RTI Secure Wan Transport

Version 5.0.0

Generated by Doxygen 1.5.5

Mon Aug 13 09:10:05 2012

Contents

Chapter 1

Secure WAN Transport for RTI Connext

Real-Time Innovations, Inc.

The Secure WAN Transport allows RTI Connext applications that are running on private networks to communicate securely over a Wide-Area Network (WAN).

1.1 Available Documentation

This document contains:

- **Overview** (p. 2)
- **Configuring the Secure WAN Transports** (p. 3)
- **Example Applications** (p. 4)

For additional information, please see the following PDF documents:

- RTI Secure WAN Transport Installation Guide.
- RTI Secure WAN Transport Release Notes.
- RTI Secure WAN Transport API Reference.

1.2 Overview

RTI Secure WAN Transport is an optional package that provides transport plugins which can be used by developers of RTI Connext applications. These transport plugins allow RTI Connext applications that are running on private networks to communicate securely over a Wide-Area Network (WAN), such the internet. There are two primary components of the package, which may be used independently or together: communication over Wide-Area Networks that involve Network Address Translators (NATs), and secure communication with support for peer authentication and encrypted data transport.

The RTI Connext core is transport-agnostic. RTI Connext offers three built-in transports: UDP/IPv4, UDP/IPv6, and inter-process shared memory. The implementation of NAT traversal and secure communication is done at the transport level so that the DDS core is not affected and does not need to be changed, although there is additional on-the-wire traffic.

Refer to the `RTI Core Libraries and Utilities User's Manual` for further information.

The basic problem to overcome in a WAN environment is that messages sent from an application on a private local-area network (LAN) appear to come from the LAN's router address, and not from the internal IP address of the host running the application. This is due to the existence of a Network Address Translator (NAT) at the gateway. This does not cause problems for client-server systems because only the server needs to be globally addressable; it is only a problem for systems with peer-to-peer communication models, such as DDS. RTI Secure WAN Transport solves this problem, allowing communication between peers that are in separate LAN networks, using a UDP hole-punching mechanism based on the STUN protocol (IETF RFC 3489bis) for NAT traversal. This requires the use of an additional rendezvous server application, the RTI WAN Server.

Once the transport has enabled traffic to cross the NAT gateway to the WAN, it is flowing on network hardware that is shared (in some cases, over the public internet). In this context, it is important to consider the security of data transmission. There are three primary issues involved: a) authenticating the communication peer (source or destination) as a trusted partner; b) encrypting the data to hide it from other parties that may have access to the network; and c) validating the received data to ensure that it was not modified in transmission. RTI Secure WAN Transport addresses these problems by wrapping all RTPS-encoded data using the DTLS protocol (IETF RFC 4347), which is a variant of SSL/TLS that can be used over a datagram network-layer transport such as UDP.

The security features of the WAN Transport may also be used on an untrusted local-area network with the Secure Transport.

In summary, the package includes two transports:

- The **WAN Transport** (p. 9) is for use on a WAN **and includes security**. It must be used with the **WAN Server** (p. 12), a rendezvous server that provides the ability to discover public addresses and to register and look up peer addresses based on a unique WAN ID. The WAN Server is based on the STUN (Session Traversal Utilities for NAT) protocol [draft-ietf-behave-rfc3489bis], with some extensions. Once information about public addresses for the application and its peers has been obtained and connections have been initiated, the server is no longer required to maintain communication with a peer. (Note: security is disabled by default.)

- The **Secure Transport** (p. 20) is an alternate transport that provides security on an untrusted LAN. Use of the RTI WAN Server is not required.

1.3 Configuring the Secure WAN Transports

There are two ways in which these transports can be configured:

1. By setting up predefined strings in the Property QoS Policy of the DomainParticipant (on UNIX, Solaris and Windows systems only, see **NDDSTransportLoadPluginModule**).
   - WAN Transport: Refer to **Configure WAN Transport with Property QoS Policy** (p. 13) for the predefined property strings.
   - Secure Transport: Refer to **Configure Secure Transport with Property QoS Policy** (p. 24) for the predefined property strings.

   With this first approach, RTI Connext will dynamically load the WAN or Secure Transport libraries at run time and then implicitly create and register the transport plugin.

2. By instantiating a new transport and registering it with the DomainParticipant (available in C/C++ API only, see **NDDS_Transport_Support_register_transport()**)
   - WAN Transport: Refer to **WAN Transport C API Reference** (p. 15).
   - Secure Transport: Refer to **Secure Transport C API Reference** (p. 26).

   To use this second approach, you need access to the Secure WAN Transport API at compile time. Therefore, you must include additional header files and libraries. Refer to the `RTI Core Libraries and Utilities User's Manual` and the `RTI Core Libraries and Utilities Platform Notes` for further information.

Refer to the C/C++ API online documentation for details on these two approaches.

1.4 Example Applications

A simple example is available to show how to configure the WAN transport. It includes example settings to enable communication over WAN, and optional settings to enable security (along with example certificate files to use for secure communicaion).

C: See the example in <RTI_Data_Distribution_Service_INSTALL_ROOT>/example/C/HelloWorldWAN.

C++: See the example in <RTI_Data_Distribution_Service_INSTALL_ROOT>/example/CPP/HelloWorldWAN.

Java: See the example in <RTI_Data_Distribution_Service_INSTALL_ROOT>/example/JAVA/HelloWorldWAN.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

Module Documentation

4.1 WAN Transport

Modules

- **WAN Server**

  How to run the RTI WAN Server.

- **Configure WAN Transport with Property QoS Policy**

  Predefined strings that can be used to configure the WAN Transport plugin through the Property QoS Policy of the DomainParticipant.

- **WAN Transport C API Reference**

  Basic types and macros provided by RTI Connext for use in the WAN Transport-Plugin interface.

4.1.1 Detailed Description

The basic problem to overcome in a WAN environment is that messages sent from an application on a private local-area network (LAN) appear to come from the LAN's router address, and not from the internal IP address of the host running the application. This is due to the existence of a Network Address Translator (NAT) at the gateway. This does not cause problems for client-server systems because only the server needs to be globally addressable; it is a problem for systems with peer-to-peer communication models, such as DDS.

In order to resolve the problem of communication across NAT boundaries, the WAN Transport implements a UDP hole-punching solution for NAT traversal

[draft-ietf-behave-p2p-state]. This solution uses a rendezvous server, which provides the ability to discover public addresses, and to register and lookup peer addresses based on a unique WAN ID. This server is based on the STUN (Session Traversal Utilities for NAT) protocol [draft-ietf-behave-rfc3489bis], with some extensions. Once information about public addresses for the application and its peers has been obtained, and connections have been initiated, the server is no longer required to maintain communication with a peer. However, if communication fails, possibly due to changes in dynamically-allocated addresses, the server will be needed to reopen new public channels.

The following figure shows the RTI WAN transport architecture

Multicast communication is not supported by the WAN transport. If the **multicast_enabled** property is set to 1, the transport will return an error message.

To use the WAN Transport, the following parameters must be set:

- **NDDS_Transport_WAN_Property_t::transport_instance_id** (p. 185): the WAN Instance ID of the transport instance; these must be unique for all the transport instances communicating with the same WAN server.
- **NDDS_Transport_WAN_Property_t::server** (p. 186): the address of the WAN server; this must be a publicly accessible address, and the server must be outside all NATs through which communication must pass.

These additional parameters may be needed:

- **NDDS_Transport_WAN_Property_t::server_port** (p. 186): the port of the WAN server; the default STUN port is 3478. This must match the port to which the WAN Server is bound.
- **NDDS_Transport_WAN_Property_t::interface_address** (p. 185): if given, all sockets will be bound to the specified network interface. For multi-homed systems, the network interface used for WAN communication must be specified to guarantee proper NAT traversal behavior. Multiple instances of the WAN transport bound to different interfaces can coexist. They must have unique WAN Instance IDs.
- **NDDS_Transport_WAN_Property_t::port_offset** (p. 185): an offset added to the DDS internal port numbering scheme to ensure that an instance of the WAN transport can coexist with a non-WAN UDP transport.

In order to enable security, **NDDS_Transport_WAN_Property_t::enable_security** (p. 184) must be set, as well as all required fields in **NDDS_Transport_WAN_Property_t::tls** (p. 185). Information about how to configure these parameters can be found in **Secure Transport** (p. 20).

4.1.2 WAN locators

The WAN transport does not use simple IP addresses to locate peers. A WAN transport locator consists of a WAN ID, which is an arbitrary 12-byte value, and a bottom 4-byte value that specifies a fallback local IPv4 address.

The anatomy of a WAN locator is illustrated below:

The address is a 128-bit address in IPv6 notation.

The "wan://" part specifies that the address is for the WAN transport.

The next part, "::1", specifies the top 12 bytes of the address to be 11 zero bytes, followed by a byte with value 1 (this corresponds to the peer's WAN ID).

The last part, "10.10.1.150" refers to the peers local IPv4 address, which will be used if the peers are on the same local network.

A DomainParticipant using the WAN transport will have to initialize the **DDS_DiscoveryQosPolicy::initial_peers** QoS with the WAN locator addresses corresponding to the peers to which it wants to connect to. The value of **DDS_DiscoveryQosPolicy::initial_peers** can be set using the environment variable NDDS_DISCOVERY_PEERS or the NDDS_DISCOVERY_PEERS configuration file.

Refer to the `RTI Core Libraries and Utilities User's Manual` for further information.

4.2 WAN Server

How to run the RTI WAN Server. The WAN transport requires a rendezvous server, which provides the ability to discover public addresses and to register and look up peer addresses based on a unique WAN ID. This server is based on the STUN (Session Traversal Utilities for NAT) protocol [draft-ietf-behave-rfc3489bis], with some extensions. The server must be bound to a network port outside all NATs through which communication must pass.

The RTI WAN Server executable is located in **$NDDSHOME/scripts**.

```
rtiwanserver [options]

Options:
    -help       Display help information
    -address <string>   Server address (default: gethostname() output)
    -port <integer>     Server port (default: 3478)
    -verbosity <integer>       Log verbosity [0-5] (default: 1 - exceptions -)
```

4.3 Configure WAN Transport with Property QoS Policy

Predefined strings that can be used to configure the WAN Transport plugin through the Property QoS Policy of the DomainParticipant.

Property Name	Description	Required?
dds.transport.load_plugins	Comma-separated strings indicating the prefix names of all plugins that will be loaded by RTI Connext. Up to 8 plugins may be specified. For example, "dds.transport.WAN.wan1". In the following examples, <*WAN_prefix*> indicates the string that is used as a prefix in the property names for all the settings that are related to this WAN transport plugin. <*WAN_prefix*> must begin with "dds.transport." (such as "dds.transport.WAN.wan1").	**YES**
<*WAN_prefix*>.library	Must set to **"libnddstransport-wan.so"** (for UNIX/Solaris) or **"nddstransport-wan.dll"** (for Windows). This library (and the dependent openssl libraries if security is turned on) needs to be in the path during run time for use by RTI Connext (in the **LD_LIBRARY_PATH** environment variable on UNIX/Solaris, in **PATH** for Windows)	**YES**
<*WAN_prefix*>.create_function	Must be set to **"NDDS_Transport_WAN_create"** .	**YES**
<*WAN_prefix*>.aliases	Used to register the transport plugin returned by **NDDS_Transport_WAN_create()** (p. 18) (as specified by <***WAN_prefix***>**.create_function**) to the domain participant.	NO

4.4 WAN Transport C API Reference

Basic types and macros provided by RTI Connext for use in the WAN Transport-Plugin interface.

Data Structures

- struct **NDDS_Transport_WAN_Property_t**

  WAN transport plugin property.

Defines

- #define **NDDS_TRANSPORT_WAN_TRANSPORT_INSTANCE_ID_LENGTH** 12

  *Length of private network ID in **NDDS_Transport_WAN_Property_t::transport_instance_id** (p. 185).*

- #define **NDDS_TRANSPORT_WAN_ADDRESS_BIT_COUNT** (128)

  Default value of **NDDS_Transport_Property_t::address_bit_count** *in **NDDS_Transport_WAN_Property_t::parent** (p. 184).*

- #define **NDDS_TRANSPORT_WAN_PROPERTIES_BITMAP_DEFAULT** (0)

  Default value of **NDDS_Transport_Property_t::properties_bitmap** *in **NDDS_Transport_WAN_Property_t::parent** (p. 184).*

- #define **NDDS_TRANSPORT_WAN_DTLS_FRAGMENT_SIZE_MAX_DEFAULT** (9216)

  *Default value of **NDDS_Transport_WAN_Property_t::recv_decode_buffer_size** (p. 184).*

- #define **NDDS_TRANSPORT_WAN_PROPERTY_DEFAULT**

  *Use this to initialize a **NDDS_Transport_WAN_Property_t** (p. 183) structure.*

Functions

- NDDS_Transport_Plugin * **NDDS_Transport_WAN_create** (**NDDS_Transport_Address_t** *default_network_address_out, const struct **DDS_PropertyQosPolicy** *property_in)

Create an instance of a WAN Transport Plugin using PropertyQosPolicy (for dynamic loading).

ˆ NDDS_Transport_Plugin * **NDDS_Transport_WAN_new** (const struct **NDDS_Transport_WAN_Property_t** *property_in)

Create an instance of a WAN Transport Plugin.

4.4.1 Detailed Description

Basic types and macros provided by RTI Connext for use in the WAN Transport-Plugin interface.

4.4.2 Define Documentation

4.4.2.1 #define NDDS_TRANSPORT_WAN_TRANSPORT_INSTANCE_ID_LENGTH 12

Length of private network ID in **NDDS_Transport_WAN_Property_t::transport_instance_id** (p. 185).

4.4.2.2 #define NDDS_TRANSPORT_WAN_ADDRESS_BIT_COUNT (128)

Default value of **NDDS_Transport_Property_t::address_bit_count** in **NDDS_Transport_WAN_Property_t::parent** (p. 184).

4.4.2.3 #define NDDS_TRANSPORT_WAN_PROPERTIES_BITMAP_DEFAULT (0)

Default value of **NDDS_Transport_Property_t::properties_bitmap** in **NDDS_Transport_WAN_Property_t::parent** (p. 184).

4.4.2.4 #define NDDS_TRANSPORT_WAN_DTLS_FRAGMENT_SIZE_MAX_DEFAULT (9216)

Default value of **NDDS_Transport_WAN_Property_t::recv_decode_buffer_size** (p. 184).

4.4.2.5 #define NDDS_TRANSPORT_WAN_PROPERTY_DEFAULT

Value:

```
{ \
    { \
      { NDDS_TRANSPORT_CLASSID_WAN, /* classid */ \
        NDDS_TRANSPORT_WAN_ADDRESS_BIT_COUNT, /* address_bit_count */ \
        NDDS_TRANSPORT_WAN_PROPERTIES_BITMAP_DEFAULT, /* properties_bitmap */ \
        NDDS_TRANSPORT_UDPV4_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT, /* gather_send_buffer_count_max */ \
        NDDS_TRANSPORT_UDPV4_MESSAGE_SIZE_MAX_DEFAULT, /* message_size_max */ \
        NULL, 0, /* allow_interfaces_list, allow_interfaces_list_length */ \
        NULL, 0, /* deny_interfaces_list deny_interfaces_list_length */ \
        NULL, 0, /* allow_multicast_interfaces_list, allow_multicast_interfaces_list_length */ \
        NULL, 0, /* deny_multicast_interfaces_list, deny_multicast_interfaces_list_length */ \
      }, /* parent (NDDS_Transport_Property_t) */ \
      NDDS_TRANSPORT_UDPV4_MESSAGE_SIZE_MAX_DEFAULT, /* send_socket_buffer_size */ \
      NDDS_TRANSPORT_UDPV4_MESSAGE_SIZE_MAX_DEFAULT, /* recv_socket_buffer_size */ \
      1, /* unicast_enabled (use unicast) */   \
      0, /* multicast_enabled (do NOT use multicast) */ \
      NDDS_TRANSPORT_UDPV4_MULTICAST_TTL_DEFAULT, /* multicast_ttl */ \
      0, /* multicast_loopback_disabled (enable) */ \
      -1, /* ignore_loopback_interface (auto) */ \
      1, /* ignore_nonup_interfaces */ \
      0, /* ignore_nonrunning_interfaces (do not ignore non-RUNNING) */ \
      0, /* no_zero_copy */ \
      NDDS_TRANSPORT_UDPV4_BLOCKING_DEFAULT, /* send_blocking */ \
      0, /* transport_priority_mask (no mapping to IP_TOS by default) */ \
      0, /* transport_priority_mapping_low */ \
      0xff /* transport_priority_mapping_high */ \
    }, /* parent (NDDS_Transport_UDPv4_Property_t) */ \
    0, /* enable_security */ \
    NDDS_TRANSPORT_WAN_DTLS_FRAGMENT_SIZE_MAX_DEFAULT, /* recv_decode_buffer_size (actually want MTU) */ \
    144, /* port_offset */ \
    1000, /* dtls_handshake_resend_interval (1 second) */ \
    7, /* dtls_handshake_number_of_resends (last resend at ~60s) */ \
    0, /* transport_mtu (don't set MTU) */ \
    NDDS_TRANSPORT_TLS_OPENSSL_CONFIGURATION_DEFAULT, /* tls */ \
    {0,0,0,0,0,0,0,0,0,0,0,0}, /* transport_instance_id (Network id) */ \
    NULL, /* interface_address (Interface address to bind) */ \
    NULL, 3478, /* server, server_port */ \
    100, /* stun_retransmission_interval */ \
    7, /* stun_number_of_retransmissions */ \
    15000 /* stun_liveliness_period */ \
}
```

Use this to initialize a **NDDS_Transport_WAN_Property_t** (p. 183) structure.

4.4.3 Function Documentation

4.4.3.1 NDDS_Transport_Plugin* NDDS_Transport_WAN_create (NDDS_Transport_Address_t * *default_network_address_out*, const struct DDS_PropertyQosPolicy * *property_in*)

Create an instance of a WAN Transport Plugin using PropertyQosPolicy (for dynamic loading).

An application may create and register multiple instances of this Transport Plugin with RTI Connext. This may be to partition the network interfaces across multiple RTI Connext domains. However, note that the underlying transport, the operating system's IP layer, is still a "singleton". For example, if a unicast transport has already bound to a port, and another unicast transport tries to bind to the same port, the second attempt will fail. The **NDDS_Transport_WAN_Property_t::port_offset** (p. 185) field is included to allow compatibility with a parallel non-WAN transport.

The transport plugin honors the interface/multicast "white" and "black" lists specified in the NDDS_Transport_WAN_Property_t::parent::parent:

- **NDDS_Transport_Property_t::allow_interfaces_list**,
- **NDDS_Transport_Property_t::deny_interfaces_list**

The WAN transport can be configured to bind all sockets to a specified interface. This is required for consistent WAN communication on multi-homed systems.

- **NDDS_Transport_WAN_Property_t::interface_address** (p. 185)

The format of a string in these lists is assumed to be in standard IPv4 dot notation, possibly containing wildcards. For example:

- 10.10.30.232
- 10.10.*.*
- 192.168.1.*
- etc. Strings not in the correct format will be ignored.

Parameters:

default_network_address_out <<**out**>> Network address to be used when registering the transport.

property_in <<**in**>> Desired behavior of this transport, through the property field in **DDS_DomainParticipantQos**.

Returns:

A WAN Transport Plugin instance on success; or NULL on failure.

4.4.3.2 NDDS_Transport_Plugin* NDDS_Transport_WAN_new (const struct NDDS_Transport_WAN_Property_t * *property_in*)

Create an instance of a WAN Transport Plugin.

An application may create and register multiple instances of this Transport Plugin with RTI Connext. This may be to partition the network interfaces across multiple RTI Connext domains. However, note that the underlying transport, the operating system's IP layer, is still a "singleton". For example, if a unicast transport has already bound to a port, and another unicast transport tries to bind to the same port, the second attempt will fail.

The transport plugin honors the interface/multicast "white" and "black" lists specified in the **NDDS_Transport_WAN_Property_t::parent** (p. 184):

- **NDDS_Transport_Property_t::allow_interfaces_list**,
- **NDDS_Transport_Property_t::deny_interfaces_list**

The WAN transport can be configured to bind all sockets to a specified interface. This is required for consistent WAN communication on multi-homed systems.

- **NDDS_Transport_WAN_Property_t::interface_address** (p. 185)

The format of a string in these lists is assumed to be in standard IPv4 dot notation, possibly containing wildcards. For example:

- 10.10.30.232
- 10.10.*.*
- 192.168.1.*
- etc. Strings not in the correct format will be ignored.

Parameters:

property_in <<**in**>> Desired behavior of this transport. May be NULL for default property.

Returns:

A WAN Transport Plugin instance on success; or NULL on failure.

4.5 Secure Transport

Modules

- **Configure Secure Transport with Property QoS Policy**

  Predefined strings that can be used to configure the Secure Transport plugin through the Property QoS Policy of the DomainParticipant.

- **Secure Transport C API Reference**

  Basic types and macros provided by RTI Connext for use in the Secure Transport plugin interface.

4.5.1 Detailed Description

Data security is provided by wrapping all DDS network traffic with the Datagram Transport Layer Security (DTLS) protocol (IETF RFC 4347). DTLS is a relatively recent variant of the mature SSL/TLS family of protocols which adds the capability to secure communication over a connectionless network-layer transport such as UDP. UDP is the preferred network layer transport for the DDS wire protocol RTPS, as well as for NAT traversal. Like SSL/TLS, the DTLS protocol provides capabilities for certificate-based authentication, data encryption, and message integrity. The protocol specifies a number of standard cryptographic algorithms that must be available; the base set is listed in the TLS 1.1 specification (IETF RFC 4346).

Secure protocol support is provided by the open source OpenSSL library, which has supported the DTLS protocol since the release of OpenSSL 0.9.8. Note however that many critical issues in DTLS were resolved by the OpenSSL 0.9.8f release. For more detailed information about available ciphers, certificate support, etc. please refer to the OpenSSL documentation.

The DTLS protocol securely authenticates with each individual peer; as such, multicast communication is not supported by the Secure Transport.

4.5.2 Security Model

In order to communicate securely, an instance of the secure plugin requires: 1) a certificate authority (shared with all peers), 2) an identifying certificate which has been signed by the authority, 3) the private key associated with the public key contained in the certificate.

The Certificate Authority (CA) is specified by using a PEM format file containing its public key (**NDDS_Transport_TLS_Verification::ca_file** (p. 181)) or

by using a directory of PEM files following standard OpenSSL naming conventions (**NDDS_Transport_TLS_Verification::ca_path** (p. 182)). If a single CA file is used, it may contain multiple CA keys. In order to successfully communicate with a peer, the CA keys that are supplied must include the CA that has signed that peer's identifying certificate.

The identifying certificate is specified by using a PEM format file containing the chain of CAs used to authenticate the certificate. The identifying certificate must be signed by a CA. It will either be directly signed by a root CA (one of the CAs supplied above), by an authority whose certificate has been signed by the root CA, or by a longer chain of CA. The file must be sorted starting with the certificate to the highest level (root CA). If the certificate is directly signed by a root CA, then this file will only contain the root CA certificate followed by the identity certificate.

Finally, a private key is required. In order to avoid impersonation of an identity, this should be kept private. It can be stored in its own PEM file specified in one of the private key properties, or it can be appended to the certificate chain file.

One complication in the use of DTLS for communication by DDS is that even though DTLS is a connectionless protocol, it still has client/server semantics. The RTI Secure Transport maps a bidirectional communication channel between two peer applications into a pair of unidirectional encrypted channels. Both peers are playing the part of a client (when sending data) and a server (when receiving).

4.5.3 Certificate Support

Certificates can be generated using the supplied OpenSSL tool.

Initialize the Certificate Authority:

- create a copy of the openssl.cnf file and edit fields to specify the proper default names, paths
- create the required CA directory structure:

```
mkdir myCA
mkdir myCA/certs
mkdir myCA/private
mkdir myCA/newcerts
mkdir myCA/crl
touch myCA/index.txt
```

- create a self-signed certificate and CA private key:

```
openssl req -nodes -x509 -days 1095 -newkey rsa:2048 -keyout myCA/private/cakey.pem $\$
  -out myCA/cacert.pem -config openssl.cnf
```

For each identifying certificate:

- create a copy of the openssl.cnf file and edit fields for request template
- generate certificate request and private key

```
openssl req -nodes -new -newkey rsa:2048 -config template.cnf -keyout peer1key.pem $\$
  -out peer1req.pem
openssl ca -create_serial -config openssl.cnf -days 365 -in peer1req.pem $\$
  -out myCA/newcerts/peer1cert.pem
```

- create a certificate request
- optionally, append the private key to the peer certificate

```
cat myCA/newcerts/peer1cert.pem peer1key.pem >{private location}/peer1.pem
```

4.5.4 Parameters

To use the Secure Transport, the following parameters must be set:

- **NDDS_Transport_TLS_Verification::ca_file** (p. 181) and **NDDS_Transport_TLS_Verification::ca_path** (p. 182): the Certificate Authority certificate(s) that may be used to verify peers. At least one of these must be specified.
- **NDDS_Transport_TLS_Identity::certificate_chain_file** (p. 177): the identifying certificate to use to identify the instance of the transport. This file may also include the corresponding private key.
- **NDDS_Transport_TLS_Identity::private_key_file** (p. 178) or **NDDS_Transport_TLS_Identity::rsa_private_key_file** (p. 178): if the private key is not specified in the certificate chain file, one of these parameters must be specified.

These additional parameters may be needed:

- **NDDS_Transport_DTLS_Property_t::port_offset** (p. 172): an offset added to the DDS-internal port numbering scheme to ensure that an instance of the Secure transport can coexist with a non-Secure UDP transport.

- **NDDS_Transport_TLS_Ciphers::cipher_list** (p. 174): a list of allowed cipher types, in the standard OpenSSL format. For example, "AES256-SHA" (AES encryption with 256-bit key length; SHA1 digest) or "ALL:!ADH:!LOW:!EXPORT:!MD5:@STRENGTH" (all available ciphers ordered by strength excepting anonymous Diffie-Hellman, low key lengths, export weakened ciphers, and any using MD5 digest).
- **NDDS_Transport_TLS_Ciphers::dh_param_files** (p. 175): This must be supplied in order to use Diffie-Hellman key exchange.

4.6 Configure Secure Transport with Property QoS Policy

Predefined strings that can be used to configure the Secure Transport plugin through the Property QoS Policy of the DomainParticipant.

The following tables list the supported Secure Transport properties.

