa

6.8 DesertNode.h

Go to the documentation of this file.

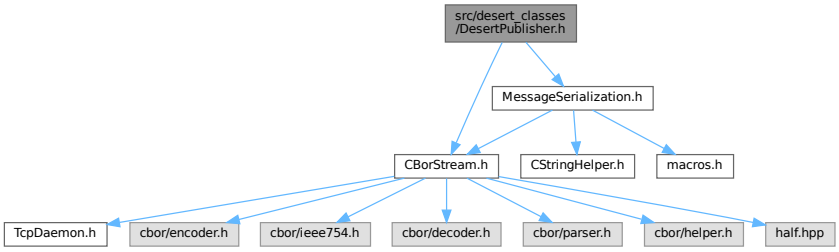
```
00001 /*********************************************************************************
00002  * Copyright (C) 2024 Davide Costa                                               *
00003  *                                                                               *
00004  * This file is part of RMW desert.                                              *
00005  *                                                                               *
00006  *   RMW desert is free software: you can redistribute it and/or modify it       *
00007  *   under the terms of the GNU General Public License as published by the       *
00008  *   Free Software Foundation, either version 3 of the License, or any           *
00009  *   later version.                                                              *
00010  *                                                                               *
00011  *   RMW desert is distributed in the hope that it will be useful,               *
00012  *   but WITHOUT ANY WARRANTY; without even the implied warranty of              *
00013  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the               *
00014  *   GNU General Public License for more details.                                *
00015  *                                                                               *
00016  *   You should have received a copy of the GNU General Public License           *
00017  *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.         *
00018  *********************************************************************************/
00019
00030 #ifndef DESERT_NODE_H_
00031 #define DESERT_NODE_H_
00032
00033 class DesertNode
00034 {
00035   public:
00036     DesertNode(const char* name)
00037     : _name(name)
00038     {}
00039
00040     const char * getName()
00041     {
00042       return _name;
00043     }
00044   private:
00045     const char * _name;
00046 };
00047
00048 #endif
```

6.9 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:



Classes

- class DesertPublisher

6.9.1 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

6.10 DesertPublisher.h

Go to the documentation of this file.

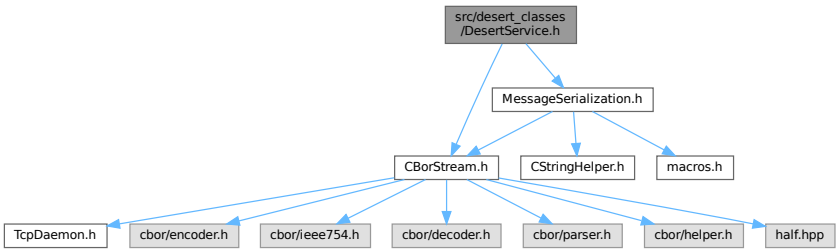
```
00001 /*******************************************************************************
00002  * Copyright (C) 2024 Davide Costa                                             *
00003  *                                                                             *
00004  * This file is part of RMW desert.                                            *
00005  *                                                                             *
00006  *   RMW desert is free software: you can redistribute it and/or modify it     *
00007  *   under the terms of the GNU General Public License as published by the     *
00008  *   Free Software Foundation, either version 3 of the License, or any         *
00009  *   later version.                                                            *
00010  *                                                                             *
00011  *   RMW desert is distributed in the hope that it will be useful,             *
00012  *   but WITHOUT ANY WARRANTY; without even the implied warranty of            *
00013  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the             *
00014  *   GNU General Public License for more details.                              *
00015  *                                                                             *
00016  *   You should have received a copy of the GNU General Public License         *
00017  *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.       *
00018  *******************************************************************************/
00019
00033 #ifndef DESERT_PUBLISHER_H_
00034 #define DESERT_PUBLISHER_H_
00035
00038 #include "rosidl_typesupport_introspection_cpp/identifier.hpp"
00039 #include "rosidl_typesupport_introspection_c/identifier.h"
00040 #include "rosidl_typesupport_introspection_cpp/message_introspection.hpp"
00041 #include "rosidl_typesupport_introspection_c/message_introspection.h"
00042 #include "rosidl_typesupport_introspection_cpp/service_introspection.hpp"
00043 #include "rosidl_typesupport_introspection_c/service_introspection.h"
00044 #include "rosidl_typesupport_introspection_cpp/field_types.hpp"
00045 #include "rosidl_typesupport_introspection_c/field_types.h"
00046
00047 #include "rosidl_typesupport_cpp/identifier.hpp"
00048 #include "rosidl_typesupport_c/type_support_map.h"
00049 #include "rosidl_typesupport_c/identifier.h"
00050
00051 #include "rosidl_runtime_c/message_type_support_struct.h"
00052
00053 #include <vector>
00054 #include <string>
00055
00058 #include "CBorStream.h"
00059 #include "MessageSerialization.h"
00060
00061 class DesertPublisher
00062 {
00063   public:
00070     DesertPublisher(std::string topic_name, const rosidl_message_type_support_t * type_supports);
00071
00081     void push(const void * msg);
00082
00083
00084   private:
00085     cbor::TxStream _data_stream;
00086     std::string _name;
00087
00088     int _c_cpp_identifier;
00089     const void * _members;
00090
00091     const void * get_members(const rosidl_message_type_support_t * type_support);
00092     const rosidl_message_type_support_t * get_type_support(const rosidl_message_type_support_t *
      type_supports);
00093
00094 };
00095
00096 #endif
```