Property Name	Description	Required?
dds.transport.load_plugins	Comma-separated strings indicating the prefix names of all plugins that will be loaded by RTI Connext. Up to 8 plugins may be specified. For example, "dds.transport.DTLS.dtls1". In the following examples, *<DTLS_prefix>* indicates the string that is used as a prefix in the property names for all the settings that are related to this DTLS tranpsort plugin. *<DTLS_prefix>* must begin with "dds.transport." (such as "dds.transport.DTLS.dtls1").	**YES**
<DTLS_prefix>.library	Must set to **"libnddstransport-tls.so"** (for UNIX/Solaris) or **"nddstransport-tls.dll"** (for Windows). This library (and the dependent openssl libraries) needs to be in the path during run time for use by RTI Connext (in the **LD_LIBRARY_PATH** envionrment variable on UNIX/Solaris, in **PATH** for Windows)	**YES**
<DTLS_prefix>.create_function	Must be set to **"NDDS_Transport_DTLS_create"** .	**YES**
<DTLS_prefix>.aliases	Used to register the transport plugin returned by **NDDS_Transport_DTLS_create()** (p. 28) (as specified by ***<DTLS_prefix>*.create_function**) to the domain participant.	NO

4.7 Secure Transport C API Reference

Basic types and macros provided by RTI Connext for use in the Secure Transport plugin interface.

Modules

- **OpenSSL-Related Configuration**

  OpenSSL-Related Configuration.

Data Structures

- struct **NDDS_Transport_DTLS_Property_t**

  DTLS transport plugin property.

Defines

- #define **NDDS_TRANSPORT_DTLS_ADDRESS_BIT_COUNT** (32)

  Default value of **NDDS_Transport_Property_t::address_bit_count** *in* ***NDDS_Transport_DTLS_Property_t::parent*** *(*p. 170*).*

- #define **NDDS_TRANSPORT_DTLS_PROPERTIES_BITMAP_DEFAULT** (0)

  Default value of **NDDS_Transport_Property_t::properties_bitmap** *in* ***NDDS_Transport_DTLS_Property_t::parent*** *(*p. 170*).*

- #define **NDDS_TRANSPORT_DTLS_MESSAGE_SIZE_MAX_DEFAULT**

  Default value of ***NDDS_Transport_DTLS_Property_t::send_socket_buffer_size*** *(*p. 171*),* ***NDDS_Transport_DTLS_Property_t::recv_socket_buffer_size*** *(*p. 171*), and* ***NDDS_Transport_DTLS_Property_t::recv_decode_buffer_size*** *(*p. 172*).*

- #define **NDDS_TRANSPORT_DTLS_PROPERTY_DEFAULT**

  Use this to initialize a ***NDDS_Transport_DTLS_Property_t*** *(*p. 169*) structure.*

Functions

- NDDS_Transport_TLS_DllExport NDDS_Transport_Plugin * **NDDS_Transport_DTLS_create** (**NDDS_Transport_Address_t** *default_network_address_out, const struct **DDS_PropertyQosPolicy** *property_in)

  Create an instance of a DTLS Transport Plugin using PropertyQosPolicy (for dynamic loading).

- NDDS_Transport_TLS_DllExport NDDS_Transport_Plugin * **NDDS_Transport_DTLS_new** (const struct **NDDS_Transport_DTLS_Property_t** *property_in)

  Create an instance of a DTLS Transport Plugin.

4.7.1 Detailed Description

Basic types and macros provided by RTI Connext for use in the Secure Transport plugin interface.

4.7.2 Define Documentation

4.7.2.1 #define NDDS_TRANSPORT_DTLS_ADDRESS_BIT_COUNT (32)

Default value of **NDDS_Transport_Property_t::address_bit_count** in **NDDS_Transport_DTLS_Property_t::parent** (p. 170).

4.7.2.2 #define NDDS_TRANSPORT_DTLS_PROPERTIES_BITMAP_DEFAULT (0)

Default value of **NDDS_Transport_Property_t::properties_bitmap** in **NDDS_Transport_DTLS_Property_t::parent** (p. 170).

4.7.2.3 #define NDDS_TRANSPORT_DTLS_MESSAGE_SIZE_MAX_DEFAULT

Default value of **NDDS_Transport_DTLS_Property_t::send_socket_buffer_size** (p. 171), **NDDS_Transport_DTLS_Property_t::recv_socket_buffer_size** (p. 171), and **NDDS_Transport_DTLS_Property_t::recv_decode_buffer_size** (p. 172).

4.7.2.4 #define NDDS_TRANSPORT_DTLS_PROPERTY_DEFAULT

Value:

```
{ \
    { NDDS_TRANSPORT_CLASSID_DTLS, /* classid */ \
      NDDS_TRANSPORT_DTLS_ADDRESS_BIT_COUNT, /* address_bit_count */ \
      NDDS_TRANSPORT_DTLS_PROPERTIES_BITMAP_DEFAULT, /* properties_bitmap */ \
      NDDS_TRANSPORT_UDPV4_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT, /* gather_send_buffer_count_max */ \
      NDDS_TRANSPORT_DTLS_MESSAGE_SIZE_MAX_DEFAULT, /* message_size_max */ \
      NULL, 0, /* allow_interfaces_list */ \
      NULL, 0, /* deny_interfaces_list */ \
      NULL, 0, /* allow_multicast_interfaces_list */ \
      NULL, 0, /* deny_multicast_interfaces_list */ \
    }, /* parent (NDDS_Transport_Property_t) */ \
    NDDS_TRANSPORT_DTLS_MESSAGE_SIZE_MAX_DEFAULT, /* send_socket_buffer_size */ \
    NDDS_TRANSPORT_DTLS_MESSAGE_SIZE_MAX_DEFAULT, /* recv_socket_buffer_size */ \
    -1, /* ignore_loopback_interface (auto) */ \
    0, /* ignore_nonrunning_interfaces (do not ignore non-RUNNING) */ \
    0, /* transport_priority_mask */ \
    0, 0xff, /* transport_priority_mapping_low, transport_priority_mapping_high (no mapping to IP_TOS by de
    /* TLS-specific: */ \
    NDDS_TRANSPORT_DTLS_MESSAGE_SIZE_MAX_DEFAULT, /* recv_decode_buffer_size (actually want MTU) */ \
    144, /* port_offset */ \
    1000, /* dtls_handshake_resend_interval (1 second) */ \
    7, /* dtls_handshake_number_of_resends (last resend at ~60s) */ \
    60, /* dtls_connection_liveliness_interval (60x resend interval) */ \
    0, /* transport_mtu (don't set MTU) */ \
    NDDS_TRANSPORT_TLS_OPENSSL_CONFIGURATION_DEFAULT /* tls */ }
```

Use this to initialize a **NDDS_Transport_DTLS_Property_t** (p. 169) structure.

4.7.3 Function Documentation

4.7.3.1 NDDS_Transport_TLS_DllExport NDDS_Transport_Plugin* NDDS_Transport_DTLS_create (NDDS_Transport_Address_t * *default_network_address_out*, const struct DDS_PropertyQosPolicy * *property_in*)

Create an instance of a DTLS Transport Plugin using PropertyQosPolicy (for dynamic loading).

An application may create and register multiple instances of this Transport Plugin with RTI Connext. This may be to partition the network interfaces across multiple RTI Connext domains. However, note that the underlying transport, the operating system's IP layer, is still a "singleton". For example, if a unicast transport has already bound to a port, and another unicast transport tries to bind to the same port, the second attempt will fail. The **NDDS_Transport_-**

DTLS_Property_t::port_offset (p. 172) field is included to allow compatibility with a parallel non-secure transport.

The transport plugin honors the interface "white" and "black" lists specified in the **NDDS_Transport_DTLS_Property_t::parent** (p. 170):

- **NDDS_Transport_Property_t::allow_interfaces_list**,
- **NDDS_Transport_Property_t::deny_interfaces_list**

The format of a string in these lists is assumed to be in standard IPv4 dot notation, possibly containing wildcards. For example:

- 10.10.30.232
- 10.10.*.*
- 192.168.1.*
- etc. Strings not in the correct format will be ignored.

Parameters:

default_network_address_out <<**out**>> Network address to be used when registering the transport.

property_in <<**in**>> Desired behavior of this transport, through the property field in DomainParticipantQos.

Returns:

A DTLS Transport Plugin instance on success; or NULL on failure.

4.7.3.2 NDDS_Transport_TLS_DllExport NDDS_Transport_Plugin* NDDS_Transport_DTLS_new (const struct NDDS_Transport_DTLS_Property_t * *property_in*)

Create an instance of a DTLS Transport Plugin.

An application may create and register multiple instances of this Transport Plugin with RTI Connext. This may be to partition the network interfaces across multiple RTI Connext domains. However, note that the underlying transport, the operating system's IP layer, is still a "singleton". For example, if a unicast transport has already bound to a port, and another unicast transport tries to bind to the same port, the second attempt will fail. The **NDDS_Transport_DTLS_Property_t::port_offset** (p. 172) field is included to allow compatibility with a parallel non-secure transport.

The transport plugin honors the interface "white" and "black" lists specified in the **NDDS_Transport_DTLS_Property_t::parent** (p. 170):

- **NDDS_Transport_Property_t::allow_interfaces_list**,
- **NDDS_Transport_Property_t::deny_interfaces_list**

The format of a string in these lists is assumed to be in standard IPv4 dot notation, possibly containing wildcards. For example:

- 10.10.30.232
- 10.10.*.*
- 192.168.1.*
- etc. Strings not in the correct format will be ignored.

Parameters:

property_in <<**in**>> Desired behavior of this transport. May be NULL for default property.

Returns:

A DTLS Transport Plugin instance on success; or NULL on failure.

4.8 OpenSSL-Related Configuration

OpenSSL-Related Configuration.

Data Structures

- struct **NDDS_Transport_TLS_Verification**

  Set of TLS properties for certificate authorities (CAs) and verification.

- struct **NDDS_Transport_TLS_Identity**

  Set of TLS properties for identity.

- struct **NDDS_Transport_TLS_DHParamFile**

  Name of a Diffie-Helman (DH) key file and the length of the contained key in bits.

- struct **NDDS_Transport_TLS_Ciphers**

  Set of TLS properties for cipher configuration.

- struct **NDDS_Transport_TLS_OpenSSL_Configuration**

  Full set of TLS properties.

Defines

- #define **NDDS_TRANSPORT_TLS_VERIFY_DEFAULT**

  Use this to initialize a NDDS_Transport_TLS_Verfication structure.

- #define **NDDS_TRANSPORT_TLS_IDENTITY_DEFAULT**

  *Use this to initialize a **NDDS_Transport_TLS_Identity** (p. 177) structure.*

- #define **NDDS_TRANSPORT_TLS_CIPHER_DEFAULT**

  Use this to initialize a NDDS_Transport_TLS_Chiphers structure.

- #define **NDDS_TRANSPORT_TLS_OPENSSL_CONFIGURATION_DEFAULT**

  *Use this to initialize a **NDDS_Transport_TLS_OpenSSL_Configuration** (p. 179) structure.*

Typedefs

- typedef int(* **NDDS_Transport_TLS_Verify_Callback**)(int preverify_ok, X509_STORE_CTX *x509_ctx)

  Callback used to verify peer certificates.

Functions

- NDDS_Transport_TLS_DllExport void **NDDS_Transport_TLS_thread_exit** ()

  clean up OpenSSL resources for current thread (call before exit)

- int **NDDS_Transport_TLS_default_verify_callback** (int ok, X509_STORE_CTX *store)

  Default verify callback: log errors when verification fails.

- int **NDDS_Transport_TLS_verbose_verify_callback** (int ok, X509_STORE_CTX *store)

  Verbose verify callback: log information about successful verification as well as errors when verification fails.

4.8.1 Detailed Description

OpenSSL-Related Configuration.

The DTLS security components are implemented by the OpenSSL library. Proper DTLS support requires at least version 0.9.8f of OpenSSL.

4.8.2 Define Documentation

4.8.2.1 #define NDDS_TRANSPORT_TLS_VERIFY_DEFAULT

Value:

```
{ \
    NULL, NULL, /* ca_file, ca_path */ \
    -1, /* verify_depth (no depth limit) */ \
    0, /* verify_peer (NOT mutual) */ \
    NULL /* callback (use default verify callback) */ }
```

Use this to initialize a NDDS_Transport_TLS_Verfication structure.

4.8.2.2 #define NDDS_TRANSPORT_TLS_IDENTITY_DEFAULT

Value:

```
{ \
    NULL, /* certificate_chain_file */ \
    NULL, /* private_key_password */ \
    NULL, /* private_key_file */ \
    NULL /* rsa_private_key_file */ }
```

Use this to initialize a **NDDS_Transport_TLS_Identity** (p. 177) structure.

4.8.2.3 #define NDDS_TRANSPORT_TLS_CIPHER_DEFAULT

Value:

```
{ \
    NULL, /* cipher_list (default cipher list) */ \
    0, NULL, /* dh_param_files_length, dh_param_files (no DH params) */ \
    NULL, /* engine_id (no engine) */ \
    0, NULL, NULL, /* engine_pre_cmd_length, engine_pre_cmd_names, engine_pre_cmd_parameters */ \
    0, NULL, NULL /* engine_post_cmd_length, engine_post_cmd_names, engeine_post_cmd_parameters */ }
```

Use this to initialize a NDDS_Transport_TLS_Chiphers structure.

4.8.2.4 #define NDDS_TRANSPORT_TLS_OPENSSL_CONFIGURATION_DEFAULT

Value:

```
{ \
    NDDS_TRANSPORT_TLS_VERIFY_DEFAULT, /* verify */ \
    NDDS_TRANSPORT_TLS_IDENTITY_DEFAULT, /* identity */ \
    NDDS_TRANSPORT_TLS_CIPHER_DEFAULT, /* cipher */ \
    NDDS_TRANSPORT_TLS_RENEGOTIATE_DEFAULT /* renegotiate */ }
```

Use this to initialize a **NDDS_Transport_TLS_OpenSSL_Configuration** (p. 179) structure.

4.8.3 Typedef Documentation

4.8.3.1 typedef int(* NDDS_Transport_TLS_Verify_Callback)(int preverify_ok, X509_STORE_CTX *x509_ctx)

Callback used to verify peer certificates.

See the OpenSSL manual page for SSL_CTX_set_verify for more information.

4.8.4 Function Documentation

4.8.4.1 NDDS_Transport_TLS_DllExport void NDDS_Transport_TLS_thread_exit ()

clean up OpenSSL resources for current thread (call before exit)

4.8.4.2 int NDDS_Transport_TLS_default_verify_callback (int *ok*, X509_STORE_CTX * *store*)

Default verify callback: log errors when verification fails.

See the OpenSSL manual page for SSL_CTX_set_verify for more information.

4.8.4.3 int NDDS_Transport_TLS_verbose_verify_callback (int *ok*, X509_STORE_CTX * *store*)

Verbose verify callback: log information about successful verification as well as errors when verification fails.

See the OpenSSL manual page for SSL_CTX_set_verify for more information.

4.9 Secure WAN Transport Examples

Secure WAN Transport Examples.

Modules

- **C Example**

  Secure WAN Transport Example Using C.

- **C++ Example**

  Secure WAN Transport Example Using C++.

- **Java Example**

  Secure WAN Transport Example Using Java.

4.9.1 Detailed Description

Secure WAN Transport Examples.

4.10 C Example

Secure WAN Transport Example Using C.

Modules

- **HelloWorld.idl**
- **HelloWorld.c**
- **HelloWorld_publisher.c**
- **HelloWorld_subscriber.c**
- **HelloWorldPlugin.c**
- **HelloWorldSupport.c**

4.10.1 Detailed Description

Secure WAN Transport Example Using C.

4.11 HelloWorld.idl

[$(NDDSHOME)/example/C/helloWorldWAN/HelloWorld.idl]

```
struct HelloWorld {
    string<128> msg;
};
```

4.12 HelloWorld.c

[$(NDDSHOME)/example/C/helloWorldWAN/HelloWorld.h]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from HelloWorld.idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/

#ifndef HelloWorld_1436885487_h
#define HelloWorld_1436885487_h

#ifndef NDDS_STANDALONE_TYPE
    #ifdef __cplusplus
        #ifndef ndds_cpp_h
            #include "ndds/ndds_cpp.h"
        #endif
    #else
        #ifndef ndds_c_h
            #include "ndds/ndds_c.h"
        #endif
    #endif
#else
    #include "ndds_standalone_type.h"
#endif


#ifdef __cplusplus
extern "C" {
#endif


extern const char *HelloWorldTYPENAME;


#ifdef __cplusplus
}
#endif


#ifdef __cplusplus
    struct HelloWorldSeq;

#ifndef NDDS_STANDALONE_TYPE
    class HelloWorldTypeSupport;
    class HelloWorldDataWriter;
    class HelloWorldDataReader;
#endif

#endif
```

```
class HelloWorld
{
public:
#ifdef __cplusplus
    typedef struct HelloWorldSeq Seq;

#ifndef NDDS_STANDALONE_TYPE
    typedef HelloWorldTypeSupport TypeSupport;
    typedef HelloWorldDataWriter DataWriter;
    typedef HelloWorldDataReader DataReader;
#endif

#endif

    char*  msg; /* maximum length = (128) */


};


#if (defined(RTI_WIN32) || defined (RTI_WINCE)) && defined(NDDS_USER_DLL_EXPORT)
  /* If the code is building on Windows, start exporting symbols.
   */
  #undef NDDSUSERDllExport
  #define NDDSUSERDllExport __declspec(dllexport)
#endif


NDDSUSERDllExport DDS_TypeCode* HelloWorld_get_typecode(void); /* Type code */


DDS_SEQUENCE(HelloWorldSeq, HelloWorld);

NDDSUSERDllExport
RTIBool HelloWorld_initialize(
        HelloWorld* self);

NDDSUSERDllExport
RTIBool HelloWorld_initialize_ex(
        HelloWorld* self,RTIBool allocatePointers,RTIBool allocateMemory);

NDDSUSERDllExport
void HelloWorld_finalize(
        HelloWorld* self);

NDDSUSERDllExport
void HelloWorld_finalize_ex(
        HelloWorld* self,RTIBool deletePointers);

NDDSUSERDllExport
RTIBool HelloWorld_copy(
        HelloWorld* dst,
        const HelloWorld* src);

#if (defined(RTI_WIN32) || defined (RTI_WINCE)) && defined(NDDS_USER_DLL_EXPORT)
  /* If the code is building on Windows, stop exporting symbols.
```

```
   */
  #undef NDDSUSERDllExport
  #define NDDSUSERDllExport
#endif



#endif /* HelloWorld_1436885487_h */
```

[$(NDDSHOME)/example/C/helloWorldWAN/HelloWorld.c]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from HelloWorld.idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/


#ifndef NDDS_STANDALONE_TYPE
    #ifdef __cplusplus
        #ifndef ndds_cpp_h
            #include "ndds/ndds_cpp.h"
        #endif
        #ifndef dds_c_log_impl_h
            #include "dds_c/dds_c_log_impl.h"
        #endif
    #else
        #ifndef ndds_c_h
            #include "ndds/ndds_c.h"
        #endif
    #endif

    #ifndef cdr_type_h
        #include "cdr/cdr_type.h"
    #endif

    #ifndef osapi_heap_h
        #include "osapi/osapi_heap.h"
    #endif
#else
    #include "ndds_standalone_type.h"
#endif



#include "HelloWorld.h"

/* ========================================================================= */
const char *HelloWorldTYPENAME = "HelloWorld";

DDS_TypeCode* HelloWorld_get_typecode()
{
    static RTIBool is_initialized = RTI_FALSE;
```

```
    static DDS_TypeCode HelloWorld_g_tc_msg_string = DDS_INITIALIZE_STRING_TYPECODE(128);

    static DDS_TypeCode_Member HelloWorld_g_tc_members[1]=
    {
        {
            (char *)"msg",/* Member name */
            {
                0,/* Representation ID */
                DDS_BOOLEAN_FALSE,/* Is a pointer? */
                -1, /* Bitfield bits */
                NULL/* Member type code is assigned later */
            },
            0, /* Ignored */
            0, /* Ignored */
            0, /* Ignored */
            NULL, /* Ignored */
            DDS_BOOLEAN_FALSE, /* Is a key? */
            DDS_PRIVATE_MEMBER,/* Ignored */
            0,/* Ignored */
            NULL/* Ignored */
        }
    };

    static DDS_TypeCode HelloWorld_g_tc =
    {{
        DDS_TK_STRUCT,/* Kind */
        DDS_BOOLEAN_FALSE, /* Ignored */
        -1,/* Ignored */
        (char *)"HelloWorld", /* Name */
        NULL, /* Ignored */
        0, /* Ignored */
        0, /* Ignored */
        NULL, /* Ignored */
        1, /* Number of members */
        HelloWorld_g_tc_members, /* Members */
        DDS_VM_NONE /* Ignored */
    }}; /* Type code for HelloWorld*/

    if (is_initialized) {
        return &HelloWorld_g_tc;
    }


    HelloWorld_g_tc_members[0]._representation._typeCode = (RTICdrTypeCode *)&HelloWorld_g_tc_msg_string;

    is_initialized = RTI_TRUE;

    return &HelloWorld_g_tc;
}


RTIBool HelloWorld_initialize(
    HelloWorld* sample) {
  return HelloWorld_initialize_ex(sample,RTI_TRUE,RTI_TRUE);
}
```

```
RTIBool HelloWorld_initialize_ex(
    HelloWorld* sample,RTIBool allocatePointers,RTIBool allocateMemory)
{


    if (allocatePointers) {} /* To avoid warnings */
    if (allocateMemory) {} /* To avoid warnings */

    if (allocateMemory) {
        sample->msg = DDS_String_alloc((128));
        if (sample->msg == NULL) {
            return RTI_FALSE;
        }
    } else {
        if (sample->msg != NULL) {
            sample->msg[0] = '\0';
        }
    }



    return RTI_TRUE;
}

void HelloWorld_finalize(
    HelloWorld* sample)
{
    HelloWorld_finalize_ex(sample,RTI_TRUE);
}

void HelloWorld_finalize_ex(
    HelloWorld* sample,RTIBool deletePointers)
{
    if (sample) { } /* To avoid warnings */
    if (deletePointers) {} /* To avoid warnings */


    DDS_String_free(sample->msg);


}

RTIBool HelloWorld_copy(
    HelloWorld* dst,
    const HelloWorld* src)
{

    if (!RTICdrType_copyString(
        dst->msg, src->msg, (128) + 1)) {
        return RTI_FALSE;
    }



    return RTI_TRUE;
}
```

```
#define T HelloWorld
#define TSeq HelloWorldSeq
#define T_initialize_ex HelloWorld_initialize_ex
#define T_finalize_ex   HelloWorld_finalize_ex
#define T_copy       HelloWorld_copy

#ifndef NDDS_STANDALONE_TYPE
#include "dds_c/generic/dds_c_sequence_TSeq.gen"
#ifdef __cplusplus
#include "dds_cpp/generic/dds_cpp_sequence_TSeq.gen"
#endif
#else
#include "dds_c_sequence_TSeq.gen"
#ifdef __cplusplus
#include "dds_cpp_sequence_TSeq.gen"
#endif
#endif

#undef T_copy
#undef T_finalize_ex
#undef T_initialize_ex
#undef TSeq
#undef T
```