6.11 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:



Classes

- class DesertService

6.11.1 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

6.12 DesertService.h

Go to the documentation of this file.

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

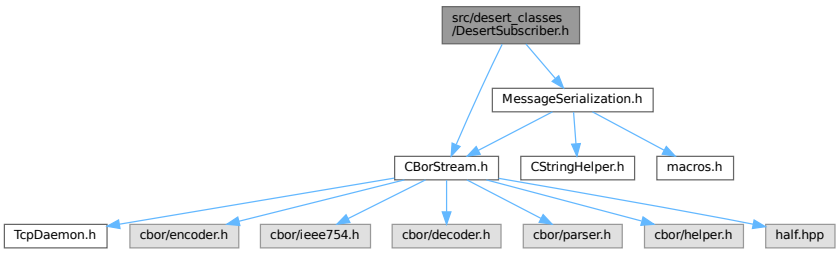
```
00019
00033 #ifndef DESERT_SERVICE_H_
00034 #define DESERT_SERVICE_H_
00035
00038 #include "rosidl_typesupport_introspection_cpp/identifier.hpp"
00039 #include "rosidl_typesupport_introspection_c/identifier.h"
00040 #include "rosidl_typesupport_introspection_cpp/message_introspection.hpp"
00041 #include "rosidl_typesupport_introspection_c/message_introspection.h"
00042 #include "rosidl_typesupport_introspection_cpp/service_introspection.hpp"
00043 #include "rosidl_typesupport_introspection_c/service_introspection.h"
00044 #include "rosidl_typesupport_introspection_cpp/field_types.hpp"
00045 #include "rosidl_typesupport_introspection_c/field_types.h"
00046
00047 #include "rosidl_typesupport_cpp/identifier.hpp"
00048 #include "rosidl_typesupport_c/type_support_map.h"
00049 #include "rosidl_typesupport_c/identifier.h"
00050
00051 #include "rosidl_runtime_c/service_type_support_struct.h"
00052
00053 #include "rmw/types.h"
00054
00055 #include <vector>
00056 #include <string>
00057
00060 #include "CBorStream.h"
00061 #include "MessageSerialization.h"
00062
00063 class DesertService
00064 {
00065   public:
00072     DesertService(std::string service_name, const rosidl_service_type_support_t * type_supports);
00073
00083     bool has_data();
00094     void read_request(void * req, rmw_service_info_t * req_header);
00105     void send_response(void * res, rmw_request_id_t * req_header);
00106
00107
00108   private:
00109     cbor::RxStream _request_data_stream;
00110     cbor::TxStream _response_data_stream;
00111     std::string _name;
00112     int64_t _sequence_id;
00113
00114     int _c_cpp_identifier;
00115     const void * _service;
00116
00117     const void * get_service(const rosidl_service_type_support_t * service_type_support);
00118     const rosidl_service_type_support_t * get_service_type_support(const rosidl_service_type_support_t
      * type_supports);
00119
00120 };
00121
00122 #endif
```