4.13 HelloWorld_publisher.c

[$(NDDSHOME)/example/C/helloWorldWAN/HelloWorld_publisher.c]

```
/* HelloWorld_publisher.c

   A publication of data of type HelloWorld

   This file is derived from code automatically generated by the rtiddsgen
   command:

   rtiddsgen -language C -example <arch> HelloWorld.idl

   Example publication of type HelloWorld automatically generated by
   'rtiddsgen'. To test them follow these steps:

   (1) Compile this file and the example subscription.

   (2) Start the subscription on the same domain used for RTI Connext with the command
       objs/<arch>/HelloWorld_subscriber <domain_id> <sample_count>

   (3) Start the publication on the same domain used for RTI Connext with the command
       objs/<arch>/HelloWorld_publisher <domain_id> <sample_count>

   (4) [Optional] Specify the list of discovery initial peers and
       multicast receive addresses via an environment variable or a file
       (in the current working directory) called NDDS_DISCOVERY_PEERS.

   You can run any number of publishers and subscribers programs, and can
   add and remove them dynamically from the domain.


   Example:

       To run the example application on domain <domain_id>:

       On Unix:

       objs/<arch>/HelloWorld_publisher <domain_id>
       objs/<arch>/HelloWorld_subscriber <domain_id>

       On Windows:

       objs\<arch>\HelloWorld_publisher <domain_id>
       objs\<arch>\HelloWorld_subscriber <domain_id>


modification history
------------ -------
*/

#include <stdio.h>
#include <stdlib.h>
#include "ndds/ndds_c.h"
#include "HelloWorld.h"
#include "HelloWorldSupport.h"
```

```
/* define this to enable security using DTLS */
/*#define USE_SECURITY*/

/* set this value to the address of your WAN Rendezvous server */
#define WAN_SERVER "127.0.0.1"

/* your publisher and subscriber should have unique IDs */
#define WAN_ID "1"

/* Delete all entities */
static int publisher_shutdown(
    DDS_DomainParticipant *participant)
{
    DDS_ReturnCode_t retcode;
    int status = 0;

    if (participant != NULL) {
        retcode = DDS_DomainParticipant_delete_contained_entities(participant);
        if (retcode != DDS_RETCODE_OK) {
            printf("delete_contained_entities error %d\n", retcode);
            status = -1;
        }

        retcode = DDS_DomainParticipantFactory_delete_participant(
            DDS_TheParticipantFactory, participant);
        if (retcode != DDS_RETCODE_OK) {
            printf("delete_participant error %d\n", retcode);
            status = -1;
        }
    }

    /* RTI Connext provides finalize_instance() method on
       domain participant factory for people who want to release memory used
       by the participant factory singleton. Uncomment the following block of
       code for clean destruction of the singleton. */
/*
    retcode = DDS_DomainParticipantFactory_finalize_instance();
    if (retcode != DDS_RETCODE_OK) {
        printf("finalize_instance error %d\n", retcode);
        status = -1;
    }
*/

    return status;
}

int publisher_main(int domainId, int sample_count)
{
    DDS_DomainParticipant *participant = NULL;
    DDS_Publisher *publisher = NULL;
    DDS_Topic *topic = NULL;
    DDS_DataWriter *writer = NULL;
    HelloWorldDataWriter *HelloWorld_writer = NULL;
    HelloWorld *instance = NULL;
    DDS_ReturnCode_t retcode;
    DDS_InstanceHandle_t instance_handle = DDS_HANDLE_NIL;
```

```
    const char *type_name = NULL;
    int count = 0;
    struct DDS_Duration_t send_period = {4,0};

    struct DDS_DomainParticipantQos participant_qos =
        DDS_DomainParticipantQos_INITIALIZER;

    /* Get default participant QoS from participant factory */
    retcode = DDS_DomainParticipantFactory_get_default_participant_qos(
        DDS_TheParticipantFactory, &participant_qos);
    if (retcode != DDS_RETCODE_OK) {
        printf("could not get default participant qos\n");
        publisher_shutdown(participant);
        return -1;
    }

    /* Disable builtin transports */
    participant_qos.transport_builtin.mask = DDS_TRANSPORTBUILTIN_MASK_NONE;

    /* Set up property Qos to load plugin */
    retcode = DDS_PropertyQosPolicyHelper_add_property(
        &participant_qos.property,
        "dds.transport.load_plugins", "dds.transport.wan_plugin.wan",
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.load_plugins\n");
        publisher_shutdown(participant);
    }

    /* library */
    retcode = DDS_PropertyQosPolicyHelper_add_property(
        &participant_qos.property,
        "dds.transport.wan_plugin.wan.library",
#ifdef RTI_WIN32
        "nddstransportwan.dll",
#else
        "libnddstransportwan.so",
#endif
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.library\n");
        publisher_shutdown(participant);
    }

    /* create function */
    retcode = DDS_PropertyQosPolicyHelper_add_property(
        &participant_qos.property,
        "dds.transport.wan_plugin.wan.create_function",
        "NDDS_Transport_WAN_create",
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.create_function\n");
        publisher_shutdown(participant);
    }

    /* plugin properties */
#ifdef USE_SECURITY
```

```
    printf("Enabling secure WAN transport\n");
    retcode = DDS_PropertyQosPolicyHelper_add_property(
        &participant_qos.property,
        "dds.transport.wan_plugin.wan.enable_security", "1",
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.enable_security\n");
        publisher_shutdown(participant);
    }
    retcode = DDS_PropertyQosPolicyHelper_add_property(
        &participant_qos.property,
        "dds.transport.wan_plugin.wan.tls.verify.ca_file", "cacert.pem",
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.tls.verify.ca_file\n");
        publisher_shutdown(participant);
    }
    retcode = DDS_PropertyQosPolicyHelper_add_property(
        &participant_qos.property,
        "dds.transport.wan_plugin.wan.tls.identity.certificate_chain_file",
        "peer1.pem", DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.tls.identity.certificate_chain_file\n");
        publisher_shutdown(participant);
    }
#endif /* USE_SECURITY */

    retcode = DDS_PropertyQosPolicyHelper_add_property(
        &participant_qos.property,
        "dds.transport.wan_plugin.wan.server", WAN_SERVER,
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.server\n");
        publisher_shutdown(participant);
    }

    retcode = DDS_PropertyQosPolicyHelper_add_property(
        &participant_qos.property,
        "dds.transport.wan_plugin.wan.transport_instance_id", WAN_ID,
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.transport_instance_id\n");
        publisher_shutdown(participant);
    }

    participant = DDS_DomainParticipantFactory_create_participant(
        DDS_TheParticipantFactory, domainId, &participant_qos,
        NULL /* listener */, DDS_STATUS_MASK_NONE);
    if (participant == NULL) {
        printf("create_participant error\n");
        publisher_shutdown(participant);
        return -1;
    }

    /* To customize publisher QoS, use
       DDS_DomainParticipant_get_default_publisher_qos() */
    publisher = DDS_DomainParticipant_create_publisher(
```

```
    participant, &DDS_PUBLISHER_QOS_DEFAULT, NULL /* listener */,
    DDS_STATUS_MASK_NONE);
if (publisher == NULL) {
    printf("create_publisher error\n");
    publisher_shutdown(participant);
    return -1;
}

/* Register type before creating topic */
type_name = HelloWorldTypeSupport_get_type_name();
retcode = HelloWorldTypeSupport_register_type(
    participant, type_name);
if (retcode != DDS_RETCODE_OK) {
    printf("register_type error %d\n", retcode);
    publisher_shutdown(participant);
    return -1;
}

/* To customize topic QoS, use
   DDS_DomainParticipant_get_default_topic_qos() */
topic = DDS_DomainParticipant_create_topic(
    participant, "Example HelloWorld",
    type_name, &DDS_TOPIC_QOS_DEFAULT, NULL /* listener */,
    DDS_STATUS_MASK_NONE);
if (topic == NULL) {
    printf("create_topic error\n");
    publisher_shutdown(participant);
    return -1;
}

/* To customize data writer QoS, use
   DDS_Publisher_get_default_datawriter_qos() */
writer = DDS_Publisher_create_datawriter(
    publisher, topic,
    &DDS_DATAWRITER_QOS_DEFAULT, NULL /* listener */, DDS_STATUS_MASK_NONE);
if (writer == NULL) {
    printf("create_datawriter error\n");
    publisher_shutdown(participant);
    return -1;
}
HelloWorld_writer = HelloWorldDataWriter_narrow(writer);
if (HelloWorld_writer == NULL) {
    printf("DataWriter narrow error\n");
    publisher_shutdown(participant);
    return -1;
}

/* Create data sample for writing */
instance = HelloWorldTypeSupport_create_data_ex(DDS_BOOLEAN_TRUE);
if (instance == NULL) {
    printf("HelloWorldTypeSupport_create_data error\n");
    publisher_shutdown(participant);
    return -1;
}

/* For data type that has key, if the same instance is going to be
   written multiple times, initialize the key here
```

```
       and register the keyed instance prior to writing */
/*
    instance_handle = HelloWorldDataWriter_register_instance(
        HelloWorld_writer, instance);
*/

    /* Main loop */
    for (count=0; (sample_count == 0) || (count < sample_count); ++count) {

        printf("Writing HelloWorld to WAN, count %d\n", count);

        /* Modify the data to be written here */
        sprintf(instance->msg, "Hello Wide Area World! (%d)", count);

        /* Write data */
        retcode = HelloWorldDataWriter_write(
            HelloWorld_writer, instance, &instance_handle);
        if (retcode != DDS_RETCODE_OK) {
            printf("write error %d\n", retcode);
        }

        NDDS_Utility_sleep(&send_period);
    }

/*
    retcode = HelloWorldDataWriter_unregister_instance(
        HelloWorld_writer, instance, &instance_handle);
    if (retcode != DDS_RETCODE_OK) {
        printf("unregister instance error %d\n", retcode);
    }
*/

    /* Delete data sample */
    retcode = HelloWorldTypeSupport_delete_data_ex(instance, DDS_BOOLEAN_TRUE);
    if (retcode != DDS_RETCODE_OK) {
        printf("HelloWorldTypeSupport_delete_data error %d\n", retcode);
    }

    /* Cleanup and delete delete all entities */
    DDS_DomainParticipantQos_finalize(&participant_qos);
    return publisher_shutdown(participant);
}

#if defined(RTI_WINCE)
int wmain(int argc, wchar_t** argv)
{
    int domainId = 0;
    int sample_count = 0; /* infinite loop */

    if (argc >= 2) {
        domainId = _wtoi(argv[1]);
    }
    if (argc >= 3) {
        sample_count = _wtoi(argv[2]);
    }

    /* Uncomment this to turn on additional logging
```

```
    NDDS_Config_Logger_set_verbosity_by_category(
        NDDS_Config_Logger_get_instance(),
        NDDS_CONFIG_LOG_CATEGORY_API,
        NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
    */

    return publisher_main(domainId, sample_count);
}
#elif !(defined(RTI_VXWORKS) && !defined(__RTP__)) && !defined(RTI_PSOS)
int main(int argc, char *argv[])
{
    int domainId = 0;
    int sample_count = 0; /* infinite loop */

    if (argc >= 2) {
        domainId = atoi(argv[1]);
    }
    if (argc >= 3) {
        sample_count = atoi(argv[2]);
    }

    /* Uncomment this to turn on additional logging
    NDDS_Config_Logger_set_verbosity_by_category(
        NDDS_Config_Logger_get_instance(),
        NDDS_CONFIG_LOG_CATEGORY_API,
        NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
    */

    return publisher_main(domainId, sample_count);
}
#endif
```

4.14 HelloWorld_subscriber.c

[$(NDDSHOME)/example/C/helloWorldWAN/HelloWorld_subscriber.c]

```
/* HelloWorld_subscriber.c

   A subscription example

   This file is derived from code automatically generated by the rtiddsgen
   command:

   rtiddsgen -language C -example <arch> HelloWorld.idl

   Example subscription of type HelloWorld automatically generated by
   'rtiddsgen'. To test them follow these steps:

   (1) Compile this file and the example publication.

   (2) Start the subscription on the same domain used for RTI Connext  with the command
       objs/<arch>/HelloWorld_subscriber <domain_id> <sample_count>

   (3) Start the publication on the same domain used for RTI Connext with the command
       objs/<arch>/HelloWorld_publisher <domain_id> <sample_count>

   (4) [Optional] Specify the list of discovery initial peers and
       multicast receive addresses via an environment variable or a file
       (in the current working directory) called NDDS_DISCOVERY_PEERS.

   You can run any number of publishers and subscribers programs, and can
   add and remove them dynamically from the domain.


   Example:

       To run the example application on domain <domain_id>:

       On Unix:

       objs/<arch>/HelloWorld_publisher <domain_id>
       objs/<arch>/HelloWorld_subscriber <domain_id>

       On Windows:

       objs\<arch>\HelloWorld_publisher <domain_id>
       objs\<arch>\HelloWorld_subscriber <domain_id>


modification history
------------ -------
*/

#include <stdio.h>
#include <stdlib.h>
#include "ndds/ndds_c.h"
#include "HelloWorld.h"
#include "HelloWorldSupport.h"
```

```
/* set this to enable security using DTLS */
/*#define USE_SECURITY*/

/* set this value to the address of your WAN Rendezvous server */
#define WAN_SERVER "127.0.0.1"

/* your publisher and subscriber should have unique IDs */
#define WAN_ID "2"

void HelloWorldListener_on_requested_deadline_missed(
    void* listener_data,
    DDS_DataReader* reader,
    const struct DDS_RequestedDeadlineMissedStatus *status)
{
}

void HelloWorldListener_on_requested_incompatible_qos(
    void* listener_data,
    DDS_DataReader* reader,
    const struct DDS_RequestedIncompatibleQosStatus *status)
{
}

void HelloWorldListener_on_sample_rejected(
    void* listener_data,
    DDS_DataReader* reader,
    const struct DDS_SampleRejectedStatus *status)
{
}

void HelloWorldListener_on_liveliness_changed(
    void* listener_data,
    DDS_DataReader* reader,
    const struct DDS_LivelinessChangedStatus *status)
{
}

void HelloWorldListener_on_sample_lost(
    void* listener_data,
    DDS_DataReader* reader,
    const struct DDS_SampleLostStatus *status)
{
}

void HelloWorldListener_on_subscription_matched(
    void* listener_data,
    DDS_DataReader* reader,
    const struct DDS_SubscriptionMatchedStatus *status)
{
}

void HelloWorldListener_on_data_available(
    void* listener_data,
    DDS_DataReader* reader)
{
    HelloWorldDataReader *HelloWorld_reader = NULL;
```

```
    struct HelloWorldSeq data_seq = DDS_SEQUENCE_INITIALIZER;
    struct DDS_SampleInfoSeq info_seq = DDS_SEQUENCE_INITIALIZER;
    DDS_ReturnCode_t retcode;
    int i;

    HelloWorld_reader = HelloWorldDataReader_narrow(reader);
    if (HelloWorld_reader == NULL) {
        printf("DataReader narrow error\n");
        return;
    }

    retcode = HelloWorldDataReader_take(
        HelloWorld_reader,
        &data_seq, &info_seq, DDS_LENGTH_UNLIMITED,
        DDS_ANY_SAMPLE_STATE, DDS_ANY_VIEW_STATE, DDS_ANY_INSTANCE_STATE);
    if (retcode == DDS_RETCODE_NO_DATA) {
        return;
    } else if (retcode != DDS_RETCODE_OK) {
        printf("take error %d\n", retcode);
        return;
    }

    for (i = 0; i < HelloWorldSeq_get_length(&data_seq); ++i) {
        if (DDS_SampleInfoSeq_get_reference(&info_seq, i)->valid_data) {
            HelloWorldTypeSupport_print_data(
                HelloWorldSeq_get_reference(&data_seq, i));
        }
    }

    retcode = HelloWorldDataReader_return_loan(
        HelloWorld_reader,
        &data_seq, &info_seq);
    if (retcode != DDS_RETCODE_OK) {
        printf("return loan error %d\n", retcode);
    }
}

/* Delete all entities */
static int subscriber_shutdown(
    DDS_DomainParticipant *participant)
{
    DDS_ReturnCode_t retcode;
    int status = 0;

    if (participant != NULL) {
        retcode = DDS_DomainParticipant_delete_contained_entities(participant);
        if (retcode != DDS_RETCODE_OK) {
            printf("delete_contained_entities error %d\n", retcode);
            status = -1;
        }

        retcode = DDS_DomainParticipantFactory_delete_participant(
            DDS_TheParticipantFactory, participant);
        if (retcode != DDS_RETCODE_OK) {
            printf("delete_participant error %d\n", retcode);
            status = -1;
        }
```

```
    }

    /* RTI Connext provides finalize_instance() method on
       domain participant factory for people who want to release memory used
       by the participant factory singleton. Uncomment the following block of
       code for clean destruction of the singleton. */
/*
    retcode = DDS_DomainParticipantFactory_finalize_instance();
    if (retcode != DDS_RETCODE_OK) {
        printf("finalize_instance error %d\n", retcode);
        status = -1;
    }
*/

    return status;
}

int subscriber_main(int domainId, int sample_count)
{
    DDS_DomainParticipant *participant = NULL;
    DDS_Subscriber *subscriber = NULL;
    DDS_Topic *topic = NULL;
    struct DDS_DataReaderListener reader_listener =
        DDS_DataReaderListener_INITIALIZER;
    DDS_DataReader *reader = NULL;
    DDS_ReturnCode_t retcode;
    const char *type_name = NULL;
    int count = 0;
    struct DDS_Duration_t poll_period = {4,0};


    struct DDS_DomainParticipantQos participant_qos =
        DDS_DomainParticipantQos_INITIALIZER;

    /* Get default participant QoS from participant factory */
    retcode = DDS_DomainParticipantFactory_get_default_participant_qos(
        DDS_TheParticipantFactory, &participant_qos);
    if (retcode != DDS_RETCODE_OK) {
        printf("could not get default participant qos\n");
        subscriber_shutdown(participant);
        return -1;
    }

    /* Disable builtin transports */
    participant_qos.transport_builtin.mask = DDS_TRANSPORTBUILTIN_MASK_NONE;

    /* Set up property Qos to load plugin */
    retcode = DDS_PropertyQosPolicyHelper_add_property(
        &participant_qos.property,
        "dds.transport.load_plugins", "dds.transport.wan_plugin.wan",
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.load_plugins\n");
        subscriber_shutdown(participant);
    }

    /* library */
```

```
    retcode = DDS_PropertyQosPolicyHelper_add_property(
        &participant_qos.property,
        "dds.transport.wan_plugin.wan.library",
#ifdef RTI_WIN32
        "nddstransportwan.dll",
#else
        "libnddstransportwan.so",
#endif
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.library\n");
        subscriber_shutdown(participant);
    }

    /* create function */
    retcode = DDS_PropertyQosPolicyHelper_add_property(
        &participant_qos.property,
        "dds.transport.wan_plugin.wan.create_function",
        "NDDS_Transport_WAN_create",
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.create_function\n");
        subscriber_shutdown(participant);
    }

    /* plugin properties */
#ifdef USE_SECURITY
    printf("Enabling secure WAN transport\n");
    retcode = DDS_PropertyQosPolicyHelper_add_property(
        &participant_qos.property,
        "dds.transport.wan_plugin.wan.enable_security", "1",
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.enable_security\n");
        subscriber_shutdown(participant);
    }
    retcode = DDS_PropertyQosPolicyHelper_add_property(
        &participant_qos.property,
        "dds.transport.wan_plugin.wan.tls.verify.ca_file", "cacert.pem",
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.tls.verify.ca_file\n");
        subscriber_shutdown(participant);
    }
    retcode = DDS_PropertyQosPolicyHelper_add_property(
        &participant_qos.property,
        "dds.transport.wan_plugin.wan.tls.identity.certificate_chain_file",
        "peer2.pem", DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.tls.identity.certificate_chain_file\n");
        subscriber_shutdown(participant);
    }
#endif /* USE_SECURITY */

    retcode = DDS_PropertyQosPolicyHelper_add_property(
        &participant_qos.property,
        "dds.transport.wan_plugin.wan.server", WAN_SERVER,
```

```
    DDS_BOOLEAN_FALSE);
if (retcode != DDS_RETCODE_OK) {
    printf("Failed to add property dds.transport.wan_plugin.wan.server\n");
    subscriber_shutdown(participant);
}

retcode = DDS_PropertyQosPolicyHelper_add_property(
    &participant_qos.property,
    "dds.transport.wan_plugin.wan.transport_instance_id", WAN_ID,
    DDS_BOOLEAN_FALSE);
if (retcode != DDS_RETCODE_OK) {
    printf("Failed to add property dds.transport.wan_plugin.wan.transport_instance_id\n");
    subscriber_shutdown(participant);
}

participant = DDS_DomainParticipantFactory_create_participant(
    DDS_TheParticipantFactory, domainId, &participant_qos,
    NULL /* listener */, DDS_STATUS_MASK_NONE);
if (participant == NULL) {
    printf("create_participant error\n");
    subscriber_shutdown(participant);
    return -1;
}

/* To customize subscriber QoS, use
   DDS_DomainParticipant_get_default_subscriber_qos() */
subscriber = DDS_DomainParticipant_create_subscriber(
    participant, &DDS_SUBSCRIBER_QOS_DEFAULT, NULL /* listener */,
    DDS_STATUS_MASK_NONE);
if (subscriber == NULL) {
    printf("create_subscriber error\n");
    subscriber_shutdown(participant);
    return -1;
}

/* Register type before creating topic */
type_name = HelloWorldTypeSupport_get_type_name();
retcode = HelloWorldTypeSupport_register_type(participant, type_name);
if (retcode != DDS_RETCODE_OK) {
    printf("register_type error %d\n", retcode);
    subscriber_shutdown(participant);
    return -1;
}

/* To customize topic QoS, use
   DDS_DomainParticipant_get_default_topic_qos() */
topic = DDS_DomainParticipant_create_topic(
    participant, "Example HelloWorld",
    type_name, &DDS_TOPIC_QOS_DEFAULT, NULL /* listener */,
    DDS_STATUS_MASK_NONE);
if (topic == NULL) {
    printf("create_topic error\n");
    subscriber_shutdown(participant);
    return -1;
}

/* Setup data reader listener */
```

```
    reader_listener.on_requested_deadline_missed  =
        HelloWorldListener_on_requested_deadline_missed;
    reader_listener.on_requested_incompatible_qos =
        HelloWorldListener_on_requested_incompatible_qos;
    reader_listener.on_sample_rejected =
        HelloWorldListener_on_sample_rejected;
    reader_listener.on_liveliness_changed =
        HelloWorldListener_on_liveliness_changed;
    reader_listener.on_sample_lost =
        HelloWorldListener_on_sample_lost;
    reader_listener.on_subscription_matched =
        HelloWorldListener_on_subscription_matched;
    reader_listener.on_data_available =
        HelloWorldListener_on_data_available;

    /* To customize data reader QoS, use
       DDS_Subscriber_get_default_datareader_qos() */
    reader = DDS_Subscriber_create_datareader(
        subscriber, DDS_Topic_as_topicdescription(topic),
        &DDS_DATAREADER_QOS_DEFAULT, &reader_listener, DDS_STATUS_MASK_ALL);
    if (reader == NULL) {
        printf("create_datareader error\n");
        subscriber_shutdown(participant);
        return -1;
    }

    /* Main loop */
    for (count=0; (sample_count == 0) || (count < sample_count); ++count) {
        printf("HelloWorld subscriber sleeping for %d sec...\n",
               poll_period.sec);
        NDDS_Utility_sleep(&poll_period);
    }

    /* Cleanup and delete delete all entities */
    DDS_DomainParticipantQos_finalize(&participant_qos);
    return subscriber_shutdown(participant);
}

#if defined(RTI_WINCE)
int wmain(int argc, wchar_t** argv)
{
    int domainId = 0;
    int sample_count = 0; /* infinite loop */

    if (argc >= 2) {
        domainId = _wtoi(argv[1]);
    }
    if (argc >= 3) {
        sample_count = _wtoi(argv[2]);
    }

    /* Uncomment this to turn on additional logging
    NDDS_Config_Logger_set_verbosity_by_category(
        NDDS_Config_Logger_get_instance(),
        NDDS_CONFIG_LOG_CATEGORY_API,
        NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
    */
```

```
    return subscriber_main(domainId, sample_count);
}
#elif !(defined(RTI_VXWORKS) && !defined(__RTP__)) && !defined(RTI_PSOS)
int main(int argc, char *argv[])
{
    int domainId = 0;
    int sample_count = 0; /* infinite loop */

    if (argc >= 2) {
        domainId = atoi(argv[1]);
    }
    if (argc >= 3) {
        sample_count = atoi(argv[2]);
    }

    /* Uncomment this to turn on additional logging
    NDDS_Config_Logger_set_verbosity_by_category(
        NDDS_Config_Logger_get_instance(),
        NDDS_CONFIG_LOG_CATEGORY_API,
        NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
    */

    return subscriber_main(domainId, sample_count);
}
#endif
```

4.15 HelloWorldPlugin.c

[$(NDDSHOME)/example/C/helloWorldWAN/HelloWorldPlugin.h]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from HelloWorld.idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/

#ifndef HelloWorldPlugin_1436885487_h
#define HelloWorldPlugin_1436885487_h

#include "HelloWorld.h"



struct RTICdrStream;

#ifndef pres_typePlugin_h
#include "pres/pres_typePlugin.h"
#endif


#if (defined(RTI_WIN32) || defined (RTI_WINCE)) && defined(NDDS_USER_DLL_EXPORT)
/* If the code is building on Windows, start exporting symbols.
*/
#undef NDDSUSERDllExport
#define NDDSUSERDllExport __declspec(dllexport)
#endif


#ifdef __cplusplus
extern "C" {
#endif

#define HelloWorld_LAST_MEMBER_ID 0

#define HelloWorldPlugin_get_sample PRESTypePluginDefaultEndpointData_getSample
#define HelloWorldPlugin_return_sample PRESTypePluginDefaultEndpointData_returnSample
#define HelloWorldPlugin_get_buffer PRESTypePluginDefaultEndpointData_getBuffer
#define HelloWorldPlugin_return_buffer PRESTypePluginDefaultEndpointData_returnBuffer


#define HelloWorldPlugin_create_sample PRESTypePluginDefaultEndpointData_createSample
#define HelloWorldPlugin_destroy_sample PRESTypePluginDefaultEndpointData_deleteSample

/* --------------------------------------------------------------------------------
    Support functions:
 * -------------------------------------------------------------------------------- */

NDDSUSERDllExport extern HelloWorld*
```

```
HelloWorldPluginSupport_create_data_ex(RTIBool allocate_pointers);

NDDSUSERDllExport extern HelloWorld*
HelloWorldPluginSupport_create_data(void);

NDDSUSERDllExport extern RTIBool
HelloWorldPluginSupport_copy_data(
    HelloWorld *out,
    const HelloWorld *in);

NDDSUSERDllExport extern void
HelloWorldPluginSupport_destroy_data_ex(
    HelloWorld *sample,RTIBool deallocate_pointers);

NDDSUSERDllExport extern void
HelloWorldPluginSupport_destroy_data(
    HelloWorld *sample);

NDDSUSERDllExport extern void
HelloWorldPluginSupport_print_data(
    const HelloWorld *sample,
    const char *desc,
    unsigned int indent);


/* ----------------------------------------------------------------------------
    Callback functions:
 * ---------------------------------------------------------------------------- */

NDDSUSERDllExport extern PRESTypePluginParticipantData
HelloWorldPlugin_on_participant_attached(
    void *registration_data,
    const struct PRESTypePluginParticipantInfo *participant_info,
    RTIBool top_level_registration,
    void *container_plugin_context,
    RTICdrTypeCode *typeCode);

NDDSUSERDllExport extern void
HelloWorldPlugin_on_participant_detached(
    PRESTypePluginParticipantData participant_data);

NDDSUSERDllExport extern PRESTypePluginEndpointData
HelloWorldPlugin_on_endpoint_attached(
    PRESTypePluginParticipantData participant_data,
    const struct PRESTypePluginEndpointInfo *endpoint_info,
    RTIBool top_level_registration,
    void *container_plugin_context);

NDDSUSERDllExport extern void
HelloWorldPlugin_on_endpoint_detached(
    PRESTypePluginEndpointData endpoint_data);


NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_copy_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *out,
```

```
    const HelloWorld *in);

/* ----------------------------------------------------------------------------------------
    (De)Serialize functions:
 * ---------------------------------------------------------------------------------------- */

NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_serialize(
    PRESTypePluginEndpointData endpoint_data,
    const HelloWorld *sample,
    struct RTICdrStream *stream,
    RTIBool serialize_encapsulation,
    RTIEncapsulationId encapsulation_id,
    RTIBool serialize_sample,
    void *endpoint_plugin_qos);

NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_deserialize_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_sample,
    void *endpoint_plugin_qos);


NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_deserialize(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld **sample,
    RTIBool * drop_sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_sample,
    void *endpoint_plugin_qos);




NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_skip(
    PRESTypePluginEndpointData endpoint_data,
    struct RTICdrStream *stream,
    RTIBool skip_encapsulation,
    RTIBool skip_sample,
    void *endpoint_plugin_qos);

NDDSUSERDllExport extern unsigned int
HelloWorldPlugin_get_serialized_sample_max_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment);

NDDSUSERDllExport extern unsigned int
HelloWorldPlugin_get_serialized_sample_min_size(
    PRESTypePluginEndpointData endpoint_data,
```

```
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment);

NDDSUSERDllExport extern unsigned int
HelloWorldPlugin_get_serialized_sample_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment,
    const HelloWorld * sample);



/* ----------------------------------------------------------------------------
    Key Management functions:
 * ---------------------------------------------------------------------------- */

NDDSUSERDllExport extern PRESTypePluginKeyKind
HelloWorldPlugin_get_key_kind(void);

NDDSUSERDllExport extern unsigned int
HelloWorldPlugin_get_serialized_key_max_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment);

NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_serialize_key(
    PRESTypePluginEndpointData endpoint_data,
    const HelloWorld *sample,
    struct RTICdrStream *stream,
    RTIBool serialize_encapsulation,
    RTIEncapsulationId encapsulation_id,
    RTIBool serialize_key,
    void *endpoint_plugin_qos);

NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_deserialize_key_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld * sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_key,
    void *endpoint_plugin_qos);


NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_deserialize_key(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld ** sample,
    RTIBool * drop_sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_key,
    void *endpoint_plugin_qos);
```

```
NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_serialized_sample_to_key(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_key,
    void *endpoint_plugin_qos);


/* Plugin Functions */
NDDSUSERDllExport extern struct PRESTypePlugin*
HelloWorldPlugin_new(void);

NDDSUSERDllExport extern void
HelloWorldPlugin_delete(struct PRESTypePlugin *);

#ifdef __cplusplus
}
#endif


#if (defined(RTI_WIN32) || defined (RTI_WINCE)) && defined(NDDS_USER_DLL_EXPORT)
/* If the code is building on Windows, stop exporting symbols.
*/
#undef NDDSUSERDllExport
#define NDDSUSERDllExport
#endif

#endif /* HelloWorldPlugin_1436885487_h */
```

[$(NDDSHOME)/example/C/helloWorldWAN/HelloWorldPlugin.c]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from HelloWorld.idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/


#include <string.h>

#ifdef __cplusplus
#ifndef ndds_cpp_h
  #include "ndds/ndds_cpp.h"
#endif
#else
#ifndef ndds_c_h
  #include "ndds/ndds_c.h"
#endif
#endif
```

```
#ifndef osapi_type_h
  #include "osapi/osapi_type.h"
#endif
#ifndef osapi_heap_h
  #include "osapi/osapi_heap.h"
#endif

#ifndef osapi_utility_h
  #include "osapi/osapi_utility.h"
#endif

#ifndef cdr_type_h
  #include "cdr/cdr_type.h"
#endif

#ifndef cdr_type_object_h
  #include "cdr/cdr_typeObject.h"
#endif

#ifndef cdr_encapsulation_h
  #include "cdr/cdr_encapsulation.h"
#endif

#ifndef cdr_stream_h
  #include "cdr/cdr_stream.h"
#endif

#ifndef pres_typePlugin_h
  #include "pres/pres_typePlugin.h"
#endif



#include "HelloWorldPlugin.h"


/* ----------------------------------------------------------------------------------------
 *  Type HelloWorld
 * ---------------------------------------------------------------------------------------- */

/* ----------------------------------------------------------------------------------------
    Support functions:
 * ---------------------------------------------------------------------------------------- */

HelloWorld *
HelloWorldPluginSupport_create_data_ex(RTIBool allocate_pointers){
    HelloWorld *sample = NULL;

    RTIOsapiHeap_allocateStructure(
        &sample, HelloWorld);

    if(sample != NULL) {
        if (!HelloWorld_initialize_ex(sample,allocate_pointers, RTI_TRUE)) {
            RTIOsapiHeap_freeStructure(sample);
            return NULL;
        }
```

```
    }
    return sample;
}


HelloWorld *
HelloWorldPluginSupport_create_data(void)
{
    return HelloWorldPluginSupport_create_data_ex(RTI_TRUE);
}


void
HelloWorldPluginSupport_destroy_data_ex(
    HelloWorld *sample,RTIBool deallocate_pointers) {

    HelloWorld_finalize_ex(sample,deallocate_pointers);

    RTIOsapiHeap_freeStructure(sample);
}


void
HelloWorldPluginSupport_destroy_data(
    HelloWorld *sample) {

    HelloWorldPluginSupport_destroy_data_ex(sample,RTI_TRUE);

}


RTIBool
HelloWorldPluginSupport_copy_data(
    HelloWorld *dst,
    const HelloWorld *src)
{
    return HelloWorld_copy(dst,src);
}


void
HelloWorldPluginSupport_print_data(
    const HelloWorld *sample,
    const char *desc,
    unsigned int indent_level)
{


    RTICdrType_printIndent(indent_level);

    if (desc != NULL) {
      RTILog_debug("%s:\n", desc);
    } else {
      RTILog_debug("\n");
    }

    if (sample == NULL) {
```

```
      RTILog_debug("NULL\n");
      return;
    }


    if (&sample->msg==NULL) {
        RTICdrType_printString(
            NULL, "msg", indent_level + 1);
    } else {
        RTICdrType_printString(
            sample->msg, "msg", indent_level + 1);
    }



}


/* ------------------------------------------------------------------------
    Callback functions:
 * ---------------------------------------------------------------------- */



PRESTypePluginParticipantData
HelloWorldPlugin_on_participant_attached(
    void *registration_data,
    const struct PRESTypePluginParticipantInfo *participant_info,
    RTIBool top_level_registration,
    void *container_plugin_context,
    RTICdrTypeCode *type_code)
{

    if (registration_data) {} /* To avoid warnings */
    if (participant_info) {} /* To avoid warnings */
    if (top_level_registration) {} /* To avoid warnings */
    if (container_plugin_context) {} /* To avoid warnings */
    if (type_code) {} /* To avoid warnings */
    return PRESTypePluginDefaultParticipantData_new(participant_info);

}


void
HelloWorldPlugin_on_participant_detached(
    PRESTypePluginParticipantData participant_data)
{

  PRESTypePluginDefaultParticipantData_delete(participant_data);
}


PRESTypePluginEndpointData
HelloWorldPlugin_on_endpoint_attached(
    PRESTypePluginParticipantData participant_data,
    const struct PRESTypePluginEndpointInfo *endpoint_info,
    RTIBool top_level_registration,
```

```
    void *containerPluginContext)
{
    PRESTypePluginEndpointData epd = NULL;

    unsigned int serializedSampleMaxSize;

   if (top_level_registration) {} /* To avoid warnings */
   if (containerPluginContext) {} /* To avoid warnings */

    epd = PRESTypePluginDefaultEndpointData_new(
            participant_data,
            endpoint_info,
            (PRESTypePluginDefaultEndpointDataCreateSampleFunction)
            HelloWorldPluginSupport_create_data,
            (PRESTypePluginDefaultEndpointDataDestroySampleFunction)
            HelloWorldPluginSupport_destroy_data,
            NULL, NULL);

    if (epd == NULL) {
        return NULL;
    }



    if (endpoint_info->endpointKind == PRES_TYPEPLUGIN_ENDPOINT_WRITER) {
        serializedSampleMaxSize = HelloWorldPlugin_get_serialized_sample_max_size(
            epd,RTI_FALSE,RTI_CDR_ENCAPSULATION_ID_CDR_BE,0);

        PRESTypePluginDefaultEndpointData_setMaxSizeSerializedSample(epd, serializedSampleMaxSize);

        if (PRESTypePluginDefaultEndpointData_createWriterPool(
                epd,
                endpoint_info,
            (PRESTypePluginGetSerializedSampleMaxSizeFunction)
                HelloWorldPlugin_get_serialized_sample_max_size, epd,
            (PRESTypePluginGetSerializedSampleSizeFunction)
            HelloWorldPlugin_get_serialized_sample_size,
            epd) == RTI_FALSE) {
            PRESTypePluginDefaultEndpointData_delete(epd);
            return NULL;
        }
    }



    return epd;
}


void
HelloWorldPlugin_on_endpoint_detached(
    PRESTypePluginEndpointData endpoint_data)
{

    PRESTypePluginDefaultEndpointData_delete(endpoint_data);
}
```

```
RTIBool
HelloWorldPlugin_copy_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *dst,
    const HelloWorld *src)
{
    if (endpoint_data) {} /* To avoid warnings */
    return HelloWorldPluginSupport_copy_data(dst,src);
}

/* ------------------------------------------------------------------------------------
    (De)Serialize functions:
 * ------------------------------------------------------------------------------------ */

unsigned int
HelloWorldPlugin_get_serialized_sample_max_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment);


RTIBool
HelloWorldPlugin_serialize(
    PRESTypePluginEndpointData endpoint_data,
    const HelloWorld *sample,
    struct RTICdrStream *stream,
    RTIBool serialize_encapsulation,
    RTIEncapsulationId encapsulation_id,
    RTIBool serialize_sample,
    void *endpoint_plugin_qos)
{
    char * position = NULL;
    RTIBool retval = RTI_TRUE;

    if (endpoint_data) {} /* To avoid warnings */
    if (endpoint_plugin_qos) {} /* To avoid warnings */


    if(serialize_encapsulation) {

        if (!RTICdrStream_serializeAndSetCdrEncapsulation(stream, encapsulation_id)) {
            return RTI_FALSE;
        }

        position = RTICdrStream_resetAlignment(stream);

    }


    if(serialize_sample) {

    if (!RTICdrStream_serializeString(
        stream, sample->msg, (128) + 1)) {
        return RTI_FALSE;
```

```
    }


    }


    if(serialize_encapsulation) {
        RTICdrStream_restoreAlignment(stream,position);
    }


  return retval;
}


RTIBool
HelloWorldPlugin_deserialize_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_sample,
    void *endpoint_plugin_qos)
{
    char * position = NULL;

    RTIBool done = RTI_FALSE;

    if (endpoint_data) {} /* To avoid warnings */
    if (endpoint_plugin_qos) {} /* To avoid warnings */


    if(deserialize_encapsulation) {
        /* Deserialize encapsulation */
        if (!RTICdrStream_deserializeAndSetCdrEncapsulation(stream)) {
            return RTI_FALSE;
        }

        position = RTICdrStream_resetAlignment(stream);

    }


    if(deserialize_sample) {
        HelloWorld_initialize_ex(sample, RTI_FALSE, RTI_FALSE);

    if (!RTICdrStream_deserializeString(
        stream, sample->msg, (128) + 1)) {
        goto fin;
    }


    }

    done = RTI_TRUE;
fin:
    if (done != RTI_TRUE && RTICdrStream_getRemainder(stream) >  0) {
```

```
        return RTI_FALSE;
    }

    if(deserialize_encapsulation) {
        RTICdrStream_restoreAlignment(stream,position);
    }


    return RTI_TRUE;
}



RTIBool
HelloWorldPlugin_deserialize(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld **sample,
    RTIBool * drop_sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_sample,
    void *endpoint_plugin_qos)
{

    if (drop_sample) {} /* To avoid warnings */

    return HelloWorldPlugin_deserialize_sample(
        endpoint_data, (sample != NULL)?*sample:NULL,
        stream, deserialize_encapsulation, deserialize_sample,
        endpoint_plugin_qos);

}



RTIBool HelloWorldPlugin_skip(
    PRESTypePluginEndpointData endpoint_data,
    struct RTICdrStream *stream,
    RTIBool skip_encapsulation,
    RTIBool skip_sample,
    void *endpoint_plugin_qos)
{
    char * position = NULL;

    RTIBool done = RTI_FALSE;

    if (endpoint_data) {} /* To avoid warnings */
    if (endpoint_plugin_qos) {} /* To avoid warnings */


    if(skip_encapsulation) {
        if (!RTICdrStream_skipEncapsulation(stream)) {
            return RTI_FALSE;
        }
```

```
        position = RTICdrStream_resetAlignment(stream);

    }

    if (skip_sample) {

    if (!RTICdrStream_skipString(stream, (128) + 1)) {
        goto fin;
    }



    }


    done = RTI_TRUE;
fin:
    if (done != RTI_TRUE && RTICdrStream_getRemainder(stream) >  0) {
        return RTI_FALSE;
    }

    if(skip_encapsulation) {
        RTICdrStream_restoreAlignment(stream,position);
    }


    return RTI_TRUE;
}


unsigned int
HelloWorldPlugin_get_serialized_sample_max_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment)
{

    unsigned int initial_alignment = current_alignment;

    unsigned int encapsulation_size = current_alignment;

    if (endpoint_data) {} /* To avoid warnings */


    if (include_encapsulation) {

        if (!RTICdrEncapsulation_validEncapsulationId(encapsulation_id)) {
            return 1;
        }

        RTICdrStream_getEncapsulationSize(encapsulation_size);
        encapsulation_size -= current_alignment;
        current_alignment = 0;
        initial_alignment = 0;
```

```
    }


    current_alignment +=  RTICdrType_getStringMaxSizeSerialized(
        current_alignment, (128) + 1);


    if (include_encapsulation) {
        current_alignment += encapsulation_size;
    }

    return current_alignment - initial_alignment;
}


unsigned int
HelloWorldPlugin_get_serialized_sample_min_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment)
{

    unsigned int initial_alignment = current_alignment;

    unsigned int encapsulation_size = current_alignment;

    if (endpoint_data) {} /* To avoid warnings */


    if (include_encapsulation) {

        if (!RTICdrEncapsulation_validEncapsulationId(encapsulation_id)) {
            return 1;
        }

        RTICdrStream_getEncapsulationSize(encapsulation_size);
        encapsulation_size -= current_alignment;
        current_alignment = 0;
        initial_alignment = 0;

    }


    current_alignment +=  RTICdrType_getStringMaxSizeSerialized(
        current_alignment, 1);


    if (include_encapsulation) {
        current_alignment += encapsulation_size;
    }

    return current_alignment - initial_alignment;
}


/* Returns the size of the sample in its serialized form (in bytes).
```

```
 * It can also be an estimation in excess of the real buffer needed
 * during a call to the serialize() function.
 * The value reported does not have to include the space for the
 * encapsulation flags.
 */
unsigned int
HelloWorldPlugin_get_serialized_sample_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment,
    const HelloWorld * sample)
{

    unsigned int initial_alignment = current_alignment;

    unsigned int encapsulation_size = current_alignment;

    if (endpoint_data) {} /* To avoid warnings */
    if (sample) {} /* To avoid warnings */


    if (include_encapsulation) {

        if (!RTICdrEncapsulation_validEncapsulationId(encapsulation_id)) {
            return 1;
        }

        RTICdrStream_getEncapsulationSize(encapsulation_size);
        encapsulation_size -= current_alignment;
        current_alignment = 0;
        initial_alignment = 0;

    }


    current_alignment += RTICdrType_getStringSerializedSize(
        current_alignment, sample->msg);


    if (include_encapsulation) {
        current_alignment += encapsulation_size;
    }

    return current_alignment - initial_alignment;
}






/* ----------------------------------------------------------------------------
    Key Management functions:
 * ---------------------------------------------------------------------------- */
```

```
PRESTypePluginKeyKind
HelloWorldPlugin_get_key_kind(void)
{

    return PRES_TYPEPLUGIN_NO_KEY;

}


RTIBool
HelloWorldPlugin_serialize_key(
    PRESTypePluginEndpointData endpoint_data,
    const HelloWorld *sample,
    struct RTICdrStream *stream,
    RTIBool serialize_encapsulation,
    RTIEncapsulationId encapsulation_id,
    RTIBool serialize_key,
    void *endpoint_plugin_qos)
{
    char * position = NULL;

    if (endpoint_data) {} /* To avoid warnings */
    if (endpoint_plugin_qos) {} /* To avoid warnings */


    if(serialize_encapsulation) {

        if (!RTICdrStream_serializeAndSetCdrEncapsulation(stream, encapsulation_id)) {
            return RTI_FALSE;
        }


        position = RTICdrStream_resetAlignment(stream);

    }

    if(serialize_key) {

        if (!HelloWorldPlugin_serialize(
                endpoint_data,
                sample,
                stream,
                RTI_FALSE, encapsulation_id,
                RTI_TRUE,
                endpoint_plugin_qos)) {
            return RTI_FALSE;
        }

    }


    if(serialize_encapsulation) {
        RTICdrStream_restoreAlignment(stream,position);
    }
```

```
    return RTI_TRUE;
}


RTIBool HelloWorldPlugin_deserialize_key_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_key,
    void *endpoint_plugin_qos)
{
    char * position = NULL;

    if (endpoint_data) {} /* To avoid warnings */
    if (endpoint_plugin_qos) {} /* To avoid warnings */


    if(deserialize_encapsulation) {
        /* Deserialize encapsulation */
        if (!RTICdrStream_deserializeAndSetCdrEncapsulation(stream)) {
            return RTI_FALSE;
        }


        position = RTICdrStream_resetAlignment(stream);

    }

    if (deserialize_key) {

        if (!HelloWorldPlugin_deserialize_sample(
                endpoint_data, sample, stream,
                RTI_FALSE, RTI_TRUE,
                endpoint_plugin_qos)) {
            return RTI_FALSE;
        }

    }


    if(deserialize_encapsulation) {
        RTICdrStream_restoreAlignment(stream,position);
    }


    return RTI_TRUE;
}



RTIBool HelloWorldPlugin_deserialize_key(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld **sample,
    RTIBool * drop_sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
```

```
    RTIBool deserialize_key,
    void *endpoint_plugin_qos)
{
    if (drop_sample) {} /* To avoid warnings */
    return HelloWorldPlugin_deserialize_key_sample(
        endpoint_data, (sample != NULL)?*sample:NULL, stream,
        deserialize_encapsulation, deserialize_key, endpoint_plugin_qos);
}



unsigned int
HelloWorldPlugin_get_serialized_key_max_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment)
{

    unsigned int encapsulation_size = current_alignment;


    unsigned int initial_alignment = current_alignment;


    if (endpoint_data) {} /* To avoid warnings */


    if (include_encapsulation) {
        if (!RTICdrEncapsulation_validEncapsulationId(encapsulation_id)) {
            return 1;
        }


        RTICdrStream_getEncapsulationSize(encapsulation_size);
        encapsulation_size -= current_alignment;
        current_alignment = 0;
        initial_alignment = 0;

    }


    current_alignment += HelloWorldPlugin_get_serialized_sample_max_size(
        endpoint_data,RTI_FALSE, encapsulation_id, current_alignment);

    if (include_encapsulation) {
        current_alignment += encapsulation_size;
    }

    return current_alignment - initial_alignment;
}


RTIBool
HelloWorldPlugin_serialized_sample_to_key(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *sample,
```

```
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_key,
    void *endpoint_plugin_qos)
{
    char * position = NULL;

    RTIBool done = RTI_FALSE;

    if (stream == NULL) goto fin; /* To avoid warnings */


    if(deserialize_encapsulation) {
        if (!RTICdrStream_deserializeAndSetCdrEncapsulation(stream)) {
            return RTI_FALSE;
        }

        position = RTICdrStream_resetAlignment(stream);

    }

    if (deserialize_key) {

        if (!HelloWorldPlugin_deserialize_sample(
            endpoint_data, sample, stream, RTI_FALSE,
            RTI_TRUE, endpoint_plugin_qos)) {
            return RTI_FALSE;
        }

    }


    done = RTI_TRUE;
fin:
    if (done != RTI_TRUE && RTICdrStream_getRemainder(stream) >  0) {
        return RTI_FALSE;
    }

    if(deserialize_encapsulation) {
        RTICdrStream_restoreAlignment(stream,position);
    }


    return RTI_TRUE;
}




/* ----------------------------------------------------------------------
 * Plug-in Installation Methods
 * ---------------------------------------------------------------------- */

struct PRESTypePlugin *HelloWorldPlugin_new(void)
{
    struct PRESTypePlugin *plugin = NULL;
    const struct PRESTypePluginVersion PLUGIN_VERSION =
```

```
    PRES_TYPE_PLUGIN_VERSION_2_0;

RTIOsapiHeap_allocateStructure(
    &plugin, struct PRESTypePlugin);
if (plugin == NULL) {
   return NULL;
}

plugin->version = PLUGIN_VERSION;

/* set up parent's function pointers */
plugin->onParticipantAttached =
    (PRESTypePluginOnParticipantAttachedCallback)
    HelloWorldPlugin_on_participant_attached;
plugin->onParticipantDetached =
    (PRESTypePluginOnParticipantDetachedCallback)
    HelloWorldPlugin_on_participant_detached;
plugin->onEndpointAttached =
    (PRESTypePluginOnEndpointAttachedCallback)
    HelloWorldPlugin_on_endpoint_attached;
plugin->onEndpointDetached =
    (PRESTypePluginOnEndpointDetachedCallback)
    HelloWorldPlugin_on_endpoint_detached;

plugin->copySampleFnc =
    (PRESTypePluginCopySampleFunction)
    HelloWorldPlugin_copy_sample;
plugin->createSampleFnc =
    (PRESTypePluginCreateSampleFunction)
    HelloWorldPlugin_create_sample;
plugin->destroySampleFnc =
    (PRESTypePluginDestroySampleFunction)
    HelloWorldPlugin_destroy_sample;

plugin->serializeFnc =
    (PRESTypePluginSerializeFunction)
    HelloWorldPlugin_serialize;
plugin->deserializeFnc =
    (PRESTypePluginDeserializeFunction)
    HelloWorldPlugin_deserialize;
plugin->getSerializedSampleMaxSizeFnc =
    (PRESTypePluginGetSerializedSampleMaxSizeFunction)
    HelloWorldPlugin_get_serialized_sample_max_size;
plugin->getSerializedSampleMinSizeFnc =
    (PRESTypePluginGetSerializedSampleMinSizeFunction)
    HelloWorldPlugin_get_serialized_sample_min_size;


plugin->getSampleFnc =
    (PRESTypePluginGetSampleFunction)
    HelloWorldPlugin_get_sample;
plugin->returnSampleFnc =
    (PRESTypePluginReturnSampleFunction)
    HelloWorldPlugin_return_sample;

plugin->getKeyKindFnc =
    (PRESTypePluginGetKeyKindFunction)
```

```
        HelloWorldPlugin_get_key_kind;


    /* These functions are only used for keyed types. As this is not a keyed
    type they are all set to NULL
    */
    plugin->serializeKeyFnc = NULL;
    plugin->deserializeKeyFnc = NULL;
    plugin->getKeyFnc = NULL;
    plugin->returnKeyFnc = NULL;
    plugin->instanceToKeyFnc = NULL;
    plugin->keyToInstanceFnc = NULL;
    plugin->getSerializedKeyMaxSizeFnc = NULL;
    plugin->instanceToKeyHashFnc = NULL;
    plugin->serializedSampleToKeyHashFnc = NULL;
    plugin->serializedKeyToKeyHashFnc = NULL;

    plugin->typeCode =  (struct RTICdrTypeCode *)HelloWorld_get_typecode();

    plugin->languageKind = PRES_TYPEPLUGIN_DDS_TYPE;

    /* Serialized buffer */
    plugin->getBuffer =
        (PRESTypePluginGetBufferFunction)
        HelloWorldPlugin_get_buffer;
    plugin->returnBuffer =
        (PRESTypePluginReturnBufferFunction)
        HelloWorldPlugin_return_buffer;
    plugin->getSerializedSampleSizeFnc =
        (PRESTypePluginGetSerializedSampleSizeFunction)
        HelloWorldPlugin_get_serialized_sample_size;

    plugin->endpointTypeName = HelloWorldTYPENAME;

    return plugin;
}

void
HelloWorldPlugin_delete(struct PRESTypePlugin *plugin)
{
    RTIOsapiHeap_freeStructure(plugin);
}
```

4.16 HelloWorldSupport.c

[$(NDDSHOME)/example/C/helloWorldWAN/HelloWorldSupport.h]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from HelloWorld.idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/

#ifndef HelloWorldSupport_1436885487_h
#define HelloWorldSupport_1436885487_h

/* Uses */
#include "HelloWorld.h"



#ifdef __cplusplus
#ifndef ndds_cpp_h
  #include "ndds/ndds_cpp.h"
#endif
#else
#ifndef ndds_c_h
  #include "ndds/ndds_c.h"
#endif
#endif



/* ========================================================================= */
#if (defined(RTI_WIN32) || defined (RTI_WINCE)) && defined(NDDS_USER_DLL_EXPORT)
  /* If the code is building on Windows, start exporting symbols.
   */
  #undef NDDSUSERDllExport
  #define NDDSUSERDllExport __declspec(dllexport)

#ifdef __cplusplus
  /* If we're building on Windows, explicitly import the superclasses of
   * the types declared below.
   */
  class __declspec(dllimport) DDSTypeSupport;
  class __declspec(dllimport) DDSDataWriter;
  class __declspec(dllimport) DDSDataReader;
#endif

#endif

#ifdef __cplusplus

DDS_TYPESUPPORT_CPP(HelloWorldTypeSupport, HelloWorld);

DDS_DATAWRITER_CPP(HelloWorldDataWriter, HelloWorld);
```

```
DDS_DATAREADER_CPP(HelloWorldDataReader, HelloWorldSeq, HelloWorld);


#else

DDS_TYPESUPPORT_C(HelloWorldTypeSupport, HelloWorld);
DDS_DATAWRITER_C(HelloWorldDataWriter, HelloWorld);
DDS_DATAREADER_C(HelloWorldDataReader, HelloWorldSeq, HelloWorld);

#endif

#if (defined(RTI_WIN32) || defined (RTI_WINCE)) && defined(NDDS_USER_DLL_EXPORT)
  /* If the code is building on Windows, stop exporting symbols.
   */
  #undef NDDSUSERDllExport
  #define NDDSUSERDllExport
#endif



#endif  /* HelloWorldSupport_1436885487_h */
```

[$(NDDSHOME)/example/C/helloWorldWAN/HelloWorldSupport.c]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from HelloWorld.idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/

#include "HelloWorldSupport.h"
#include "HelloWorldPlugin.h"



#ifdef __cplusplus
    #ifndef dds_c_log_impl_h
         #include "dds_c/dds_c_log_impl.h"
    #endif
#endif




/* ========================================================================= */
/* ------------------------------------------------------------------------- */
/* DDSDataWriter
*/

/* Requires */
#define TTYPENAME   HelloWorldTYPENAME

/* Defines */
```

```
#define TDataWriter HelloWorldDataWriter
#define TData       HelloWorld


#ifdef __cplusplus
#include "dds_cpp/generic/dds_cpp_data_TDataWriter.gen"
#else
#include "dds_c/generic/dds_c_data_TDataWriter.gen"
#endif


#undef TDataWriter
#undef TData

#undef TTYPENAME

/* ------------------------------------------------------------------ */
/* DDSDataReader
*/

/* Requires */
#define TTYPENAME   HelloWorldTYPENAME

/* Defines */
#define TDataReader HelloWorldDataReader
#define TDataSeq    HelloWorldSeq
#define TData       HelloWorld


#ifdef __cplusplus
#include "dds_cpp/generic/dds_cpp_data_TDataReader.gen"
#else
#include "dds_c/generic/dds_c_data_TDataReader.gen"
#endif


#undef TDataReader
#undef TDataSeq
#undef TData

#undef TTYPENAME

/* ------------------------------------------------------------------ */
/* TypeSupport

  <<IMPLEMENTATION >>

   Requires:  TTYPENAME,
              TPlugin_new
              TPlugin_delete
   Defines:   TTypeSupport, TData, TDataReader, TDataWriter
*/

/* Requires */
#define TTYPENAME    HelloWorldTYPENAME
#define TPlugin_new  HelloWorldPlugin_new
#define TPlugin_delete  HelloWorldPlugin_delete
```

```
/* Defines */
#define TTypeSupport HelloWorldTypeSupport
#define TData        HelloWorld
#define TDataReader  HelloWorldDataReader
#define TDataWriter  HelloWorldDataWriter
#ifdef __cplusplus



#include "dds_cpp/generic/dds_cpp_data_TTypeSupport.gen"



#else
#include "dds_c/generic/dds_c_data_TTypeSupport.gen"
#endif
#undef TTypeSupport
#undef TData
#undef TDataReader
#undef TDataWriter

#undef TTYPENAME
#undef TPlugin_new
#undef TPlugin_delete
```

4.17 C++ Example

Secure WAN Transport Example Using C++.