6.13 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:



Classes

- class DesertSubscriber

6.13.1 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

6.14 DesertSubscriber.h

Go to the documentation of this file.

```
00001 /*********************************************************************
00002  * Copyright (C) 2024 Davide Costa                                   *
00003  *                                                                   *
00004  * This file is part of RMW desert.                                  *
00005  *                                                                   *
00006  *   RMW desert is free software: you can redistribute it and/or modify it  *
00007  *   under the terms of the GNU General Public License as published by the  *
00008  *   Free Software Foundation, either version 3 of the License, or any      *
00009  *   later version.                                                  *
00010  *                                                                   *
00011  *   RMW desert is distributed in the hope that it will be useful,          *
00012  *   but WITHOUT ANY WARRANTY; without even the implied warranty of         *
00013  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          *
00014  *   GNU General Public License for more details.                           *
00015  *                                                                   *
00016  *   You should have received a copy of the GNU General Public License      *
00017  *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.    *
00018  *********************************************************************/
00019
00032 #ifndef DESERT_SUBSCRIBER_H_
00033 #define DESERT_SUBSCRIBER_H_
00034
00037 #include "rosidl_typesupport_introspection_cpp/identifier.hpp"
00038 #include "rosidl_typesupport_introspection_c/identifier.h"
00039 #include "rosidl_typesupport_introspection_cpp/message_introspection.hpp"
00040 #include "rosidl_typesupport_introspection_c/message_introspection.h"
00041 #include "rosidl_typesupport_introspection_cpp/service_introspection.hpp"
00042 #include "rosidl_typesupport_introspection_c/service_introspection.h"
00043 #include "rosidl_typesupport_introspection_cpp/field_types.hpp"
00044 #include "rosidl_typesupport_introspection_c/field_types.h"
00045
00046 #include "rosidl_typesupport_cpp/identifier.hpp"
00047 #include "rosidl_typesupport_c/type_support_map.h"
00048 #include "rosidl_typesupport_c/identifier.h"
00049
00050 #include "rosidl_runtime_c/message_type_support_struct.h"
00051
00052 #include <vector>
00053 #include <string>
00054
00057 #include "CBorStream.h"
00058 #include "MessageSerialization.h"
00059
00060 class DesertSubscriber
00061 {
00062   public:
00069     DesertSubscriber(std::string topic_name, const rosidl_message_type_support_t * type_supports);
00070
00080     bool has_data();
00090     void read_data(void * msg);
00091
00092   private:
00093     cbor::RxStream _data_stream;
```

```
00094     std::string _name;
00095
00096     int _c_cpp_identifier;
00097     const void * _members;
00098
00099     const void * get_members(const rosidl_message_type_support_t * type_support);
00100     const rosidl_message_type_support_t * get_type_support(const rosidl_message_type_support_t *
      type_supports);
00101
00102 };
00103
00104 #endif
```

6.15 src/desert_classes/DesertWaitSet.h File Reference

Implementation of the WaitSet structure for DESERT.

Classes

- class DesertWaitset

6.15.1 Detailed Description

Implementation of the WaitSet structure for DESERT.

Unimplemented class included for future expansions

Author

Prof. Davide Costa

6.16 DesertWaitSet.h

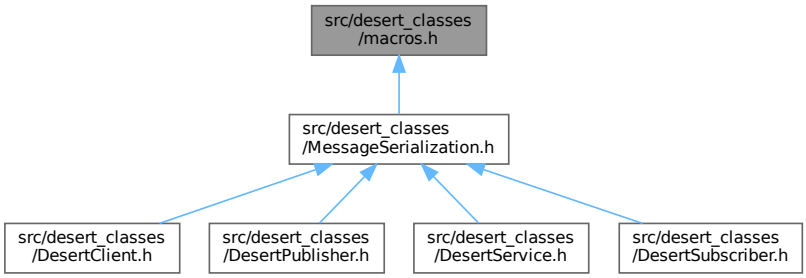
Go to the documentation of this file.

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

6.17 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)

6.17.1 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

6.17.2 Macro Definition Documentation

6.17.2.1 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); \
    } \
  };
```

6.18 macros.h

Go to the documentation of this file.

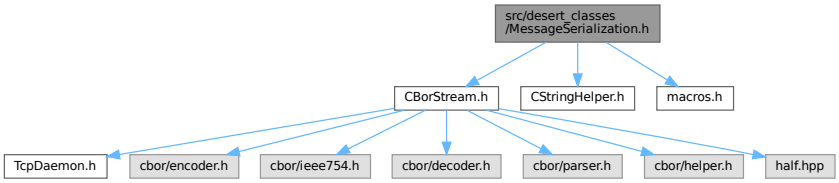
```
00001 /*********************************************************************
00002  * Copyright (C) 2024 Davide Costa                                   *
00003  *                                                                   *
00004  * This file is part of RMW desert.                        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

6.19 src/desert_classes/MessageSerialization.h File Reference

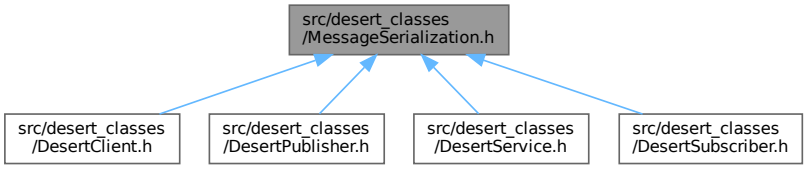
Namespace containing serialization functions.

```
#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.

6.19.1 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

6.20 MessageSerialization.h

Go to the documentation of this file.

```
00001 /*************************************************************************
00002  * Copyright (C) 2024 Davide Costa                                       *
00003  *                                                                       *
00004  * This file is part of RMW desert.                                      *
00005  *                                                                       *
00006  *   RMW desert is free software: you can redistribute it and/or modify it  *
00007  *   under the terms of the GNU General Public License as published by the  *
00008  *   Free Software Foundation, either version 3 of the License, or any      *
00009  *   later version.                                                      *
00010  *                                                                       *
00011  *   RMW desert is distributed in the hope that it will be useful,          *
00012  *   but WITHOUT ANY WARRANTY; without even the implied warranty of         *
00013  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the          *
00014  *   GNU General Public License for more details.                        *
00015  *                                                                       *
00016  *   You should have received a copy of the GNU General Public License      *
00017  *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.    *
00018  *************************************************************************/
00019
00034 #ifndef MESSAGE_SERIALIZATION_H_
00035 #define MESSAGE_SERIALIZATION_H_
00036
00037 #include "CBorStream.h"
00038 #include "CStringHelper.h"
00039 #include "macros.h"
00040
00043 #include <vector>
00044 #include <string>
00045
00048 #define INTROSPECTION_C_MEMBER rosidl_typesupport_introspection_c__MessageMember
```