Modules

- **HelloWorld.idl**
- **HelloWorld.cxx**
- **HelloWorld_publisher.cxx**
- **HelloWorld_subscriber.cxx**
- **HelloWorldPlugin.cxx**
- **HelloWorldSupport.cxx**

4.17.1 Detailed Description

Secure WAN Transport Example Using C++.

4.18 HelloWorld.idl

[$(NDDSHOME)/example/CPP/helloWorldWAN/HelloWorld.idl]

```
struct HelloWorld {
    string<128> msg;
};
```

4.19 HelloWorld.cxx

[$(NDDSHOME)/example/CPP/helloWorldWAN/HelloWorld.h]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from HelloWorld.idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/

#ifndef HelloWorld_1436885487_h
#define HelloWorld_1436885487_h

#ifndef NDDS_STANDALONE_TYPE
    #ifdef __cplusplus
        #ifndef ndds_cpp_h
            #include "ndds/ndds_cpp.h"
        #endif
    #else
        #ifndef ndds_c_h
            #include "ndds/ndds_c.h"
        #endif
    #endif
#else
    #include "ndds_standalone_type.h"
#endif


#ifdef __cplusplus
extern "C" {
#endif


extern const char *HelloWorldTYPENAME;


#ifdef __cplusplus
}
#endif


#ifdef __cplusplus
    struct HelloWorldSeq;

#ifndef NDDS_STANDALONE_TYPE
    class HelloWorldTypeSupport;
    class HelloWorldDataWriter;
    class HelloWorldDataReader;
#endif

#endif
```

```
class HelloWorld
{
public:
#ifdef __cplusplus
    typedef struct HelloWorldSeq Seq;

#ifndef NDDS_STANDALONE_TYPE
    typedef HelloWorldTypeSupport TypeSupport;
    typedef HelloWorldDataWriter DataWriter;
    typedef HelloWorldDataReader DataReader;
#endif

#endif

    char*  msg; /* maximum length = (128) */


};


#if (defined(RTI_WIN32) || defined (RTI_WINCE)) && defined(NDDS_USER_DLL_EXPORT)
  /* If the code is building on Windows, start exporting symbols.
   */
  #undef NDDSUSERDllExport
  #define NDDSUSERDllExport __declspec(dllexport)
#endif


NDDSUSERDllExport DDS_TypeCode* HelloWorld_get_typecode(void); /* Type code */


DDS_SEQUENCE(HelloWorldSeq, HelloWorld);

NDDSUSERDllExport
RTIBool HelloWorld_initialize(
        HelloWorld* self);

NDDSUSERDllExport
RTIBool HelloWorld_initialize_ex(
        HelloWorld* self,RTIBool allocatePointers,RTIBool allocateMemory);

NDDSUSERDllExport
void HelloWorld_finalize(
        HelloWorld* self);

NDDSUSERDllExport
void HelloWorld_finalize_ex(
        HelloWorld* self,RTIBool deletePointers);

NDDSUSERDllExport
RTIBool HelloWorld_copy(
        HelloWorld* dst,
        const HelloWorld* src);

#if (defined(RTI_WIN32) || defined (RTI_WINCE)) && defined(NDDS_USER_DLL_EXPORT)
  /* If the code is building on Windows, stop exporting symbols.
```

```
   */
  #undef NDDSUSERDllExport
  #define NDDSUSERDllExport
#endif



#endif /* HelloWorld_1436885487_h */
```

[$(NDDSHOME)/example/CPP/helloWorldWAN/HelloWorld.cxx]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from HelloWorld.idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/


#ifndef NDDS_STANDALONE_TYPE
    #ifdef __cplusplus
        #ifndef ndds_cpp_h
            #include "ndds/ndds_cpp.h"
        #endif
        #ifndef dds_c_log_impl_h
            #include "dds_c/dds_c_log_impl.h"
        #endif
    #else
        #ifndef ndds_c_h
            #include "ndds/ndds_c.h"
        #endif
    #endif

    #ifndef cdr_type_h
        #include "cdr/cdr_type.h"
    #endif

    #ifndef osapi_heap_h
        #include "osapi/osapi_heap.h"
    #endif
#else
    #include "ndds_standalone_type.h"
#endif



#include "HelloWorld.h"

/* ========================================================================= */
const char *HelloWorldTYPENAME = "HelloWorld";

DDS_TypeCode* HelloWorld_get_typecode()
{
    static RTIBool is_initialized = RTI_FALSE;
```

```
    static DDS_TypeCode HelloWorld_g_tc_msg_string = DDS_INITIALIZE_STRING_TYPECODE(128);

    static DDS_TypeCode_Member HelloWorld_g_tc_members[1]=
    {
        {
            (char *)"msg",/* Member name */
            {
                0,/* Representation ID */
                DDS_BOOLEAN_FALSE,/* Is a pointer? */
                -1, /* Bitfield bits */
                NULL/* Member type code is assigned later */
            },
            0, /* Ignored */
            0, /* Ignored */
            0, /* Ignored */
            NULL, /* Ignored */
            DDS_BOOLEAN_FALSE, /* Is a key? */
            DDS_PRIVATE_MEMBER,/* Ignored */
            0,/* Ignored */
            NULL/* Ignored */
        }
    };

    static DDS_TypeCode HelloWorld_g_tc =
    {{
        DDS_TK_STRUCT,/* Kind */
        DDS_BOOLEAN_FALSE, /* Ignored */
        -1,/* Ignored */
        (char *)"HelloWorld", /* Name */
        NULL, /* Ignored */
        0, /* Ignored */
        0, /* Ignored */
        NULL, /* Ignored */
        1, /* Number of members */
        HelloWorld_g_tc_members, /* Members */
        DDS_VM_NONE /* Ignored */
    }}; /* Type code for HelloWorld*/

    if (is_initialized) {
        return &HelloWorld_g_tc;
    }


    HelloWorld_g_tc_members[0]._representation._typeCode = (RTICdrTypeCode *)&HelloWorld_g_tc_msg_string;

    is_initialized = RTI_TRUE;

    return &HelloWorld_g_tc;
}


RTIBool HelloWorld_initialize(
    HelloWorld* sample) {
  return HelloWorld_initialize_ex(sample,RTI_TRUE,RTI_TRUE);
}
```

```
RTIBool HelloWorld_initialize_ex(
    HelloWorld* sample,RTIBool allocatePointers,RTIBool allocateMemory)
{


    if (allocatePointers) {} /* To avoid warnings */
    if (allocateMemory) {} /* To avoid warnings */

    if (allocateMemory) {
        sample->msg = DDS_String_alloc((128));
        if (sample->msg == NULL) {
            return RTI_FALSE;
        }
    } else {
        if (sample->msg != NULL) {
            sample->msg[0] = '\0';
        }
    }



    return RTI_TRUE;
}

void HelloWorld_finalize(
    HelloWorld* sample)
{
    HelloWorld_finalize_ex(sample,RTI_TRUE);
}

void HelloWorld_finalize_ex(
    HelloWorld* sample,RTIBool deletePointers)
{
    if (sample) { } /* To avoid warnings */
    if (deletePointers) {} /* To avoid warnings */


    DDS_String_free(sample->msg);


}

RTIBool HelloWorld_copy(
    HelloWorld* dst,
    const HelloWorld* src)
{

    if (!RTICdrType_copyString(
        dst->msg, src->msg, (128) + 1)) {
        return RTI_FALSE;
    }



    return RTI_TRUE;
}
```

```
#define T HelloWorld
#define TSeq HelloWorldSeq
#define T_initialize_ex HelloWorld_initialize_ex
#define T_finalize_ex   HelloWorld_finalize_ex
#define T_copy       HelloWorld_copy

#ifndef NDDS_STANDALONE_TYPE
#include "dds_c/generic/dds_c_sequence_TSeq.gen"
#ifdef __cplusplus
#include "dds_cpp/generic/dds_cpp_sequence_TSeq.gen"
#endif
#else
#include "dds_c_sequence_TSeq.gen"
#ifdef __cplusplus
#include "dds_cpp_sequence_TSeq.gen"
#endif
#endif

#undef T_copy
#undef T_finalize_ex
#undef T_initialize_ex
#undef TSeq
#undef T
```

4.20 HelloWorld_publisher.cxx

[$(NDDSHOME)/example/CPP/helloWorldWAN/HelloWorld_publisher.cxx]

```
/* HelloWorld_publisher.cxx

   A publication of data of type HelloWorld

   This file is derived from code automatically generated by the rtiddsgen
   command:

   rtiddsgen -language C++ -example <arch> HelloWorld.idl

   Example publication of type HelloWorld automatically generated by
   'rtiddsgen'. To test them follow these steps:

   (1) Compile this file and the example subscription.

   (2) Start the subscription on the same domain used for RTI Connext with the command
       objs/<arch>/HelloWorld_subscriber <domain_id> <sample_count>

   (3) Start the publication on the same domain used for RTI Connext with the command
       objs/<arch>/HelloWorld_publisher <domain_id> <sample_count>

   (4) [Optional] Specify the list of discovery initial peers and
       multicast receive addresses via an environment variable or a file
       (in the current working directory) called NDDS_DISCOVERY_PEERS.

   You can run any number of publishers and subscribers programs, and can
   add and remove them dynamically from the domain.


   Example:

       To run the example application on domain <domain_id>:

       On Unix:

       objs/<arch>/HelloWorld_publisher <domain_id>
       objs/<arch>/HelloWorld_subscriber <domain_id>

       On Windows:

       objs\<arch>\HelloWorld_publisher <domain_id>
       objs\<arch>\HelloWorld_subscriber <domain_id>


modification history
------------ -------
*/

#include <stdio.h>
#include <stdlib.h>
#include "ndds/ndds_cpp.h"
#include "HelloWorld.h"
#include "HelloWorldSupport.h"
```

```
/* define this to enable security using DTLS */
/*#define USE_SECURITY*/

/* set this value to the address of your WAN Rendezvous server */
#define WAN_SERVER "127.0.0.1"

/* your publisher and subscriber should have unique IDs */
#define WAN_ID "1"

/* Delete all entities */
static int publisher_shutdown(
    DDSDomainParticipant *participant)
{
    DDS_ReturnCode_t retcode;
    int status = 0;

    if (participant != NULL) {
        retcode = participant->delete_contained_entities();
        if (retcode != DDS_RETCODE_OK) {
            printf("delete_contained_entities error %d\n", retcode);
            status = -1;
        }

        retcode = DDSTheParticipantFactory->delete_participant(participant);
        if (retcode != DDS_RETCODE_OK) {
            printf("delete_participant error %d\n", retcode);
            status = -1;
        }
    }

    /* RTI Connext provides finalize_instance() method on
       domain participant factory for people who want to release memory used
       by the participant factory and singleton. Uncomment the following
       block of code for clean destruction of the singleton. */
/*
    retcode = DDSDomainParticipantFactory::finalize_instance();
    if (retcode != DDS_RETCODE_OK) {
        printf("finalize_instance error %d\n", retcode);
        status = -1;
    }
*/
    return status;
}

extern "C" int publisher_main(int domainId, int sample_count)
{
    DDSDomainParticipant *participant = NULL;
    DDSPublisher *publisher = NULL;
    DDSTopic *topic = NULL;
    DDSDataWriter *writer = NULL;
    HelloWorldDataWriter *HelloWorld_writer = NULL;
    HelloWorld *instance = NULL;
    DDS_ReturnCode_t retcode;
    DDS_InstanceHandle_t instance_handle = DDS_HANDLE_NIL;
    const char *type_name = NULL;
    int count = 0;
```

```
    struct DDS_Duration_t send_period = {4,0};

    struct DDS_DomainParticipantQos participant_qos;

    /* Get default participant QoS from participant factory */
    retcode = DDSDomainParticipantFactory::get_instance()->get_default_participant_qos(
        participant_qos);
    if (retcode != DDS_RETCODE_OK) {
        printf("could not get default participant qos\n");
        publisher_shutdown(participant);
        return -1;
    }

    /* Disable builtin transports */
    participant_qos.transport_builtin.mask = DDS_TRANSPORTBUILTIN_MASK_NONE;

    /* Set up property Qos to load plugin */
    retcode = DDSPropertyQosPolicyHelper::add_property(
        participant_qos.property,
        "dds.transport.load_plugins", "dds.transport.wan_plugin.wan",
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.load_plugins\n");
        publisher_shutdown(participant);
    }

    /* library */
    retcode = DDSPropertyQosPolicyHelper::add_property(
        participant_qos.property,
        "dds.transport.wan_plugin.wan.library",
#ifdef RTI_WIN32
        "nddstransportwan.dll",
#else
        "libnddstransportwan.so",
#endif
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.library\n");
        publisher_shutdown(participant);
    }

    /* create function */
    retcode = DDSPropertyQosPolicyHelper::add_property(
        participant_qos.property,
        "dds.transport.wan_plugin.wan.create_function",
        "NDDS_Transport_WAN_create",
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.create_function\n");
        publisher_shutdown(participant);
    }

    /* plugin properties */
#ifdef USE_SECURITY
    printf("Enabling secure WAN transport\n");
    retcode = DDSPropertyQosPolicyHelper::add_property(
        participant_qos.property,
```

```
        "dds.transport.wan_plugin.wan.enable_security", "1",
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.enable_security\n");
        publisher_shutdown(participant);
    }
    retcode = DDSPropertyQosPolicyHelper::add_property(
        participant_qos.property,
        "dds.transport.wan_plugin.wan.tls.verify.ca_file", "cacert.pem",
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.tls.verify.ca_file\n");
        publisher_shutdown(participant);
    }
    retcode = DDSPropertyQosPolicyHelper::add_property(
        participant_qos.property,
        "dds.transport.wan_plugin.wan.tls.identity.certificate_chain_file",
        "peer1.pem", DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.tls.identity.certificate_chain_file\n");
        publisher_shutdown(participant);
    }
#endif /* USE_SECURITY */

    retcode = DDSPropertyQosPolicyHelper::add_property(
        participant_qos.property,
        "dds.transport.wan_plugin.wan.server", WAN_SERVER,
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.server\n");
        publisher_shutdown(participant);
    }

    retcode = DDSPropertyQosPolicyHelper::add_property(
        participant_qos.property,
        "dds.transport.wan_plugin.wan.transport_instance_id", WAN_ID,
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.transport_instance_id\n");
        publisher_shutdown(participant);
    }

    participant = DDSTheParticipantFactory->create_participant(
        domainId, participant_qos,
        NULL /* listener */, DDS_STATUS_MASK_NONE);
    if (participant == NULL) {
        printf("create_participant error\n");
        publisher_shutdown(participant);
        return -1;
    }

    /* To customize publisher QoS, use
       participant->get_default_publisher_qos() */
    publisher = participant->create_publisher(
        DDS_PUBLISHER_QOS_DEFAULT, NULL /* listener */, DDS_STATUS_MASK_NONE);
    if (publisher == NULL) {
        printf("create_publisher error\n");
```

```
        publisher_shutdown(participant);
        return -1;
    }

    /* Register type before creating topic */
    type_name = HelloWorldTypeSupport::get_type_name();
    retcode = HelloWorldTypeSupport::register_type(
        participant, type_name);
    if (retcode != DDS_RETCODE_OK) {
        printf("register_type error %d\n", retcode);
        publisher_shutdown(participant);
        return -1;
    }

    /* To customize topic QoS, use
       participant->get_default_topic_qos() */
    topic = participant->create_topic(
        "Example HelloWorld",
        type_name, DDS_TOPIC_QOS_DEFAULT, NULL /* listener */,
        DDS_STATUS_MASK_NONE);
    if (topic == NULL) {
        printf("create_topic error\n");
        publisher_shutdown(participant);
        return -1;
    }

    /* To customize data writer QoS, use
       publisher->get_default_datawriter_qos() */
    writer = publisher->create_datawriter(
        topic, DDS_DATAWRITER_QOS_DEFAULT, NULL /* listener */,
        DDS_STATUS_MASK_NONE);
    if (writer == NULL) {
        printf("create_datawriter error\n");
        publisher_shutdown(participant);
        return -1;
    }
    HelloWorld_writer = HelloWorldDataWriter::narrow(writer);
    if (HelloWorld_writer == NULL) {
        printf("DataWriter narrow error\n");
        publisher_shutdown(participant);
        return -1;
    }

    /* Create data sample for writing */
    instance = HelloWorldTypeSupport::create_data();
    if (instance == NULL) {
        printf("HelloWorldTypeSupport::create_data error\n");
        publisher_shutdown(participant);
        return -1;
    }

    /* For data type that has key, if the same instance is going to be
       written multiple times, initialize the key here
       and register the keyed instance prior to writing */
/*
    instance_handle = HelloWorld_writer->register_instance(*instance);
*/
```

```
    /* Main loop */
    for (count=0; (sample_count == 0) || (count < sample_count); ++count) {

        printf("Writing HelloWorld to WAN, count %d\n", count);

        /* Modify the data to be sent here */
        sprintf(instance->msg, "Hello Wide Area World! (%d)", count);

        retcode = HelloWorld_writer->write(*instance, instance_handle);
        if (retcode != DDS_RETCODE_OK) {
            printf("write error %d\n", retcode);
        }

        NDDSUtility::sleep(send_period);
    }

/*
    retcode = HelloWorld_writer->unregister_instance(
        *instance, instance_handle);
    if (retcode != DDS_RETCODE_OK) {
        printf("unregister instance error %d\n", retcode);
    }
*/

    /* Delete data sample */
    retcode = HelloWorldTypeSupport::delete_data(instance);
    if (retcode != DDS_RETCODE_OK) {
        printf("HelloWorldTypeSupport::delete_data error %d\n", retcode);
    }

    /* Delete all entities */
    return publisher_shutdown(participant);
}

#if defined(RTI_WINCE)
int wmain(int argc, wchar_t** argv)
{
    int domainId = 0;
    int sample_count = 0; /* infinite loop */

    if (argc >= 2) {
        domainId = _wtoi(argv[1]);
    }
    if (argc >= 3) {
        sample_count = _wtoi(argv[2]);
    }

    /* Uncomment this to turn on additional logging
    NDDSConfigLogger::get_instance()->
        set_verbosity_by_category(NDDS_CONFIG_LOG_CATEGORY_API,
                                  NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
    */

    return publisher_main(domainId, sample_count);
}
```

```
#elif !(defined(RTI_VXWORKS) && !defined(__RTP__)) && !defined(RTI_PSOS)
int main(int argc, char *argv[])
{
    int domainId = 0;
    int sample_count = 0; /* infinite loop */

    if (argc >= 2) {
        domainId = atoi(argv[1]);
    }
    if (argc >= 3) {
        sample_count = atoi(argv[2]);
    }

    /* Uncomment this to turn on additional logging
    NDDSConfigLogger::get_instance()->
        set_verbosity_by_category(NDDS_CONFIG_LOG_CATEGORY_API,
                                  NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
    */

    return publisher_main(domainId, sample_count);
}
#endif
```

4.21 HelloWorld_subscriber.cxx

[$(NDDSHOME)/example/CPP/helloWorldWAN/HelloWorld_subscriber.cxx]

```
/* HelloWorld_subscriber.cxx

   A subscription example

   This file is derived from code automatically generated by the rtiddsgen
   command:

   rtiddsgen -language C++ -example <arch> HelloWorld.idl

   Example subscription of type HelloWorld automatically generated by
   'rtiddsgen'. To test them follow these steps:

   (1) Compile this file and the example publication.

   (2) Start the subscription on the same domain used for RTI Connext with the command
       objs/<arch>/HelloWorld_subscriber <domain_id> <sample_count>

   (3) Start the publication on the same domain used for RTI Connext
       with the command
       objs/<arch>/HelloWorld_publisher <domain_id> <sample_count>

   (4) [Optional] Specify the list of discovery initial peers and
       multicast receive addresses via an environment variable or a file
       (in the current working directory) called NDDS_DISCOVERY_PEERS.

   You can run any number of publishers and subscribers programs, and can
   add and remove them dynamically from the domain.


   Example:

       To run the example application on domain <domain_id>:

       On Unix:

       objs/<arch>/HelloWorld_publisher <domain_id>
       objs/<arch>/HelloWorld_subscriber <domain_id>

       On Windows:

       objs\<arch>\HelloWorld_publisher <domain_id>
       objs\<arch>\HelloWorld_subscriber <domain_id>


modification history
------------ -------
*/

#include <stdio.h>
#include <stdlib.h>
#include "ndds/ndds_cpp.h"
#include "HelloWorld.h"
```

```
#include "HelloWorldSupport.h"

/* set this to enable security using DTLS */
/*#define USE_SECURITY*/

/* set this value to the address of your WAN Rendezvous server */
#define WAN_SERVER "127.0.0.1"

/* your publisher and subscriber should have unique IDs */
#define WAN_ID "2"

class HelloWorldListener : public DDSDataReaderListener {
  public:
    virtual void on_requested_deadline_missed(
        DDSDataReader* /*reader*/,
        const DDS_RequestedDeadlineMissedStatus& /*status*/) {}

    virtual void on_requested_incompatible_qos(
        DDSDataReader* /*reader*/,
        const DDS_RequestedIncompatibleQosStatus& /*status*/) {}

    virtual void on_sample_rejected(
        DDSDataReader* /*reader*/,
        const DDS_SampleRejectedStatus& /*status*/) {}

    virtual void on_liveliness_changed(
        DDSDataReader* /*reader*/,
        const DDS_LivelinessChangedStatus& /*status*/) {}

    virtual void on_sample_lost(
        DDSDataReader* /*reader*/,
        const DDS_SampleLostStatus& /*status*/) {}

    virtual void on_subscription_matched(
        DDSDataReader* /*reader*/,
        const DDS_SubscriptionMatchedStatus& /*status*/) {}

    virtual void on_data_available(DDSDataReader* reader);
};

void HelloWorldListener::on_data_available(DDSDataReader* reader)
{
    HelloWorldDataReader *HelloWorld_reader = NULL;
    HelloWorldSeq data_seq;
    DDS_SampleInfoSeq info_seq;
    DDS_ReturnCode_t retcode;
    int i;

    HelloWorld_reader = HelloWorldDataReader::narrow(reader);
    if (HelloWorld_reader == NULL) {
        printf("DataReader narrow error\n");
        return;
    }

    retcode = HelloWorld_reader->take(
        data_seq, info_seq, DDS_LENGTH_UNLIMITED,
        DDS_ANY_SAMPLE_STATE, DDS_ANY_VIEW_STATE, DDS_ANY_INSTANCE_STATE);
```

```
    if (retcode == DDS_RETCODE_NO_DATA) {
        return;
    } else if (retcode != DDS_RETCODE_OK) {
        printf("take error %d\n", retcode);
        return;
    }

    for (i = 0; i < data_seq.length(); ++i) {
        if (info_seq[i].valid_data) {
            HelloWorldTypeSupport::print_data(&data_seq[i]);
        }
    }

    retcode = HelloWorld_reader->return_loan(data_seq, info_seq);
    if (retcode != DDS_RETCODE_OK) {
        printf("return loan error %d\n", retcode);
    }
}

/* Delete all entities */
static int subscriber_shutdown(
    DDSDomainParticipant *participant)
{
    DDS_ReturnCode_t retcode;
    int status = 0;

    if (participant != NULL) {
        retcode = participant->delete_contained_entities();
        if (retcode != DDS_RETCODE_OK) {
            printf("delete_contained_entities error %d\n", retcode);
            status = -1;
        }

        retcode = DDSTheParticipantFactory->delete_participant(participant);
        if (retcode != DDS_RETCODE_OK) {
            printf("delete_participant error %d\n", retcode);
            status = -1;
        }
    }

    /* RTI Connext provides finalize_instance() method on
       domain participant factory for people who want to release memory used
       by the participant factory singleton. Uncomment the following block of
       code for clean destruction of the singleton. */
/*
    retcode = DDSDomainParticipantFactory::finalize_instance();
    if (retcode != DDS_RETCODE_OK) {
        printf("finalize_instance error %d\n", retcode);
        status = -1;
    }
*/
    return status;
}

extern "C" int subscriber_main(int domainId, int sample_count)
{
```

```
    DDSDomainParticipant *participant = NULL;
    DDSSubscriber *subscriber = NULL;
    DDSTopic *topic = NULL;
    HelloWorldListener *reader_listener = NULL;
    DDSDataReader *reader = NULL;
    DDS_ReturnCode_t retcode;
    const char *type_name = NULL;
    int count = 0;
    struct DDS_Duration_t receive_period = {4,0};
    int status = 0;

    struct DDS_DomainParticipantQos participant_qos;

    /* Get default participant QoS from participant factory */
    retcode = DDSDomainParticipantFactory::get_instance()->get_default_participant_qos(
        participant_qos);
    if (retcode != DDS_RETCODE_OK) {
        printf("could not get default participant qos\n");
        subscriber_shutdown(participant);
        return -1;
    }

    /* Disable builtin transports */
    participant_qos.transport_builtin.mask = DDS_TRANSPORTBUILTIN_MASK_NONE;

    /* Set up property Qos to load plugin */
    retcode = DDSPropertyQosPolicyHelper::add_property(
        participant_qos.property,
        "dds.transport.load_plugins", "dds.transport.wan_plugin.wan",
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.load_plugins\n");
        subscriber_shutdown(participant);
    }

    /* library */
    retcode = DDSPropertyQosPolicyHelper::add_property(
        participant_qos.property,
        "dds.transport.wan_plugin.wan.library",
#ifdef RTI_WIN32
        "nddstransportwan.dll",
#else
        "libnddstransportwan.so",
#endif
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.library\n");
        subscriber_shutdown(participant);
    }

    /* create function */
    retcode = DDSPropertyQosPolicyHelper::add_property(
        participant_qos.property,
        "dds.transport.wan_plugin.wan.create_function",
        "NDDS_Transport_WAN_create",
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
```

```
        printf("Failed to add property dds.transport.wan_plugin.wan.create_function\n");
        subscriber_shutdown(participant);
    }

    /* plugin properties */
#ifdef USE_SECURITY
    printf("Enabling secure WAN transport\n");
    retcode = DDSPropertyQosPolicyHelper::add_property(
        participant_qos.property,
        "dds.transport.wan_plugin.wan.enable_security", "1",
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.enable_security\n");
        subscriber_shutdown(participant);
    }
    retcode = DDSPropertyQosPolicyHelper::add_property(
        participant_qos.property,
        "dds.transport.wan_plugin.wan.tls.verify.ca_file", "cacert.pem",
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.tls.verify.ca_file\n");
        subscriber_shutdown(participant);
    }
    retcode = DDSPropertyQosPolicyHelper::add_property(
        participant_qos.property,
        "dds.transport.wan_plugin.wan.tls.identity.certificate_chain_file",
        "peer2.pem", DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.tls.identity.certificate_chain_file\n");
        subscriber_shutdown(participant);
    }
#endif /* USE_SECURITY */

    retcode = DDSPropertyQosPolicyHelper::add_property(
        participant_qos.property,
        "dds.transport.wan_plugin.wan.server", WAN_SERVER,
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.server\n");
        subscriber_shutdown(participant);
    }

    retcode = DDSPropertyQosPolicyHelper::add_property(
        participant_qos.property,
        "dds.transport.wan_plugin.wan.transport_instance_id", WAN_ID,
        DDS_BOOLEAN_FALSE);
    if (retcode != DDS_RETCODE_OK) {
        printf("Failed to add property dds.transport.wan_plugin.wan.transport_instance_id\n");
        subscriber_shutdown(participant);
    }

    participant = DDSTheParticipantFactory->create_participant(
        domainId, participant_qos,
        NULL /* listener */, DDS_STATUS_MASK_NONE);
    if (participant == NULL) {
        printf("create_participant error\n");
        subscriber_shutdown(participant);
```

```
    return -1;
}

/* To customize subscriber QoS, use
   participant->get_default_subscriber_qos() */
subscriber = participant->create_subscriber(
    DDS_SUBSCRIBER_QOS_DEFAULT, NULL /* listener */, DDS_STATUS_MASK_NONE);
if (subscriber == NULL) {
    printf("create_subscriber error\n");
    subscriber_shutdown(participant);
    return -1;
}

/* Register type before creating topic */
type_name = HelloWorldTypeSupport::get_type_name();
retcode = HelloWorldTypeSupport::register_type(
    participant, type_name);
if (retcode != DDS_RETCODE_OK) {
    printf("register_type error %d\n", retcode);
    subscriber_shutdown(participant);
    return -1;
}

/* To customize topic QoS, use
   participant->get_default_topic_qos() */
topic = participant->create_topic(
    "Example HelloWorld",
    type_name, DDS_TOPIC_QOS_DEFAULT, NULL /* listener */,
    DDS_STATUS_MASK_NONE);
if (topic == NULL) {
    printf("create_topic error\n");
    subscriber_shutdown(participant);
    return -1;
}

/* Create data reader listener */
reader_listener = new HelloWorldListener();
if (reader_listener ==  NULL) {
    printf("listener instantiation error\n");
    subscriber_shutdown(participant);
    return -1;
}

/* To customize data reader QoS, use
   subscriber->get_default_datareader_qos() */
reader = subscriber->create_datareader(
    topic, DDS_DATAREADER_QOS_DEFAULT, reader_listener,
    DDS_STATUS_MASK_ALL);
if (reader == NULL) {
    printf("create_datareader error\n");
    subscriber_shutdown(participant);
    delete reader_listener;
    return -1;
}

/* Main loop */
for (count=0; (sample_count == 0) || (count < sample_count); ++count) {
```

```
        printf("HelloWorld subscriber sleeping for %d sec...\n",
               receive_period.sec);

        NDDSUtility::sleep(receive_period);
    }

    /* Delete all entities */
    status = subscriber_shutdown(participant);
    delete reader_listener;

    return status;
}

#if defined(RTI_WINCE)
int wmain(int argc, wchar_t** argv)
{
    int domainId = 0;
    int sample_count = 0; /* infinite loop */

    if (argc >= 2) {
        domainId = _wtoi(argv[1]);
    }
    if (argc >= 3) {
        sample_count = _wtoi(argv[2]);
    }

    /* Uncomment this to turn on additional logging
    NDDSConfigLogger::get_instance()->
        set_verbosity_by_category(NDDS_CONFIG_LOG_CATEGORY_API,
                                  NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
    */

    return subscriber_main(domainId, sample_count);
}

#elif !(defined(RTI_VXWORKS) && !defined(__RTP__)) && !defined(RTI_PSOS)
int main(int argc, char *argv[])
{
    int domainId = 0;
    int sample_count = 0; /* infinite loop */

    if (argc >= 2) {
        domainId = atoi(argv[1]);
    }
    if (argc >= 3) {
        sample_count = atoi(argv[2]);
    }


    /* Uncomment this to turn on additional logging
    NDDSConfigLogger::get_instance()->
        set_verbosity_by_category(NDDS_CONFIG_LOG_CATEGORY_API,
                                  NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
    */

    return subscriber_main(domainId, sample_count);
```

```
}
#endif
```