```
00049 #define INTROSPECTION_CPP_MEMBER rosidl_typesupport_introspection_cpp::MessageMember
00050
00051 #define INTROSPECTION_C_MEMBERS rosidl_typesupport_introspection_c__MessageMembers
00052 #define INTROSPECTION_CPP_MEMBERS rosidl_typesupport_introspection_cpp::MessageMembers
00053
00054 #define INTROSPECTION_C_SERVICE_MEMBERS rosidl_typesupport_introspection_c__ServiceMembers
00055 #define INTROSPECTION_CPP_SERVICE_MEMBERS rosidl_typesupport_introspection_cpp::ServiceMembers
00056
00067 namespace MessageSerialization
00068 {
00069
00081   template<typename T>
00082   void serialize_field(const INTROSPECTION_CPP_MEMBER * member, void * field, cbor::TxStream & stream)
00083   {
00084     if (!member->is_array_)
00085     {
00086       stream « * static_cast<T *>(field);
00087     }
00088     else if (member->array_size_ && !member->is_upper_bound_)
00089     {
00090       stream.serialize_sequence(static_cast<T *>(field), member->array_size_);
00091     }
00092     else
00093     {
00094       std::vector<T> & data = *reinterpret_cast<std::vector<T> *>(field);
00095       stream « data;
00096     }
00097   }
00098
00110   template<typename T>
00111   void serialize_field(const INTROSPECTION_C_MEMBER * member, void * field, cbor::TxStream & stream)
00112   {
00113     // String specific implementation
00114     if constexpr(std::is_same_v<T, std::string>)
00115     {
00116       if (!member->is_array_)
00117       {
00118         stream « CStringHelper::convert_to_std_string(field);
00119       }
00120       else if (member->array_size_ && !member->is_upper_bound_)
00121       {
00122         stream « CStringHelper::convert_to_std_vector_string(field, member->array_size_);
00123       }
00124       else
00125       {
00126         printf("WARNING: non-fixed size sequences are currently sperimental\n");
00127         stream « CStringHelper::convert_sequence_to_std_vector_string(field);
00128       }
00129     }
00130     // U16string specific implementation
00131     else if constexpr(std::is_same_v<T, std::u16string>)
00132     {
00133       if (!member->is_array_)
00134       {
00135         stream « CStringHelper::convert_to_std_u16string(field);
00136       }
00137       else if (member->array_size_ && !member->is_upper_bound_)
00138       {
00139         stream « CStringHelper::convert_to_std_vector_u16string(field, member->array_size_);
00140       }
00141       else
00142       {
00143         printf("WARNING: non-fixed size sequences are currently sperimental\n");
00144         stream « CStringHelper::convert_sequence_to_std_vector_u16string(field);
00145       }
00146     }
00147     // Generic implementation
00148     else
00149     {
00150       if (!member->is_array_)
00151       {
00152         stream « * static_cast<T *>(field);
00153       }
00154       else if (member->array_size_ && !member->is_upper_bound_)
00155       {
00156         stream.serialize_sequence(static_cast<T *>(field), member->array_size_);
00157       }
00158       else
00159       {
00160         printf("WARNING: non-fixed size sequences are currently sperimental\n");
00161         auto & data = *reinterpret_cast<typename GenericCSequence<T>::type *>(field);
00162
00163         // Serialize length
00164         stream « (uint32_t)data.size;
00165
00166         stream.serialize_sequence(reinterpret_cast<T *>(data.data), data.size);
00167       }
```

```
00168     }
00169   }
00170
00183   template<typename MembersType>
00184   void serialize(const void * msg, const MembersType * casted_members, cbor::TxStream & stream)
00185   {
00186     for (uint32_t i = 0; i < casted_members->member_count_; ++i) {
00187       const auto member = casted_members->members_ + i;
00188       void * field = const_cast<char *>(static_cast<const char *>(msg)) + member->offset_;
00189       switch (member->type_id_) {
00190         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_MESSAGE:
00191           {
00192             auto sub_members = static_cast<const MembersType *>(member->members_->data);
00193             if (!member->is_array_) {
00194               serialize(field, sub_members, stream);
00195             }
00196             else if (member->array_size_ && !member->is_upper_bound_)
00197             {
00198               for (size_t index = 0; index < member->array_size_; ++index) {
00199                 serialize(member->get_function(field, index), sub_members, stream);
00200               }
00201             }
00202             else
00203             {
00204               size_t array_size = member->size_function(field);
00205
00206               if (member->is_upper_bound_ && array_size > member->array_size_)
00207               {
00208                 throw std::runtime_error("Sequence overcomes the maximum length");
00209               }
00210
00211               // Serialize length
00212               stream << (uint32_t)array_size;
00213
00214               for (size_t index = 0; index < array_size; ++index) {
00215                 serialize(member->get_function(field, index), sub_members, stream);
00216               }
00217             }
00218           }
00219           break;
00220         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_BOOLEAN:
00221           if (!member->is_array_)
00222           {
00223             // Don't cast to bool here because if the bool is uninitialized the random value can't be
      deserialized
00224             stream << (*static_cast<uint8_t *>(field) ? true : false);
00225           }
00226           else
00227           {
00228             serialize_field<bool>(member, field, stream);
00229           }
00230           break;
00231         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_OCTET:
00232           //throw std::runtime_error("OCTET type unsupported");
00233           break;
00234         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT8:
00235           serialize_field<uint8_t>(member, field, stream);
00236           break;
00237         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_CHAR:
00238           serialize_field<char>(member, field, stream);
00239           break;
00240         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT8:
00241           serialize_field<int8_t>(member, field, stream);
00242           break;
00243         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_FLOAT:
00244           serialize_field<float>(member, field, stream);
00245           break;
00246         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_DOUBLE:
00247           serialize_field<double>(member, field, stream);
00248           break;
00249         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT16:
00250           serialize_field<int16_t>(member, field, stream);
00251           break;
00252         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT16:
00253           serialize_field<uint16_t>(member, field, stream);
00254           break;
00255         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT32:
00256           serialize_field<int32_t>(member, field, stream);
00257           break;
00258         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT32:
00259           serialize_field<uint32_t>(member, field, stream);
00260           break;
00261         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT64:
00262           serialize_field<int64_t>(member, field, stream);
00263           break;
00264         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT64:
00265           serialize_field<uint64_t>(member, field, stream);
```