4.22 HelloWorldPlugin.cxx

[$(NDDSHOME)/example/CPP/helloWorldWAN/HelloWorldPlugin.h]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from HelloWorld.idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/

#ifndef HelloWorldPlugin_1436885487_h
#define HelloWorldPlugin_1436885487_h

#include "HelloWorld.h"




struct RTICdrStream;

#ifndef pres_typePlugin_h
#include "pres/pres_typePlugin.h"
#endif


#if (defined(RTI_WIN32) || defined (RTI_WINCE)) && defined(NDDS_USER_DLL_EXPORT)
/* If the code is building on Windows, start exporting symbols.
*/
#undef NDDSUSERDllExport
#define NDDSUSERDllExport __declspec(dllexport)
#endif


#ifdef __cplusplus
extern "C" {
#endif

#define HelloWorld_LAST_MEMBER_ID 0

#define HelloWorldPlugin_get_sample PRESTypePluginDefaultEndpointData_getSample
#define HelloWorldPlugin_return_sample PRESTypePluginDefaultEndpointData_returnSample
#define HelloWorldPlugin_get_buffer PRESTypePluginDefaultEndpointData_getBuffer
#define HelloWorldPlugin_return_buffer PRESTypePluginDefaultEndpointData_returnBuffer


#define HelloWorldPlugin_create_sample PRESTypePluginDefaultEndpointData_createSample
#define HelloWorldPlugin_destroy_sample PRESTypePluginDefaultEndpointData_deleteSample

/* --------------------------------------------------------------------------------
    Support functions:
 * -------------------------------------------------------------------------------- */

NDDSUSERDllExport extern HelloWorld*
```

```
HelloWorldPluginSupport_create_data_ex(RTIBool allocate_pointers);

NDDSUSERDllExport extern HelloWorld*
HelloWorldPluginSupport_create_data(void);

NDDSUSERDllExport extern RTIBool
HelloWorldPluginSupport_copy_data(
    HelloWorld *out,
    const HelloWorld *in);

NDDSUSERDllExport extern void
HelloWorldPluginSupport_destroy_data_ex(
    HelloWorld *sample,RTIBool deallocate_pointers);

NDDSUSERDllExport extern void
HelloWorldPluginSupport_destroy_data(
    HelloWorld *sample);

NDDSUSERDllExport extern void
HelloWorldPluginSupport_print_data(
    const HelloWorld *sample,
    const char *desc,
    unsigned int indent);


/* ----------------------------------------------------------------------
    Callback functions:
 * ---------------------------------------------------------------------- */

NDDSUSERDllExport extern PRESTypePluginParticipantData
HelloWorldPlugin_on_participant_attached(
    void *registration_data,
    const struct PRESTypePluginParticipantInfo *participant_info,
    RTIBool top_level_registration,
    void *container_plugin_context,
    RTICdrTypeCode *typeCode);

NDDSUSERDllExport extern void
HelloWorldPlugin_on_participant_detached(
    PRESTypePluginParticipantData participant_data);

NDDSUSERDllExport extern PRESTypePluginEndpointData
HelloWorldPlugin_on_endpoint_attached(
    PRESTypePluginParticipantData participant_data,
    const struct PRESTypePluginEndpointInfo *endpoint_info,
    RTIBool top_level_registration,
    void *container_plugin_context);

NDDSUSERDllExport extern void
HelloWorldPlugin_on_endpoint_detached(
    PRESTypePluginEndpointData endpoint_data);


NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_copy_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *out,
```

```
    const HelloWorld *in);

/* ------------------------------------------------------------------------------------
    (De)Serialize functions:
 * ------------------------------------------------------------------------------------ */

NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_serialize(
    PRESTypePluginEndpointData endpoint_data,
    const HelloWorld *sample,
    struct RTICdrStream *stream,
    RTIBool serialize_encapsulation,
    RTIEncapsulationId encapsulation_id,
    RTIBool serialize_sample,
    void *endpoint_plugin_qos);

NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_deserialize_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_sample,
    void *endpoint_plugin_qos);


NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_deserialize(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld **sample,
    RTIBool * drop_sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_sample,
    void *endpoint_plugin_qos);




NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_skip(
    PRESTypePluginEndpointData endpoint_data,
    struct RTICdrStream *stream,
    RTIBool skip_encapsulation,
    RTIBool skip_sample,
    void *endpoint_plugin_qos);

NDDSUSERDllExport extern unsigned int
HelloWorldPlugin_get_serialized_sample_max_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment);

NDDSUSERDllExport extern unsigned int
HelloWorldPlugin_get_serialized_sample_min_size(
    PRESTypePluginEndpointData endpoint_data,
```

```
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment);

NDDSUSERDllExport extern unsigned int
HelloWorldPlugin_get_serialized_sample_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment,
    const HelloWorld * sample);



/* ----------------------------------------------------------------------------
    Key Management functions:
 * ---------------------------------------------------------------------------- */

NDDSUSERDllExport extern PRESTypePluginKeyKind
HelloWorldPlugin_get_key_kind(void);

NDDSUSERDllExport extern unsigned int
HelloWorldPlugin_get_serialized_key_max_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment);

NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_serialize_key(
    PRESTypePluginEndpointData endpoint_data,
    const HelloWorld *sample,
    struct RTICdrStream *stream,
    RTIBool serialize_encapsulation,
    RTIEncapsulationId encapsulation_id,
    RTIBool serialize_key,
    void *endpoint_plugin_qos);

NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_deserialize_key_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld * sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_key,
    void *endpoint_plugin_qos);


NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_deserialize_key(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld ** sample,
    RTIBool * drop_sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_key,
    void *endpoint_plugin_qos);
```

```
NDDSUSERDllExport extern RTIBool
HelloWorldPlugin_serialized_sample_to_key(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_key,
    void *endpoint_plugin_qos);


/* Plugin Functions */
NDDSUSERDllExport extern struct PRESTypePlugin*
HelloWorldPlugin_new(void);

NDDSUSERDllExport extern void
HelloWorldPlugin_delete(struct PRESTypePlugin *);

#ifdef __cplusplus
}
#endif


#if (defined(RTI_WIN32) || defined (RTI_WINCE)) && defined(NDDS_USER_DLL_EXPORT)
/* If the code is building on Windows, stop exporting symbols.
*/
#undef NDDSUSERDllExport
#define NDDSUSERDllExport
#endif

#endif /* HelloWorldPlugin_1436885487_h */
```

[$(NDDSHOME)/example/CPP/helloWorldWAN/HelloWorldPlugin.cxx]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from HelloWorld.idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/


#include <string.h>

#ifdef __cplusplus
#ifndef ndds_cpp_h
  #include "ndds/ndds_cpp.h"
#endif
#else
#ifndef ndds_c_h
  #include "ndds/ndds_c.h"
#endif
#endif
```

```
#ifndef osapi_type_h
  #include "osapi/osapi_type.h"
#endif
#ifndef osapi_heap_h
  #include "osapi/osapi_heap.h"
#endif

#ifndef osapi_utility_h
  #include "osapi/osapi_utility.h"
#endif

#ifndef cdr_type_h
  #include "cdr/cdr_type.h"
#endif

#ifndef cdr_type_object_h
  #include "cdr/cdr_typeObject.h"
#endif

#ifndef cdr_encapsulation_h
  #include "cdr/cdr_encapsulation.h"
#endif

#ifndef cdr_stream_h
  #include "cdr/cdr_stream.h"
#endif

#ifndef pres_typePlugin_h
  #include "pres/pres_typePlugin.h"
#endif



#include "HelloWorldPlugin.h"


/* ----------------------------------------------------------------------------------------
 *  Type HelloWorld
 * ---------------------------------------------------------------------------------------- */

/* ----------------------------------------------------------------------------------------
    Support functions:
 * ---------------------------------------------------------------------------------------- */

HelloWorld *
HelloWorldPluginSupport_create_data_ex(RTIBool allocate_pointers){
    HelloWorld *sample = NULL;

    RTIOsapiHeap_allocateStructure(
        &sample, HelloWorld);

    if(sample != NULL) {
        if (!HelloWorld_initialize_ex(sample,allocate_pointers, RTI_TRUE)) {
            RTIOsapiHeap_freeStructure(sample);
            return NULL;
        }
```

```
    }
    return sample;
}


HelloWorld *
HelloWorldPluginSupport_create_data(void)
{
    return HelloWorldPluginSupport_create_data_ex(RTI_TRUE);
}


void
HelloWorldPluginSupport_destroy_data_ex(
    HelloWorld *sample,RTIBool deallocate_pointers) {

    HelloWorld_finalize_ex(sample,deallocate_pointers);

    RTIOsapiHeap_freeStructure(sample);
}


void
HelloWorldPluginSupport_destroy_data(
    HelloWorld *sample) {

    HelloWorldPluginSupport_destroy_data_ex(sample,RTI_TRUE);

}


RTIBool
HelloWorldPluginSupport_copy_data(
    HelloWorld *dst,
    const HelloWorld *src)
{
    return HelloWorld_copy(dst,src);
}


void
HelloWorldPluginSupport_print_data(
    const HelloWorld *sample,
    const char *desc,
    unsigned int indent_level)
{


    RTICdrType_printIndent(indent_level);

    if (desc != NULL) {
      RTILog_debug("%s:\n", desc);
    } else {
      RTILog_debug("\n");
    }

    if (sample == NULL) {
```

```
      RTILog_debug("NULL\n");
      return;
    }


    if (&sample->msg==NULL) {
        RTICdrType_printString(
            NULL, "msg", indent_level + 1);
    } else {
        RTICdrType_printString(
            sample->msg, "msg", indent_level + 1);
    }



}


/* ----------------------------------------------------------------------------
    Callback functions:
 * ---------------------------------------------------------------------------- */


PRESTypePluginParticipantData
HelloWorldPlugin_on_participant_attached(
    void *registration_data,
    const struct PRESTypePluginParticipantInfo *participant_info,
    RTIBool top_level_registration,
    void *container_plugin_context,
    RTICdrTypeCode *type_code)
{

    if (registration_data) {} /* To avoid warnings */
    if (participant_info) {} /* To avoid warnings */
    if (top_level_registration) {} /* To avoid warnings */
    if (container_plugin_context) {} /* To avoid warnings */
    if (type_code) {} /* To avoid warnings */
    return PRESTypePluginDefaultParticipantData_new(participant_info);

}


void
HelloWorldPlugin_on_participant_detached(
    PRESTypePluginParticipantData participant_data)
{

  PRESTypePluginDefaultParticipantData_delete(participant_data);
}


PRESTypePluginEndpointData
HelloWorldPlugin_on_endpoint_attached(
    PRESTypePluginParticipantData participant_data,
    const struct PRESTypePluginEndpointInfo *endpoint_info,
    RTIBool top_level_registration,
```

```
    void *containerPluginContext)
{
    PRESTypePluginEndpointData epd = NULL;

    unsigned int serializedSampleMaxSize;

   if (top_level_registration) {} /* To avoid warnings */
   if (containerPluginContext) {} /* To avoid warnings */

    epd = PRESTypePluginDefaultEndpointData_new(
            participant_data,
            endpoint_info,
            (PRESTypePluginDefaultEndpointDataCreateSampleFunction)
            HelloWorldPluginSupport_create_data,
            (PRESTypePluginDefaultEndpointDataDestroySampleFunction)
            HelloWorldPluginSupport_destroy_data,
            NULL, NULL);

    if (epd == NULL) {
        return NULL;
    }



    if (endpoint_info->endpointKind == PRES_TYPEPLUGIN_ENDPOINT_WRITER) {
        serializedSampleMaxSize = HelloWorldPlugin_get_serialized_sample_max_size(
            epd,RTI_FALSE,RTI_CDR_ENCAPSULATION_ID_CDR_BE,0);

        PRESTypePluginDefaultEndpointData_setMaxSizeSerializedSample(epd, serializedSampleMaxSize);

        if (PRESTypePluginDefaultEndpointData_createWriterPool(
                epd,
                endpoint_info,
            (PRESTypePluginGetSerializedSampleMaxSizeFunction)
                HelloWorldPlugin_get_serialized_sample_max_size, epd,
            (PRESTypePluginGetSerializedSampleSizeFunction)
            HelloWorldPlugin_get_serialized_sample_size,
            epd) == RTI_FALSE) {
            PRESTypePluginDefaultEndpointData_delete(epd);
            return NULL;
        }
    }



    return epd;
}


void
HelloWorldPlugin_on_endpoint_detached(
    PRESTypePluginEndpointData endpoint_data)
{

    PRESTypePluginDefaultEndpointData_delete(endpoint_data);
}
```

```
RTIBool
HelloWorldPlugin_copy_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *dst,
    const HelloWorld *src)
{
    if (endpoint_data) {} /* To avoid warnings */
    return HelloWorldPluginSupport_copy_data(dst,src);
}

/* ------------------------------------------------------------------------------------
    (De)Serialize functions:
 * ------------------------------------------------------------------------------------ */

unsigned int
HelloWorldPlugin_get_serialized_sample_max_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment);


RTIBool
HelloWorldPlugin_serialize(
    PRESTypePluginEndpointData endpoint_data,
    const HelloWorld *sample,
    struct RTICdrStream *stream,
    RTIBool serialize_encapsulation,
    RTIEncapsulationId encapsulation_id,
    RTIBool serialize_sample,
    void *endpoint_plugin_qos)
{
    char * position = NULL;
    RTIBool retval = RTI_TRUE;

    if (endpoint_data) {} /* To avoid warnings */
    if (endpoint_plugin_qos) {} /* To avoid warnings */


    if(serialize_encapsulation) {

        if (!RTICdrStream_serializeAndSetCdrEncapsulation(stream, encapsulation_id)) {
            return RTI_FALSE;
        }

        position = RTICdrStream_resetAlignment(stream);

    }


    if(serialize_sample) {

    if (!RTICdrStream_serializeString(
        stream, sample->msg, (128) + 1)) {
        return RTI_FALSE;
```

```
    }


    }


    if(serialize_encapsulation) {
        RTICdrStream_restoreAlignment(stream,position);
    }


  return retval;
}


RTIBool
HelloWorldPlugin_deserialize_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_sample,
    void *endpoint_plugin_qos)
{
    char * position = NULL;

    RTIBool done = RTI_FALSE;

    if (endpoint_data) {} /* To avoid warnings */
    if (endpoint_plugin_qos) {} /* To avoid warnings */


    if(deserialize_encapsulation) {
        /* Deserialize encapsulation */
        if (!RTICdrStream_deserializeAndSetCdrEncapsulation(stream)) {
            return RTI_FALSE;
        }

        position = RTICdrStream_resetAlignment(stream);

    }


    if(deserialize_sample) {
        HelloWorld_initialize_ex(sample, RTI_FALSE, RTI_FALSE);

    if (!RTICdrStream_deserializeString(
        stream, sample->msg, (128) + 1)) {
        goto fin;
    }


    }

    done = RTI_TRUE;
fin:
    if (done != RTI_TRUE && RTICdrStream_getRemainder(stream) >  0) {
```

```
        return RTI_FALSE;
    }

    if(deserialize_encapsulation) {
        RTICdrStream_restoreAlignment(stream,position);
    }


    return RTI_TRUE;
}



RTIBool
HelloWorldPlugin_deserialize(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld **sample,
    RTIBool * drop_sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_sample,
    void *endpoint_plugin_qos)
{

    if (drop_sample) {} /* To avoid warnings */

    return HelloWorldPlugin_deserialize_sample(
        endpoint_data, (sample != NULL)?*sample:NULL,
        stream, deserialize_encapsulation, deserialize_sample,
        endpoint_plugin_qos);

}



RTIBool HelloWorldPlugin_skip(
    PRESTypePluginEndpointData endpoint_data,
    struct RTICdrStream *stream,
    RTIBool skip_encapsulation,
    RTIBool skip_sample,
    void *endpoint_plugin_qos)
{
    char * position = NULL;

    RTIBool done = RTI_FALSE;

    if (endpoint_data) {} /* To avoid warnings */
    if (endpoint_plugin_qos) {} /* To avoid warnings */


    if(skip_encapsulation) {
        if (!RTICdrStream_skipEncapsulation(stream)) {
            return RTI_FALSE;
        }
```

```
        position = RTICdrStream_resetAlignment(stream);

    }

    if (skip_sample) {

    if (!RTICdrStream_skipString(stream, (128) + 1)) {
        goto fin;
    }



    }


    done = RTI_TRUE;
fin:
    if (done != RTI_TRUE && RTICdrStream_getRemainder(stream) >  0) {
        return RTI_FALSE;
    }

    if(skip_encapsulation) {
        RTICdrStream_restoreAlignment(stream,position);
    }


    return RTI_TRUE;
}


unsigned int
HelloWorldPlugin_get_serialized_sample_max_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment)
{

    unsigned int initial_alignment = current_alignment;

    unsigned int encapsulation_size = current_alignment;

    if (endpoint_data) {} /* To avoid warnings */


    if (include_encapsulation) {

        if (!RTICdrEncapsulation_validEncapsulationId(encapsulation_id)) {
            return 1;
        }

        RTICdrStream_getEncapsulationSize(encapsulation_size);
        encapsulation_size -= current_alignment;
        current_alignment = 0;
        initial_alignment = 0;
```

```
    }


    current_alignment +=  RTICdrType_getStringMaxSizeSerialized(
        current_alignment, (128) + 1);


    if (include_encapsulation) {
        current_alignment += encapsulation_size;
    }

    return current_alignment - initial_alignment;
}


unsigned int
HelloWorldPlugin_get_serialized_sample_min_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment)
{

    unsigned int initial_alignment = current_alignment;

    unsigned int encapsulation_size = current_alignment;

    if (endpoint_data) {} /* To avoid warnings */


    if (include_encapsulation) {

        if (!RTICdrEncapsulation_validEncapsulationId(encapsulation_id)) {
            return 1;
        }

        RTICdrStream_getEncapsulationSize(encapsulation_size);
        encapsulation_size -= current_alignment;
        current_alignment = 0;
        initial_alignment = 0;

    }


    current_alignment +=  RTICdrType_getStringMaxSizeSerialized(
        current_alignment, 1);


    if (include_encapsulation) {
        current_alignment += encapsulation_size;
    }

    return current_alignment - initial_alignment;
}


/* Returns the size of the sample in its serialized form (in bytes).
```

```
 * It can also be an estimation in excess of the real buffer needed
 * during a call to the serialize() function.
 * The value reported does not have to include the space for the
 * encapsulation flags.
 */
unsigned int
HelloWorldPlugin_get_serialized_sample_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment,
    const HelloWorld * sample)
{

    unsigned int initial_alignment = current_alignment;

    unsigned int encapsulation_size = current_alignment;

    if (endpoint_data) {} /* To avoid warnings */
    if (sample) {} /* To avoid warnings */


    if (include_encapsulation) {

        if (!RTICdrEncapsulation_validEncapsulationId(encapsulation_id)) {
            return 1;
        }

        RTICdrStream_getEncapsulationSize(encapsulation_size);
        encapsulation_size -= current_alignment;
        current_alignment = 0;
        initial_alignment = 0;

    }


    current_alignment += RTICdrType_getStringSerializedSize(
        current_alignment, sample->msg);


    if (include_encapsulation) {
        current_alignment += encapsulation_size;
    }

    return current_alignment - initial_alignment;
}






/* ----------------------------------------------------------------------------
    Key Management functions:
 * ---------------------------------------------------------------------------- */
```

```
PRESTypePluginKeyKind
HelloWorldPlugin_get_key_kind(void)
{

    return PRES_TYPEPLUGIN_NO_KEY;

}


RTIBool
HelloWorldPlugin_serialize_key(
    PRESTypePluginEndpointData endpoint_data,
    const HelloWorld *sample,
    struct RTICdrStream *stream,
    RTIBool serialize_encapsulation,
    RTIEncapsulationId encapsulation_id,
    RTIBool serialize_key,
    void *endpoint_plugin_qos)
{
    char * position = NULL;

    if (endpoint_data) {} /* To avoid warnings */
    if (endpoint_plugin_qos) {} /* To avoid warnings */


    if(serialize_encapsulation) {

        if (!RTICdrStream_serializeAndSetCdrEncapsulation(stream, encapsulation_id)) {
            return RTI_FALSE;
        }


        position = RTICdrStream_resetAlignment(stream);

    }

    if(serialize_key) {

        if (!HelloWorldPlugin_serialize(
                endpoint_data,
                sample,
                stream,
                RTI_FALSE, encapsulation_id,
                RTI_TRUE,
                endpoint_plugin_qos)) {
            return RTI_FALSE;
        }

    }


    if(serialize_encapsulation) {
        RTICdrStream_restoreAlignment(stream,position);
    }
```

```
    return RTI_TRUE;
}


RTIBool HelloWorldPlugin_deserialize_key_sample(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_key,
    void *endpoint_plugin_qos)
{
    char * position = NULL;

    if (endpoint_data) {} /* To avoid warnings */
    if (endpoint_plugin_qos) {} /* To avoid warnings */


    if(deserialize_encapsulation) {
        /* Deserialize encapsulation */
        if (!RTICdrStream_deserializeAndSetCdrEncapsulation(stream)) {
            return RTI_FALSE;
        }


        position = RTICdrStream_resetAlignment(stream);

    }

    if (deserialize_key) {

        if (!HelloWorldPlugin_deserialize_sample(
                endpoint_data, sample, stream,
                RTI_FALSE, RTI_TRUE,
                endpoint_plugin_qos)) {
            return RTI_FALSE;
        }

    }


    if(deserialize_encapsulation) {
        RTICdrStream_restoreAlignment(stream,position);
    }


    return RTI_TRUE;
}



RTIBool HelloWorldPlugin_deserialize_key(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld **sample,
    RTIBool * drop_sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
```

```
    RTIBool deserialize_key,
    void *endpoint_plugin_qos)
{
    if (drop_sample) {} /* To avoid warnings */
    return HelloWorldPlugin_deserialize_key_sample(
        endpoint_data, (sample != NULL)?*sample:NULL, stream,
        deserialize_encapsulation, deserialize_key, endpoint_plugin_qos);
}



unsigned int
HelloWorldPlugin_get_serialized_key_max_size(
    PRESTypePluginEndpointData endpoint_data,
    RTIBool include_encapsulation,
    RTIEncapsulationId encapsulation_id,
    unsigned int current_alignment)
{

    unsigned int encapsulation_size = current_alignment;


    unsigned int initial_alignment = current_alignment;


    if (endpoint_data) {} /* To avoid warnings */


    if (include_encapsulation) {
        if (!RTICdrEncapsulation_validEncapsulationId(encapsulation_id)) {
            return 1;
        }


        RTICdrStream_getEncapsulationSize(encapsulation_size);
        encapsulation_size -= current_alignment;
        current_alignment = 0;
        initial_alignment = 0;

    }


    current_alignment += HelloWorldPlugin_get_serialized_sample_max_size(
        endpoint_data,RTI_FALSE, encapsulation_id, current_alignment);

    if (include_encapsulation) {
        current_alignment += encapsulation_size;
    }

    return current_alignment - initial_alignment;
}


RTIBool
HelloWorldPlugin_serialized_sample_to_key(
    PRESTypePluginEndpointData endpoint_data,
    HelloWorld *sample,
```

```
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_key,
    void *endpoint_plugin_qos)
{
    char * position = NULL;

    RTIBool done = RTI_FALSE;

    if (stream == NULL) goto fin; /* To avoid warnings */


    if(deserialize_encapsulation) {
        if (!RTICdrStream_deserializeAndSetCdrEncapsulation(stream)) {
            return RTI_FALSE;
        }

        position = RTICdrStream_resetAlignment(stream);

    }

    if (deserialize_key) {

        if (!HelloWorldPlugin_deserialize_sample(
            endpoint_data, sample, stream, RTI_FALSE,
            RTI_TRUE, endpoint_plugin_qos)) {
            return RTI_FALSE;
        }

    }


    done = RTI_TRUE;
fin:
    if (done != RTI_TRUE && RTICdrStream_getRemainder(stream) >  0) {
        return RTI_FALSE;
    }

    if(deserialize_encapsulation) {
        RTICdrStream_restoreAlignment(stream,position);
    }


    return RTI_TRUE;
}




/* ----------------------------------------------------------------------
 * Plug-in Installation Methods
 * ---------------------------------------------------------------------- */

struct PRESTypePlugin *HelloWorldPlugin_new(void)
{
    struct PRESTypePlugin *plugin = NULL;
    const struct PRESTypePluginVersion PLUGIN_VERSION =
```

```
    PRES_TYPE_PLUGIN_VERSION_2_0;

RTIOsapiHeap_allocateStructure(
    &plugin, struct PRESTypePlugin);
if (plugin == NULL) {
   return NULL;
}

plugin->version = PLUGIN_VERSION;

/* set up parent's function pointers */
plugin->onParticipantAttached =
    (PRESTypePluginOnParticipantAttachedCallback)
    HelloWorldPlugin_on_participant_attached;
plugin->onParticipantDetached =
    (PRESTypePluginOnParticipantDetachedCallback)
    HelloWorldPlugin_on_participant_detached;
plugin->onEndpointAttached =
    (PRESTypePluginOnEndpointAttachedCallback)
    HelloWorldPlugin_on_endpoint_attached;
plugin->onEndpointDetached =
    (PRESTypePluginOnEndpointDetachedCallback)
    HelloWorldPlugin_on_endpoint_detached;

plugin->copySampleFnc =
    (PRESTypePluginCopySampleFunction)
    HelloWorldPlugin_copy_sample;
plugin->createSampleFnc =
    (PRESTypePluginCreateSampleFunction)
    HelloWorldPlugin_create_sample;
plugin->destroySampleFnc =
    (PRESTypePluginDestroySampleFunction)
    HelloWorldPlugin_destroy_sample;

plugin->serializeFnc =
    (PRESTypePluginSerializeFunction)
    HelloWorldPlugin_serialize;
plugin->deserializeFnc =
    (PRESTypePluginDeserializeFunction)
    HelloWorldPlugin_deserialize;
plugin->getSerializedSampleMaxSizeFnc =
    (PRESTypePluginGetSerializedSampleMaxSizeFunction)
    HelloWorldPlugin_get_serialized_sample_max_size;
plugin->getSerializedSampleMinSizeFnc =
    (PRESTypePluginGetSerializedSampleMinSizeFunction)
    HelloWorldPlugin_get_serialized_sample_min_size;


plugin->getSampleFnc =
    (PRESTypePluginGetSampleFunction)
    HelloWorldPlugin_get_sample;
plugin->returnSampleFnc =
    (PRESTypePluginReturnSampleFunction)
    HelloWorldPlugin_return_sample;

plugin->getKeyKindFnc =
    (PRESTypePluginGetKeyKindFunction)
```

```
        HelloWorldPlugin_get_key_kind;


    /* These functions are only used for keyed types. As this is not a keyed
    type they are all set to NULL
    */
    plugin->serializeKeyFnc = NULL;
    plugin->deserializeKeyFnc = NULL;
    plugin->getKeyFnc = NULL;
    plugin->returnKeyFnc = NULL;
    plugin->instanceToKeyFnc = NULL;
    plugin->keyToInstanceFnc = NULL;
    plugin->getSerializedKeyMaxSizeFnc = NULL;
    plugin->instanceToKeyHashFnc = NULL;
    plugin->serializedSampleToKeyHashFnc = NULL;
    plugin->serializedKeyToKeyHashFnc = NULL;

    plugin->typeCode =  (struct RTICdrTypeCode *)HelloWorld_get_typecode();

    plugin->languageKind = PRES_TYPEPLUGIN_DDS_TYPE;

    /* Serialized buffer */
    plugin->getBuffer =
        (PRESTypePluginGetBufferFunction)
        HelloWorldPlugin_get_buffer;
    plugin->returnBuffer =
        (PRESTypePluginReturnBufferFunction)
        HelloWorldPlugin_return_buffer;
    plugin->getSerializedSampleSizeFnc =
        (PRESTypePluginGetSerializedSampleSizeFunction)
        HelloWorldPlugin_get_serialized_sample_size;

    plugin->endpointTypeName = HelloWorldTYPENAME;

    return plugin;
}

void
HelloWorldPlugin_delete(struct PRESTypePlugin *plugin)
{
    RTIOsapiHeap_freeStructure(plugin);
}
```