```
00266           break;
00267         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_STRING:
00268           serialize_field<std::string>(member, field, stream);
00269           break;
00270         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_WSTRING:
00271           serialize_field<std::u16string>(member, field, stream);
00272           break;
00273         default:
00274           throw std::runtime_error("unknown type");
00275       }
00276     }
00277   }
00278
00290   template<typename T>
00291   void deserialize_field(const INTROSPECTION_CPP_MEMBER * member, void * field, cbor::RxStream &
      stream)
00292   {
00293     if (!member->is_array_) {
00294       stream » *static_cast<T *>(field);
00295     }
00296     else if (member->array_size_ && !member->is_upper_bound_)
00297     {
00298       stream.deserialize_sequence(static_cast<T *>(field), member->array_size_);
00299     }
00300     else
00301     {
00302       auto & vector = *reinterpret_cast<std::vector<T> *>(field);
00303       new(&vector) std::vector<T>;
00304       stream » vector;
00305     }
00306   }
00307
00319   template<typename T>
00320   void deserialize_field(const INTROSPECTION_C_MEMBER * member, void * field, cbor::RxStream & stream)
00321   {
00322     // String specific implementation
00323     if constexpr(std::is_same_v<T, std::string>)
00324     {
00325       if (!member->is_array_)
00326       {
00327         std::string str;
00328         stream » str;
00329         CStringHelper::assign_string(str, field);
00330       }
00331       else if (member->array_size_ && !member->is_upper_bound_)
00332       {
00333         std::vector<std::string> cpp_string_vector;
00334         stream » cpp_string_vector;
00335
00336         CStringHelper::assign_vector_string(cpp_string_vector, field, member->array_size_);
00337       }
00338       else
00339       {
00340         printf("WARNING: non-fixed size sequences are currently sperimental\n");
00341         std::vector<std::string> cpp_string_vector;
00342         stream » cpp_string_vector;
00343
00344         CStringHelper::assign_vector_string_to_sequence(cpp_string_vector, field);
00345       }
00346     }
00347     // U16string specific implementation
00348     else if constexpr(std::is_same_v<T, std::u16string>)
00349     {
00350       if (!member->is_array_)
00351       {
00352         std::u16string str;
00353         stream » str;
00354         CStringHelper::assign_u16string(str, field);
00355       }
00356       else if (member->array_size_ && !member->is_upper_bound_)
00357       {
00358         std::vector<std::u16string> cpp_string_vector;
00359         stream » cpp_string_vector;
00360
00361         CStringHelper::assign_vector_u16string(cpp_string_vector, field, member->array_size_);
00362       }
00363       else
00364       {
00365         printf("WARNING: non-fixed size sequences are currently sperimental\n");
00366         std::vector<std::u16string> cpp_string_vector;
00367         stream » cpp_string_vector;
00368
00369         CStringHelper::assign_vector_u16string_to_sequence(cpp_string_vector, field);
00370       }
00371     }
00372     // Generic implementation
00373     else
```

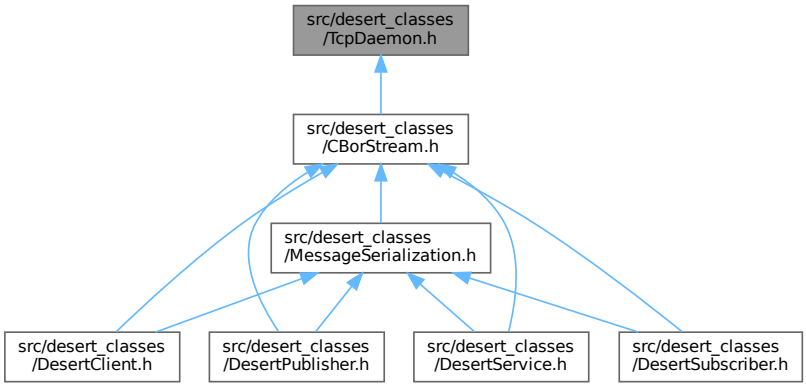
```
00374     {
00375       if (!member->is_array_)
00376       {
00377         stream » * static_cast<T *>(field);
00378       }
00379       else if (member->array_size_ && !member->is_upper_bound_)
00380       {
00381         stream.deserialize_sequence(static_cast<T *>(field), member->array_size_);
00382       }
00383       else
00384       {
00385         printf("WARNING: non-fixed size sequences are currently sperimental\n");
00386         auto & data = *reinterpret_cast<typename GenericCSequence<T>::type *>(field);
00387         uint32_t size = 0;
00388         stream » size;
00389         size_t dsize = static_cast<size_t>(size);
00390
00391         if (!GenericCSequence<T>::init(&data, dsize))
00392         {
00393           throw std::runtime_error("unable to initialize GenericCSequence");
00394         }
00395
00396         stream.deserialize_sequence(reinterpret_cast<T *>(data.data), dsize);
00397       }
00398     }
00399   }
00400
00414   template<typename MembersType>
00415   void deserialize(void * msg, const MembersType * casted_members, cbor::RxStream & stream)
00416   {
00417     for (uint32_t i = 0; i < casted_members->member_count_; ++i) {
00418       const auto member = casted_members->members_ + i;
00419       void * field = static_cast<char *>(msg) + member->offset_;
00420       switch (member->type_id_) {
00421         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_MESSAGE:
00422           {
00423             auto sub_members = static_cast<const MembersType *>(member->members_->data);
00424             if (!member->is_array_) {
00425               deserialize(field, sub_members, stream);
00426             }
00427             else if (member->array_size_ && !member->is_upper_bound_)
00428             {
00429               for (size_t index = 0; index < member->array_size_; ++index) {
00430                 deserialize(member->get_function(field, index), sub_members, stream);
00431               }
00432             }
00433             else
00434             {
00435               // Deserialize length
00436               uint32_t array_size = 0;
00437               stream » array_size;
00438
00439               auto vector = reinterpret_cast<std::vector<unsigned char> *>(field);
00440               new(vector) std::vector<unsigned char>;
00441               member->resize_function(field, array_size);
00442
00443               for (size_t index = 0; index < array_size; ++index) {
00444                 deserialize(member->get_function(field, index), sub_members, stream);
00445               }
00446             }
00447           }
00448           break;
00449         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_BOOLEAN:
00450           deserialize_field<bool>(member, field, stream);
00451           break;
00452         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_OCTET:
00453           //throw std::runtime_error("OCTET type unsupported");
00454           break;
00455         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT8:
00456           deserialize_field<uint8_t>(member, field, stream);
00457           break;
00458         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_CHAR:
00459           deserialize_field<char>(member, field, stream);
00460           break;
00461         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT8:
00462           deserialize_field<int8_t>(member, field, stream);
00463           break;
00464         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_FLOAT:
00465           deserialize_field<float>(member, field, stream);
00466           break;
00467         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_DOUBLE:
00468           deserialize_field<double>(member, field, stream);
00469           break;
00470         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT16:
00471           deserialize_field<int16_t>(member, field, stream);
00472           break;
00473         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT16:
```

```
00474           deserialize_field<uint16_t>(member, field, stream);
00475           break;
00476         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT32:
00477           deserialize_field<int32_t>(member, field, stream);
00478           break;
00479         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT32:
00480           deserialize_field<uint32_t>(member, field, stream);
00481           break;
00482         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_INT64:
00483           deserialize_field<int64_t>(member, field, stream);
00484           break;
00485         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_UINT64:
00486           deserialize_field<uint64_t>(member, field, stream);
00487           break;
00488         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_STRING:
00489           deserialize_field<std::string>(member, field, stream);
00490           break;
00491         case ::rosidl_typesupport_introspection_cpp::ROS_TYPE_WSTRING:
00492           deserialize_field<std::u16string>(member, field, stream);
00493           break;
00494         default:
00495           throw std::runtime_error("unknown type");
00496       }
00497     }
00498   }
00499
00500 }
00501
00502
00503 #endif
```

6.21 src/desert_classes/TcpDaemon.h File Reference

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

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



Classes

- class TcpDaemon

Macros

- #define **ADDRESS** "127.0.0.1"
- #define **PORT** 4000
- #define **END_MARKER** 0b01010101
- #define **BYTE_MASK** 0b11111111

6.21.1 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

6.22 TcpDaemon.h

Go to the documentation of this file.

```
00001 /*****************************************************************************
00002  * Copyright (C) 2024 Davide Costa                                           *
00003  *                                                                           *
00004  * This file is part of RMW desert.                                          *
00005  *                                                                           *
00006  *   RMW desert is free software: you can redistribute it and/or modify it   *
00007  *   under the terms of the GNU General Public License as published by the   *
00008  *   Free Software Foundation, either version 3 of the License, or any       *
00009  *   later version.                                                          *
00010  *                                                                           *
00011  *   RMW desert is distributed in the hope that it will be useful,           *
00012  *   but WITHOUT ANY WARRANTY; without even the implied warranty of          *
00013  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the           *
00014  *   GNU General Public License for more details.                            *
00015  *                                                                           *
00016  *   You should have received a copy of the GNU General Public License       *
00017  *   along with RMW desert.  If not, see <http://www.gnu.org/licenses/>.     *
00018  *****************************************************************************/
00019
00033 #ifndef TCP_DAEMON_H_
00034 #define TCP_DAEMON_H_
00035
00038 #include <queue>
00039 #include <vector>
00040 #include <cstdint>
00041 #include <cstdio>
00042 #include <cstring>
00043 #include <thread>
00044 #include <chrono>
00045
00046 #include <arpa/inet.h>
00047 #include <sys/socket.h>
00048 #include <sys/poll.h>
00049 #include <unistd.h>
00050
00051 #include "rmw/error_handling.h"
00052
00055 #define ADDRESS "127.0.0.1"
00056 #define PORT 4000
00057
00058 #define END_MARKER 0b01010101
00059 #define BYTE_MASK 0b11111111
00060
00061 class TcpDaemon
00062 {
00063   public:
00064     TcpDaemon();
00065
00072     bool init();
00081     static std::vector<uint8_t> read_packet();
00090     static void enqueue_packet(std::vector<uint8_t> packet);
00091
00092
00093   private:
00094     static int _client_fd;
00095     static std::queue<std::vector<uint8_t>> _rx_packets;
00096     static std::queue<std::vector<uint8_t>> _tx_packets;
00097
00098     void socket_rx_communication();
00099     void socket_tx_communication();
00100
00101 };
00102
00103 #endif
```

Index