4.23 HelloWorldSupport.cxx

[$(NDDSHOME)/example/CPP/helloWorldWAN/HelloWorldSupport.h]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from HelloWorld.idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/

#ifndef HelloWorldSupport_1436885487_h
#define HelloWorldSupport_1436885487_h

/* Uses */
#include "HelloWorld.h"



#ifdef __cplusplus
#ifndef ndds_cpp_h
  #include "ndds/ndds_cpp.h"
#endif
#else
#ifndef ndds_c_h
  #include "ndds/ndds_c.h"
#endif
#endif



/* ========================================================================= */
#if (defined(RTI_WIN32) || defined (RTI_WINCE)) && defined(NDDS_USER_DLL_EXPORT)
  /* If the code is building on Windows, start exporting symbols.
   */
  #undef NDDSUSERDllExport
  #define NDDSUSERDllExport __declspec(dllexport)

#ifdef __cplusplus
  /* If we're building on Windows, explicitly import the superclasses of
   * the types declared below.
   */
  class __declspec(dllimport) DDSTypeSupport;
  class __declspec(dllimport) DDSDataWriter;
  class __declspec(dllimport) DDSDataReader;
#endif

#endif

#ifdef __cplusplus

DDS_TYPESUPPORT_CPP(HelloWorldTypeSupport, HelloWorld);

DDS_DATAWRITER_CPP(HelloWorldDataWriter, HelloWorld);
```

```
DDS_DATAREADER_CPP(HelloWorldDataReader, HelloWorldSeq, HelloWorld);


#else

DDS_TYPESUPPORT_C(HelloWorldTypeSupport, HelloWorld);
DDS_DATAWRITER_C(HelloWorldDataWriter, HelloWorld);
DDS_DATAREADER_C(HelloWorldDataReader, HelloWorldSeq, HelloWorld);

#endif

#if (defined(RTI_WIN32) || defined (RTI_WINCE)) && defined(NDDS_USER_DLL_EXPORT)
  /* If the code is building on Windows, stop exporting symbols.
   */
  #undef NDDSUSERDllExport
  #define NDDSUSERDllExport
#endif



#endif  /* HelloWorldSupport_1436885487_h */
```

[$(NDDSHOME)/example/CPP/helloWorldWAN/HelloWorldSupport.cxx]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from HelloWorld.idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/

#include "HelloWorldSupport.h"
#include "HelloWorldPlugin.h"



#ifdef __cplusplus
    #ifndef dds_c_log_impl_h
         #include "dds_c/dds_c_log_impl.h"
    #endif
#endif




/* ========================================================================= */
/* ------------------------------------------------------------------------- */
/* DDSDataWriter
*/

/* Requires */
#define TTYPENAME   HelloWorldTYPENAME

/* Defines */
```

```
#define TDataWriter HelloWorldDataWriter
#define TData       HelloWorld


#ifdef __cplusplus
#include "dds_cpp/generic/dds_cpp_data_TDataWriter.gen"
#else
#include "dds_c/generic/dds_c_data_TDataWriter.gen"
#endif


#undef TDataWriter
#undef TData

#undef TTYPENAME

/* ------------------------------------------------------------------ */
/* DDSDataReader
*/

/* Requires */
#define TTYPENAME   HelloWorldTYPENAME

/* Defines */
#define TDataReader HelloWorldDataReader
#define TDataSeq    HelloWorldSeq
#define TData       HelloWorld


#ifdef __cplusplus
#include "dds_cpp/generic/dds_cpp_data_TDataReader.gen"
#else
#include "dds_c/generic/dds_c_data_TDataReader.gen"
#endif


#undef TDataReader
#undef TDataSeq
#undef TData

#undef TTYPENAME

/* ------------------------------------------------------------------ */
/* TypeSupport

  <<IMPLEMENTATION >>

   Requires:  TTYPENAME,
              TPlugin_new
              TPlugin_delete
   Defines:   TTypeSupport, TData, TDataReader, TDataWriter
*/

/* Requires */
#define TTYPENAME    HelloWorldTYPENAME
#define TPlugin_new  HelloWorldPlugin_new
#define TPlugin_delete  HelloWorldPlugin_delete
```

```
/* Defines */
#define TTypeSupport HelloWorldTypeSupport
#define TData        HelloWorld
#define TDataReader  HelloWorldDataReader
#define TDataWriter  HelloWorldDataWriter
#ifdef __cplusplus



#include "dds_cpp/generic/dds_cpp_data_TTypeSupport.gen"



#else
#include "dds_c/generic/dds_c_data_TTypeSupport.gen"
#endif
#undef TTypeSupport
#undef TData
#undef TDataReader
#undef TDataWriter

#undef TTYPENAME
#undef TPlugin_new
#undef TPlugin_delete
```

4.24 Java Example

Secure WAN Transport Example Using Java.

Modules

- **HelloWorld.idl**
- **HelloWorld.java**
- **HelloWorldDataReader.java**
- **HelloWorldDataWriter.java**
- **HelloWorldPublisher.java**
- **HelloWorldSubscriber.java**
- **HelloWorldSeq.java**
- **HelloWorldTypeCode.java**
- **HelloWorldTypeSupport.java**
- **example_makefile**

4.24.1 Detailed Description

Secure WAN Transport Example Using Java.

4.25 HelloWorld.idl

[$(NDDSHOME)/example/JAVA/helloWorldWAN/HelloWorld.idl]

```
struct HelloWorld {
    string<128> msg;
};
```

4.26 HelloWorld.java

[$(NDDSHOME)/example/JAVA/helloWorldWAN/HelloWorld.java]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from .idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/


import com.rti.dds.infrastructure.*;
import com.rti.dds.infrastructure.Copyable;

import java.io.Serializable;
import com.rti.dds.cdr.CdrHelper;


public class HelloWorld implements Copyable, Serializable
{

    public String msg = ""; /* maximum length = (128) */


    public HelloWorld() {

    }


    public HelloWorld(HelloWorld other) {

        this();
        copy_from(other);
    }



    public static Object create() {
        return new HelloWorld();
    }

    public boolean equals(Object o) {

        if (o == null) {
            return false;
        }



        if(getClass() != o.getClass()) {
            return false;
        }
```

```
        HelloWorld otherObj = (HelloWorld)o;



        if(!msg.equals(otherObj.msg)) {
            return false;
        }

        return true;
    }

    public int hashCode() {
        int __result = 0;

        __result += msg.hashCode();

        return __result;
    }


    public Object copy_from(Object src) {


        HelloWorld typedSrc = (HelloWorld) src;
        HelloWorld typedDst = this;

        typedDst.msg = typedSrc.msg;

        return this;
    }



    public String toString(){
        return toString("", 0);
    }


    public String toString(String desc, int indent) {
        StringBuffer strBuffer = new StringBuffer();


        if (desc != null) {
            CdrHelper.printIndent(strBuffer, indent);
            strBuffer.append(desc).append(":\n");
        }


        CdrHelper.printIndent(strBuffer, indent+1);
        strBuffer.append("msg: ").append(msg).append("\n");

        return strBuffer.toString();
    }

}
```

4.27 HelloWorldDataReader.java

[$(NDDSHOME)/example/JAVA/helloWorldWAN/HelloWorldDataReader.java]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from .idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/

import com.rti.dds.infrastructure.InstanceHandle_t;
import com.rti.dds.subscription.DataReaderImpl;
import com.rti.dds.subscription.DataReaderListener;
import com.rti.dds.subscription.ReadCondition;
import com.rti.dds.subscription.SampleInfo;
import com.rti.dds.subscription.SampleInfoSeq;
import com.rti.dds.topic.TypeSupportImpl;


// ===========================================================================

public class HelloWorldDataReader extends DataReaderImpl {
    // -----------------------------------------------------------------------
    // Public Methods
    // -----------------------------------------------------------------------

    public void read(HelloWorldSeq received_data, SampleInfoSeq info_seq,
             int max_samples,
             int sample_states, int view_states, int instance_states) {
        read_untyped(received_data, info_seq, max_samples, sample_states,
             view_states, instance_states);
    }


    public void take(HelloWorldSeq received_data, SampleInfoSeq info_seq,
             int max_samples,
             int sample_states, int view_states, int instance_states) {
        take_untyped(received_data, info_seq, max_samples, sample_states,
             view_states, instance_states);
    }


    public void read_w_condition(HelloWorldSeq received_data,
                 SampleInfoSeq info_seq,
                 int max_samples,
                 ReadCondition condition) {
        read_w_condition_untyped(received_data, info_seq, max_samples,
                 condition);
    }


    public void take_w_condition(HelloWorldSeq received_data,
                 SampleInfoSeq info_seq,
```

```
            int max_samples,
            ReadCondition condition) {
    take_w_condition_untyped(received_data, info_seq, max_samples,
            condition);
}


public void read_next_sample(HelloWorld received_data, SampleInfo sample_info) {
    read_next_sample_untyped(received_data, sample_info);
}


public void take_next_sample(HelloWorld received_data, SampleInfo sample_info) {
    take_next_sample_untyped(received_data, sample_info);
}


public void read_instance(HelloWorldSeq received_data, SampleInfoSeq info_seq,
        int max_samples, InstanceHandle_t a_handle, int sample_states,
        int view_states, int instance_states) {

    read_instance_untyped(received_data, info_seq, max_samples, a_handle,
        sample_states, view_states, instance_states);
}


public void take_instance(HelloWorldSeq received_data, SampleInfoSeq info_seq,
        int max_samples, InstanceHandle_t a_handle, int sample_states,
        int view_states, int instance_states) {

    take_instance_untyped(received_data, info_seq, max_samples, a_handle,
        sample_states, view_states, instance_states);
}


  public void read_instance_w_condition(HelloWorldSeq received_data,
          SampleInfoSeq info_seq, int max_samples,
          InstanceHandle_t a_handle, ReadCondition condition) {

      read_instance_w_condition_untyped(received_data, info_seq,
          max_samples, a_handle, condition);
  }


  public void take_instance_w_condition(HelloWorldSeq received_data,
          SampleInfoSeq info_seq, int max_samples,
          InstanceHandle_t a_handle, ReadCondition condition) {

      take_instance_w_condition_untyped(received_data, info_seq,
          max_samples, a_handle, condition);
  }


public void read_next_instance(HelloWorldSeq received_data,
        SampleInfoSeq info_seq, int max_samples,
        InstanceHandle_t a_handle, int sample_states, int view_states,
        int instance_states) {
```

```
    read_next_instance_untyped(received_data, info_seq, max_samples,
        a_handle, sample_states, view_states, instance_states);
}


public void take_next_instance(HelloWorldSeq received_data,
        SampleInfoSeq info_seq, int max_samples,
        InstanceHandle_t a_handle, int sample_states, int view_states,
        int instance_states) {

    take_next_instance_untyped(received_data, info_seq, max_samples,
        a_handle, sample_states, view_states, instance_states);
}


public void read_next_instance_w_condition(HelloWorldSeq received_data,
        SampleInfoSeq info_seq, int max_samples,
        InstanceHandle_t a_handle, ReadCondition condition) {

    read_next_instance_w_condition_untyped(received_data, info_seq,
        max_samples, a_handle, condition);
}


public void take_next_instance_w_condition(HelloWorldSeq received_data,
        SampleInfoSeq info_seq, int max_samples,
        InstanceHandle_t a_handle, ReadCondition condition) {

    take_next_instance_w_condition_untyped(received_data, info_seq,
        max_samples, a_handle, condition);
}


public void return_loan(HelloWorldSeq received_data, SampleInfoSeq info_seq) {
    return_loan_untyped(received_data, info_seq);
}


public void get_key_value(HelloWorld key_holder, InstanceHandle_t handle){
    get_key_value_untyped(key_holder, handle);
}


public InstanceHandle_t lookup_instance(HelloWorld key_holder) {
    return lookup_instance_untyped(key_holder);
}

// ---------------------------------------------------------------------
// Package Methods
// ---------------------------------------------------------------------

// --- Constructors: -------------------------------------------------

/*package*/ HelloWorldDataReader(long native_reader, DataReaderListener listener,
                          int mask, TypeSupportImpl data_type) {
    super(native_reader, listener, mask, data_type);
```

```
    }
}
```

4.28 HelloWorldDataWriter.java

[$(NDDSHOME)/example/JAVA/helloWorldWAN/HelloWorldDataWriter.java]1

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from .idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/

import com.rti.dds.infrastructure.Time_t;
import com.rti.dds.infrastructure.WriteParams_t;
import com.rti.dds.infrastructure.InstanceHandle_t;
import com.rti.dds.publication.DataWriterImpl;
import com.rti.dds.publication.DataWriterListener;
import com.rti.dds.topic.TypeSupportImpl;


// ===========================================================================

public class HelloWorldDataWriter extends DataWriterImpl {
    // -----------------------------------------------------------------------
    // Public Methods
    // -----------------------------------------------------------------------

    public InstanceHandle_t register_instance(HelloWorld instance_data) {
        return register_instance_untyped(instance_data);
    }


    public InstanceHandle_t register_instance_w_timestamp(HelloWorld instance_data,
                                                Time_t source_timestamp) {
        return register_instance_w_timestamp_untyped(
            instance_data, source_timestamp);
    }


    public InstanceHandle_t register_instance_w_params(HelloWorld instance_data,
                                                WriteParams_t params) {
        return register_instance_w_params_untyped(
            instance_data, params);
    }


    public void unregister_instance(HelloWorld instance_data,
                                    InstanceHandle_t handle) {
        unregister_instance_untyped(instance_data, handle);
    }


    public void unregister_instance_w_timestamp(HelloWorld instance_data,
            InstanceHandle_t handle, Time_t source_timestamp) {
```

```
        unregister_instance_w_timestamp_untyped(
            instance_data, handle, source_timestamp);
    }


    public void unregister_instance_w_params(HelloWorld instance_data,
                                             WriteParams_t params) {

        unregister_instance_w_params_untyped(
            instance_data, params);
    }


    public void write(HelloWorld instance_data, InstanceHandle_t handle) {
        write_untyped(instance_data, handle);
    }


    public void write_w_timestamp(HelloWorld instance_data,
            InstanceHandle_t handle, Time_t source_timestamp) {

        write_w_timestamp_untyped(instance_data, handle, source_timestamp);
    }


    public void write_w_params(HelloWorld instance_data,
                               WriteParams_t params) {

        write_w_params_untyped(instance_data, params);
    }


    public void dispose(HelloWorld instance_data, InstanceHandle_t instance_handle){
        dispose_untyped(instance_data, instance_handle);
    }


    public void dispose_w_timestamp(HelloWorld instance_data,
            InstanceHandle_t instance_handle, Time_t source_timestamp) {

        dispose_w_timestamp_untyped(
            instance_data, instance_handle, source_timestamp);
    }


    public void dispose_w_params(HelloWorld instance_data,
                                 WriteParams_t params) {

        dispose_w_params_untyped(instance_data, params);
    }


    public void get_key_value(HelloWorld key_holder, InstanceHandle_t handle) {
        get_key_value_untyped(key_holder, handle);
    }
```

```
    public InstanceHandle_t lookup_instance(HelloWorld key_holder) {
        return lookup_instance_untyped(key_holder);
    }

    // --------------------------------------------------------------------
    // Package Methods
    // --------------------------------------------------------------------

    // --- Constructors: ------------------------------------------------

    /*package*/ HelloWorldDataWriter(long native_writer, DataWriterListener listener,
                                     int mask, TypeSupportImpl type) {
        super(native_writer, listener, mask, type);
    }
}
```

4.29 HelloWorldPublisher.java

[$(NDDSHOME)/example/JAVA/helloWorldWAN/HelloWorldPublisher.java]

```
/* HelloWorldPublisher.java

   A publication of data of type HelloWorld

   This file is derived from code automatically generated by the rtiddsgen
   command:

   rtiddsgen -language java -example <arch> HelloWorld.idl

   Example publication of type HelloWorld automatically generated by
   'rtiddsgen'. To test them follow these steps:

   (1) Compile this file and the example subscription.

   (2) Start the subscription on the same domain used for RTI Connext with the command
       java HelloWorldSubscriber <domain_id> <sample_count>

   (3) Start the publication on the same domain used for RTI Connext with the command
       java HelloWorldPublisher <domain_id> <sample_count>

   (4) [Optional] Specify the list of discovery initial peers and
       multicast receive addresses via an environment variable or a file
       (in the current working directory) called NDDS_DISCOVERY_PEERS.

   You can run any number of publishers and subscribers programs, and can
   add and remove them dynamically from the domain.

   Example:

       To run the example application on domain <domain_id>:

       Ensure that $(NDDSHOME)/lib/<arch> is on the dynamic library path for
       Java.

        On Unix:
             add $(NDDSHOME)/lib/<arch> to the 'LD_LIBRARY_PATH' environment
             variable

        On Windows:
                    add $(NDDSHOME)\lib\<arch> to the 'Path' environment variable


       Run the Java applications:

        java -Djava.ext.dirs=$(NDDSHOME)/class HelloWorldPublisher <domain_id>

        java -Djava.ext.dirs=$(NDDSHOME)/class HelloWorldSubscriber <domain_id>


modification history
```

```
------------ -------
*/

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Arrays;

import com.rti.dds.domain.*;
import com.rti.dds.infrastructure.*;
import com.rti.dds.publication.*;
import com.rti.dds.topic.*;
import com.rti.ndds.config.*;

// ===========================================================================

public class HelloWorldPublisher {
    // --------------------------------------------------------------------
    // Public Methods
    // --------------------------------------------------------------------

    public static void main(String[] args) {
        // --- Get domain ID --- //
        int domainId = 0;
        if (args.length >= 1) {
            domainId = Integer.valueOf(args[0]).intValue();
        }

        // -- Get max loop count; 0 means infinite loop --- //
        int sampleCount = 0;
        if (args.length >= 2) {
            sampleCount = Integer.valueOf(args[1]).intValue();
        }

        /* Uncomment this to turn on additional logging
        Logger.get_instance().set_verbosity_by_category(
            LogCategory.NDDS_CONFIG_LOG_CATEGORY_API,
            LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
        */

        // --- Run --- //
        publisherMain(domainId, sampleCount);
    }


    // --------------------------------------------------------------------
    // Private Methods
    // --------------------------------------------------------------------

    // --- Constructors: -------------------------------------------------

    private HelloWorldPublisher() {
        super();
    }


    // --------------------------------------------------------------------
```

```
    private static void publisherMain(int domainId, int sampleCount) {

        DomainParticipant participant = null;
        Publisher publisher = null;
        Topic topic = null;
        HelloWorldDataWriter writer = null;
        DomainParticipantQos participant_qos = new DomainParticipantQos();

        String WAN_SERVER = "127.0.0.1";
        String WAN_ID = "1";
        boolean USE_SECURITY = false;
        String WAN_LIB = null;

        String archName = System.getProperty("os.name").toLowerCase();
        if (archName.indexOf("win") >= 0) {
            WAN_LIB = "nddstransportwan.dll";
        } else {
            WAN_LIB = "libnddstransportwan.so";
        }

        try {
            // --- Create participant --- //

            /* Get default participant QoS from participant factory */
            DomainParticipantFactory.TheParticipantFactory.get_default_participant_qos(
                participant_qos);

            /* Disable builtin transports */
            participant_qos.transport_builtin.mask =
                TransportBuiltinKind.MASK_NONE;

            /* Set up property QoS to load plugin */
            PropertyQosPolicyHelper.add_property(participant_qos.property,
                "dds.transport.load_plugins", "dds.transport.wan_plugin.wan", false);

            /* library */
            PropertyQosPolicyHelper.add_property(participant_qos.property,
                "dds.transport.wan_plugin.wan.library",
                WAN_LIB, false);

            /* create function */
            PropertyQosPolicyHelper.add_property(participant_qos.property,
                "dds.transport.wan_plugin.wan.create_function", "NDDS_Transport_WAN_create", false);

            /* plugin properties */

            /* enable security */
            if (USE_SECURITY) {
                System.out.println("Enabling secure WAN transport");
                PropertyQosPolicyHelper.add_property(participant_qos.property,
                    "dds.transport.wan_plugin.wan.enable_security", "1", false);
                PropertyQosPolicyHelper.add_property(participant_qos.property,
                    "dds.transport.wan_plugin.wan.tls.verify.ca_file", "cacert.pem", false);
                PropertyQosPolicyHelper.add_property(participant_qos.property,
                    "dds.transport.wan_plugin.wan.tls.identity.certificate_chain_file", "peer1.pem", false);
            }
```

```
PropertyQosPolicyHelper.add_property(participant_qos.property,
    "dds.transport.wan_plugin.wan.server", WAN_SERVER, false);

PropertyQosPolicyHelper.add_property(participant_qos.property,
    "dds.transport.wan_plugin.wan.transport_instance_id", WAN_ID, false);

participant = DomainParticipantFactory.TheParticipantFactory.
    create_participant(
        domainId, participant_qos,
        null /* listener */, StatusKind.STATUS_MASK_NONE);
if (participant == null) {
    System.err.println("create_participant error\n");
    return;
}

// --- Create publisher --- //

/* To customize publisher QoS, use
   participant.get_default_publisher_qos() */
publisher = participant.create_publisher(
    DomainParticipant.PUBLISHER_QOS_DEFAULT, null /* listener */,
    StatusKind.STATUS_MASK_NONE);
if (publisher == null) {
    System.err.println("create_publisher error\n");
    return;
}


// --- Create topic --- //

/* Register type before creating topic */
String typeName = HelloWorldTypeSupport.get_type_name();
HelloWorldTypeSupport.register_type(participant, typeName);

/* To customize topic QoS, use
   participant.get_default_topic_qos() */
topic = participant.create_topic(
    "Example HelloWorld",
    typeName, DomainParticipant.TOPIC_QOS_DEFAULT,
    null /* listener */, StatusKind.STATUS_MASK_NONE);
if (topic == null) {
    System.err.println("create_topic error\n");
    return;
}

// --- Create writer --- //

/* To customize data writer QoS, use
   publisher.get_default_datawriter_qos() */
writer = (HelloWorldDataWriter)
    publisher.create_datawriter(
        topic, Publisher.DATAWRITER_QOS_DEFAULT,
        null /* listener */, StatusKind.STATUS_MASK_NONE);
if (writer == null) {
    System.err.println("create_datawriter error\n");
    return;
}
```

```
            // --- Write --- //

            /* Create data sample for writing */
            HelloWorld instance = new HelloWorld();

            InstanceHandle_t instance_handle = InstanceHandle_t.HANDLE_NIL;
            /* For data type that has key, if the same instance is going to be
               written multiple times, initialize the key here
               and register the keyed instance prior to writing */
            //instance_handle = writer.register_instance(instance);

            final long sendPeriodMillis = 4 * 1000; // 4 seconds

            for (int count = 0;
                 (sampleCount == 0) || (count < sampleCount);
                 ++count) {
                System.out.println("Writing HelloWorld to WAN, count " + count);

                /* Modify the instance to be written here */
                instance.msg = "Hello Wide Area World! (" + count + ")";

                /* Write data */
                writer.write(instance, instance_handle);
                try {
                    Thread.sleep(sendPeriodMillis);
                } catch (InterruptedException ix) {
                    System.err.println("INTERRUPTED");
                    break;
                }
            }

            //writer.unregister_instance(instance, instance_handle);

        } finally {

            // --- Shutdown --- //

            if(participant != null) {
                participant.delete_contained_entities();

                DomainParticipantFactory.TheParticipantFactory.
                    delete_participant(participant);
            }
            /* RTI Connext provides finalize_instance()
               method for people who want to release memory used by the
               participant factory singleton. Uncomment the following block of
               code for clean destruction of the participant factory
               singleton. */
            //DomainParticipantFactory.finalize_instance();
        }
    }
}
```

4.30 HelloWorldSubscriber.java

[$(NDDSHOME)/example/JAVA/helloWorldWAN/HelloWorldSubscriber.java]1

```
/* HelloWorldSubscriber.java

   A publication of data of type HelloWorld

   This file is derived from code automatically generated by the rtiddsgen
   command:

   rtiddsgen -language java -example <arch> HelloWorld.idl

   Example publication of type HelloWorld automatically generated by
   'rtiddsgen' To test them follow these steps:

   (1) Compile this file and the example subscription.

   (2) Start the subscription on the same domain used for RTI Connext with the command
       java HelloWorldSubscriber <domain_id> <sample_count>

   (3) Start the publication on the same domain used for RTI Connext with the command
       java HelloWorldPublisher <domain_id> <sample_count>

   (4) [Optional] Specify the list of discovery initial peers and
       multicast receive addresses via an environment variable or a file
       (in the current working directory) called NDDS_DISCOVERY_PEERS.

   You can run any number of publishers and subscribers programs, and can
   add and remove them dynamically from the domain.


   Example:

       To run the example application on domain <domain_id>:

       Ensure that $(NDDSHOME)/lib/<arch> is on the dynamic library path for
       Java.

        On Unix:
             add $(NDDSHOME)/lib/<arch> to the 'LD_LIBRARY_PATH' environment
             variable

        On Windows:
                    add $(NDDSHOME)\lib\<arch> to the 'Path' environment variable


       Run the Java applications:

        java -Djava.ext.dirs=$NDDSHOME/class HelloWorldPublisher <domain_id>

        java -Djava.ext.dirs=$NDDSHOME/class HelloWorldSubscriber <domain_id>


modification history
```

```
------------ -------
*/

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Arrays;

import com.rti.dds.domain.*;
import com.rti.dds.infrastructure.*;
import com.rti.dds.subscription.*;
import com.rti.dds.topic.*;
import com.rti.ndds.config.*;

// ===========================================================================

public class HelloWorldSubscriber {
    // -----------------------------------------------------------------------
    // Public Methods
    // -----------------------------------------------------------------------

    public static void main(String[] args) {
        // --- Get domain ID --- //
        int domainId = 0;
        if (args.length >= 1) {
            domainId = Integer.valueOf(args[0]).intValue();
        }

        // -- Get max loop count; 0 means infinite loop --- //
        int sampleCount = 0;
        if (args.length >= 2) {
            sampleCount = Integer.valueOf(args[1]).intValue();
        }


        /* Uncomment this to turn on additional logging
        Logger.get_instance().set_verbosity_by_category(
            LogCategory.NDDS_CONFIG_LOG_CATEGORY_API,
            LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
        */

        // --- Run --- //
        subscriberMain(domainId, sampleCount);
    }



    // -----------------------------------------------------------------------
    // Private Methods
    // -----------------------------------------------------------------------

    // --- Constructors: -----------------------------------------------------

    private HelloWorldSubscriber() {
        super();
    }
```

```
// -----------------------------------------------------------------------

private static void subscriberMain(int domainId, int sampleCount) {

    DomainParticipant participant = null;
    Subscriber subscriber = null;
    Topic topic = null;
    DataReaderListener listener = null;
    HelloWorldDataReader reader = null;
    DomainParticipantQos participant_qos = new DomainParticipantQos();

    String WAN_SERVER = "127.0.0.1";
    String WAN_ID = "2";
    boolean USE_SECURITY = false;
    String WAN_LIB = null;

    String archName = System.getProperty("os.name").toLowerCase();
    if (archName.indexOf("win") >= 0) {
        WAN_LIB = "nddstransportwan.dll";
    } else {
        WAN_LIB = "libnddstransportwan.so";
    }

    try {

        // --- Create participant --- //

        /* Get default participant QoS from participant factory */
        DomainParticipantFactory.TheParticipantFactory.get_default_participant_qos(
            participant_qos);

        /* Disable builtin transports */
        participant_qos.transport_builtin.mask =
            TransportBuiltinKind.MASK_NONE;

        /* Set up property QoS to load plugin */
        PropertyQosPolicyHelper.add_property(participant_qos.property,
            "dds.transport.load_plugins", "dds.transport.wan_plugin.wan", false);

        /* library */
        PropertyQosPolicyHelper.add_property(participant_qos.property,
            "dds.transport.wan_plugin.wan.library",
            WAN_LIB, false);

        /* create function */
        PropertyQosPolicyHelper.add_property(participant_qos.property,
            "dds.transport.wan_plugin.wan.create_function", "NDDS_Transport_WAN_create", false);

        /* plugin properties */

        /* enable security */
        if (USE_SECURITY) {
            System.out.println("Enabling secure WAN transport");
            PropertyQosPolicyHelper.add_property(participant_qos.property,
               "dds.transport.wan_plugin.wan.enable_security", "1", false);
            PropertyQosPolicyHelper.add_property(participant_qos.property,
               "dds.transport.wan_plugin.wan.tls.verify.ca_file", "cacert.pem", false);
```

```
    PropertyQosPolicyHelper.add_property(participant_qos.property,
       "dds.transport.wan_plugin.wan.tls.identity.certificate_chain_file", "peer2.pem", false);
}

PropertyQosPolicyHelper.add_property(participant_qos.property,
    "dds.transport.wan_plugin.wan.server", WAN_SERVER, false);

PropertyQosPolicyHelper.add_property(participant_qos.property,
    "dds.transport.wan_plugin.wan.transport_instance_id", WAN_ID, false);

participant = DomainParticipantFactory.TheParticipantFactory.
    create_participant(
        domainId, participant_qos,
        null /* listener */, StatusKind.STATUS_MASK_NONE);
if (participant == null) {
    System.err.println("create_participant error\n");
    return;
}

// --- Create subscriber --- //

/* To customize subscriber QoS, use
   participant.get_default_subscriber_qos() */
subscriber = participant.create_subscriber(
    DomainParticipant.SUBSCRIBER_QOS_DEFAULT, null /* listener */,
    StatusKind.STATUS_MASK_NONE);
if (subscriber == null) {
    System.err.println("create_subscriber error\n");
    return;
}

// --- Create topic --- //

/* Register type before creating topic */
String typeName = HelloWorldTypeSupport.get_type_name();
HelloWorldTypeSupport.register_type(participant, typeName);

/* To customize topic QoS, use
   participant.get_default_topic_qos() */
topic = participant.create_topic(
    "Example HelloWorld",
    typeName, DomainParticipant.TOPIC_QOS_DEFAULT,
    null /* listener */, StatusKind.STATUS_MASK_NONE);
if (topic == null) {
    System.err.println("create_topic error\n");
    return;
}

// --- Create reader --- //

listener = new HelloWorldListener();

/* To customize data reader QoS, use
   subscriber.get_default_datareader_qos() */
reader = (HelloWorldDataReader)
    subscriber.create_datareader(
        topic, Subscriber.DATAREADER_QOS_DEFAULT, listener,
```

```
                StatusKind.STATUS_MASK_ALL);
            if (reader == null) {
                System.err.println("create_datareader error\n");
                return;
            }

            // --- Wait for data --- //

            final long receivePeriodSec = 4;

            for (int count = 0;
                 (sampleCount == 0) || (count < sampleCount);
                 ++count) {
                System.out.println("HelloWorld subscriber sleeping for "
                                   + receivePeriodSec + " sec...");
                try {
                    Thread.sleep(receivePeriodSec * 1000);  // in millisec
                } catch (InterruptedException ix) {
                    System.err.println("INTERRUPTED");
                    break;
                }
            }
        } finally {

            // --- Shutdown --- //

            if(participant != null) {
                participant.delete_contained_entities();

                DomainParticipantFactory.TheParticipantFactory.
                    delete_participant(participant);
            }
            /* RTI Connext provides finalize_instance()
               method for people who want to release memory used by the
               participant factory singleton. Uncomment the following block of
               code for clean destruction of the participant factory
               singleton. */
            //DomainParticipantFactory.finalize_instance();
        }
    }

    // ----------------------------------------------------------------------
    // Private Types
    // ----------------------------------------------------------------------

    // ======================================================================

    private static class HelloWorldListener extends DataReaderAdapter {

        HelloWorldSeq _dataSeq = new HelloWorldSeq();
        SampleInfoSeq _infoSeq = new SampleInfoSeq();

        public void on_data_available(DataReader reader) {
            HelloWorldDataReader HelloWorldReader =
                (HelloWorldDataReader)reader;

            try {
```

```
                HelloWorldReader.take(
                    _dataSeq, _infoSeq,
                    ResourceLimitsQosPolicy.LENGTH_UNLIMITED,
                    SampleStateKind.ANY_SAMPLE_STATE,
                    ViewStateKind.ANY_VIEW_STATE,
                    InstanceStateKind.ANY_INSTANCE_STATE);

                for(int i = 0; i < _dataSeq.size(); ++i) {
                    SampleInfo info = (SampleInfo)_infoSeq.get(i);

                    if (info.valid_data) {
                        System.out.println(
                            ((HelloWorld)_dataSeq.get(i)).toString("Received",0));


                    }
                }
            } catch (RETCODE_NO_DATA noData) {
                // No data to process
            } finally {
                HelloWorldReader.return_loan(_dataSeq, _infoSeq);
            }
        }
    }
}
```

4.31 HelloWorldSeq.java

[$(NDDSHOME)/example/JAVA/helloWorldWAN/HelloWorldSeq.java]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from .idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/


import java.util.Collection;

import com.rti.dds.infrastructure.Copyable;
import com.rti.dds.util.Enum;
import com.rti.dds.util.Sequence;
import com.rti.dds.util.LoanableSequence;



public final class HelloWorldSeq extends LoanableSequence implements Copyable {
    // -----------------------------------------------------------------------
    // Package Fields
    // -----------------------------------------------------------------------

    /*package*/ transient Sequence _loanedInfoSequence = null;

    // -----------------------------------------------------------------------
    // Public Fields
    // -----------------------------------------------------------------------

    // --- Constructors: -----------------------------------------------------

    public HelloWorldSeq() {
        super(HelloWorld.class);
    }


    public HelloWorldSeq(int initialMaximum) {
        super(HelloWorld.class, initialMaximum);
    }


    public HelloWorldSeq(Collection elements) {
        super(HelloWorld.class, elements);
    }


    // --- From Copyable: ----------------------------------------------------

    public Object copy_from(Object src) {
        Sequence typedSrc = (Sequence) src;
        final int srcSize = typedSrc.size();
```

```
        final int origSize = size();

        // if this object's size is less than the source, ensure we have
        // enough room to store all of the objects
        if (getMaximum() < srcSize) {
            setMaximum(srcSize);
        }

        // trying to avoid clear() method here since it allocates memory
        // (an Iterator)
        // if the source object has fewer items than the current object,
        // remove from the end until the sizes are equal
        if (srcSize < origSize){
            removeRange(srcSize, origSize);
        }

        // copy the data from source into this (into positions that already
        // existed)
        for(int i = 0; (i < origSize) && (i < srcSize); i++){
            if (typedSrc.get(i) == null){
                set(i, null);
            } else {
                // check to see if our entry is null, if it is, a new instance has to be allocated
                if (get(i) == null){
                    set(i, HelloWorld.create());
                }
                set(i, ((Copyable) get(i)).copy_from(typedSrc.get(i)));
            }
        }

        // copy 'new' HelloWorld objects (beyond the original size of this object)
        for(int i = origSize; i < srcSize; i++){
            if (typedSrc.get(i) == null) {
                add(null);
            } else {
                // NOTE: we need to create a new object here to hold the copy
                add(HelloWorld.create());
                // we need to do a set here since enums aren't truely Copyable
                set(i, ((Copyable) get(i)).copy_from(typedSrc.get(i)));
            }
        }

        return this;
    }

}
```

4.32 HelloWorldTypeCode.java

[$(NDDSHOME)/example/JAVA/helloWorldWAN/HelloWorldTypeCode.java]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from .idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/

import com.rti.dds.typecode.*;


public class HelloWorldTypeCode {
    public static final TypeCode VALUE = getTypeCode();

    private static TypeCode getTypeCode() {
        TypeCode tc = null;
        int i=0;
        StructMember sm[] = new StructMember[1];

        sm[i]=new StructMember("msg",false,(short)-1,false,(TypeCode)new TypeCode(TCKind.TK_STRING,128)); i

        tc = TypeCodeFactory.TheTypeCodeFactory.create_struct_tc("HelloWorld",ExtensibilityKind.EXTENSIBLE_
        return tc;
    }
}
```

4.33 HelloWorldTypeSupport.java

[$(NDDSHOME)/example/JAVA/helloWorldWAN/HelloWorldTypeSupport.java]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from .idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Connext distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Connext manual.
*/


import com.rti.dds.cdr.CdrEncapsulation;
import com.rti.dds.cdr.CdrInputStream;
import com.rti.dds.cdr.CdrOutputStream;
import com.rti.dds.cdr.CdrPrimitiveType;
import com.rti.dds.cdr.CdrBuffer;
import com.rti.dds.cdr.CdrHelper;
import com.rti.dds.cdr.CdrMemberInfo;
import com.rti.dds.domain.DomainParticipant;
import com.rti.dds.publication.DataWriter;
import com.rti.dds.publication.DataWriterListener;
import com.rti.dds.subscription.DataReader;
import com.rti.dds.subscription.DataReaderListener;
import com.rti.dds.topic.KeyHash_t;
import com.rti.dds.topic.TypeSupportImpl;
import com.rti.dds.topic.TypeSupportType;
import com.rti.dds.util.Sequence;
import com.rti.dds.topic.DefaultEndpointData;
import com.rti.dds.infrastructure.RETCODE_ERROR;


import com.rti.dds.topic.TypeSupportParticipantInfo;
import com.rti.dds.topic.TypeSupportEndpointInfo;
import com.rti.dds.typecode.TypeCode;


import com.rti.dds.infrastructure.Copyable;


public class HelloWorldTypeSupport extends TypeSupportImpl {
    // ---------------------------------------------------------------------
    // Private Fields
    // ---------------------------------------------------------------------

    private static final String TYPE_NAME = "HelloWorld";

    private static final char[] PLUGIN_VERSION = {2, 0, 0, 0};


    public static final int LAST_MEMBER_ID = 0;


    private static final HelloWorldTypeSupport _singleton
```

```
    = new HelloWorldTypeSupport();

// ---------------------------------------------------------------------
// Public Methods
// ---------------------------------------------------------------------

// --- External methods: ----------------------------------------------
/* The methods in this section are for use by users of RTI Connext
 */

public static String get_type_name() {
    return _singleton.get_type_nameI();
}

public static void register_type(DomainParticipant participant,
                                 String type_name) {
    _singleton.register_typeI(participant, type_name);
}

public static void unregister_type(DomainParticipant participant,
                                   String type_name) {
    _singleton.unregister_typeI(participant, type_name);
}


 /* The methods in this section are for use by RTI Connext
 * itself and by the code generated by rtiddsgen for other types.
 * They should be used directly or modified only by advanced users and are
 * subject to change in future versions of RTI Connext.
 */
public static HelloWorldTypeSupport get_instance() {
    return _singleton;
}

public static HelloWorldTypeSupport getInstance() {
    return get_instance();
}

public Object create_data() {
    return new HelloWorld();
}

public void destroy_data(Object data) {
    return;
}

public Object create_key() {
    return new HelloWorld();
}

public void destroy_key(Object key) {
    return;
}

public Class get_type() {
    return HelloWorld.class;
}
```

```
    public Object copy_data(Object destination, Object source) {
        HelloWorld typedDst = (HelloWorld) destination;
        HelloWorld typedSrc = (HelloWorld) source;

        return typedDst.copy_from(typedSrc);

    }



    public long get_serialized_sample_max_size(Object endpoint_data,boolean include_encapsulation,short encapsulation_i
        long origAlignment = currentAlignment;

        long encapsulation_size = currentAlignment;


        if(include_encapsulation) {
          if (!CdrEncapsulation.isValidEncapsulationKind(encapsulation_id)) {
              throw new RETCODE_ERROR("Unsupported encapsulation");
          }

          encapsulation_size += CdrPrimitiveType.SHORT.getMaxSizeSerialized(encapsulation_size);
          encapsulation_size += CdrPrimitiveType.SHORT.getMaxSizeSerialized(encapsulation_size);
          encapsulation_size -= currentAlignment;
          currentAlignment = 0;
          origAlignment = 0;

        }



        currentAlignment +=  CdrPrimitiveType.getStringMaxSizeSerialized(currentAlignment, ((128)) + 1);

        if (include_encapsulation) {
            currentAlignment += encapsulation_size;
        }

        return currentAlignment - origAlignment;
    }


    public long get_serialized_sample_min_size(Object endpoint_data,boolean include_encapsulation,short encapsulation_i
        long origAlignment = currentAlignment;

        long encapsulation_size = currentAlignment;


        if(include_encapsulation) {
            if (!CdrEncapsulation.isValidEncapsulationKind(encapsulation_id)) {
                throw new RETCODE_ERROR("Unsupported encapsulation");
            }

            encapsulation_size += CdrPrimitiveType.SHORT.getMaxSizeSerialized(encapsulation_size);
            encapsulation_size += CdrPrimitiveType.SHORT.getMaxSizeSerialized(encapsulation_size);
            encapsulation_size -= currentAlignment;
            currentAlignment = 0;
```

```
            origAlignment = 0;

        }


        currentAlignment +=  CdrPrimitiveType.getStringMaxSizeSerialized(currentAlignment, 1);

        if (include_encapsulation) {
            currentAlignment += encapsulation_size;
        }

        return currentAlignment - origAlignment;
    }


    public long get_serialized_sample_size(
        Object endpoint_data, boolean include_encapsulation,
        short encapsulation_id, long currentAlignment,
        Object sample)
    {
        long origAlignment = currentAlignment;

        long encapsulation_size = currentAlignment;
        HelloWorld typedSrc = (HelloWorld) sample;

        if(include_encapsulation) {
            if (!CdrEncapsulation.isValidEncapsulationKind(encapsulation_id)) {
                throw new RETCODE_ERROR("Unsupported encapsulation");
            }


            encapsulation_size += CdrPrimitiveType.SHORT.getMaxSizeSerialized(encapsulation_size);
            encapsulation_size += CdrPrimitiveType.SHORT.getMaxSizeSerialized(encapsulation_size);
            encapsulation_size -= currentAlignment;
            currentAlignment = 0;
            origAlignment = 0;

        }


        currentAlignment += CdrPrimitiveType.getStringSerializedSize(currentAlignment, typedSrc.msg);

        if (include_encapsulation) {
            currentAlignment += encapsulation_size;
        }

        return currentAlignment - origAlignment;
    }


    public long get_serialized_key_max_size(
        Object endpoint_data,
        boolean include_encapsulation,
        short encapsulation_id,
        long currentAlignment)
    {
```

```
        long encapsulation_size = currentAlignment;

        long origAlignment = currentAlignment;


        if(include_encapsulation) {
            if (!CdrEncapsulation.isValidEncapsulationKind(encapsulation_id)) {
                throw new RETCODE_ERROR("Unsupported encapsulation");
            }


            encapsulation_size += CdrPrimitiveType.SHORT.getMaxSizeSerialized(encapsulation_size);
            encapsulation_size += CdrPrimitiveType.SHORT.getMaxSizeSerialized(encapsulation_size);
            encapsulation_size -= currentAlignment;
            currentAlignment = 0;
            origAlignment = 0;

        }


    currentAlignment += get_serialized_sample_max_size(
                            endpoint_data,false,encapsulation_id,currentAlignment);

        if (include_encapsulation) {
            currentAlignment += encapsulation_size;
        }

        return currentAlignment - origAlignment;
    }


    public void serialize(Object endpoint_data,Object src, CdrOutputStream dst,boolean serialize_encapsulation,
                          short encapsulation_id, boolean serialize_sample, Object endpoint_plugin_qos) {
        int position = 0;


        if(serialize_encapsulation) {

            dst.serializeAndSetCdrEncapsulation(encapsulation_id);;


            position = dst.resetAlignment();

        }


        if(serialize_sample) {
HelloWorld typedSrc = (HelloWorld) src;

        dst.writeString(typedSrc.msg,(128));

        }


        if (serialize_encapsulation) {
          dst.restoreAlignment(position);
        }
```

```
    }


    public void serialize_key(
        Object endpoint_data,
        Object src,
        CdrOutputStream dst,
        boolean serialize_encapsulation,
        short encapsulation_id,
        boolean serialize_key,
        Object endpoint_plugin_qos)
    {
        int position = 0;

        if (serialize_encapsulation) {

            dst.serializeAndSetCdrEncapsulation(encapsulation_id);


            position = dst.resetAlignment();

        }

        if (serialize_key) {
HelloWorld typedSrc = (HelloWorld) src;

            serialize(endpoint_data, src, dst, false, CdrEncapsulation.CDR_ENCAPSULATION_ID_CDR_BE, true, e

        }


        if (serialize_encapsulation) {
            dst.restoreAlignment(position);
        }

    }


    public Object deserialize_sample(
        Object endpoint_data,
        Object dst,
        CdrInputStream src, boolean deserialize_encapsulation,
        boolean deserialize_sample,
        Object endpoint_plugin_qos)
    {
        int position = 0;

        if(deserialize_encapsulation) {
            src.deserializeAndSetCdrEncapsulation();


            position = src.resetAlignment();

        }

        if(deserialize_sample) {
```

```
HelloWorld typedDst = (HelloWorld) dst;

    try {

            typedDst.msg = src.readString((128));

    } catch (Exception e) {
        if (src.available() > 0) {
            throw new RETCODE_ERROR("Error deserializing sample! Remainder: " + src.available() + "\n" +
                                                                "Exception caused by: " + e.getMessage());
        }
    }

        }


        if (deserialize_encapsulation) {
            src.restoreAlignment(position);
        }


        return dst;
    }



    public Object deserialize_key_sample(
        Object endpoint_data,
        Object dst,
        CdrInputStream src,
        boolean deserialize_encapsulation,
        boolean deserialize_key,
        Object endpoint_plugin_qos)
    {
        int position = 0;


        if(deserialize_encapsulation) {
            src.deserializeAndSetCdrEncapsulation();


            position = src.resetAlignment();

        }

        if(deserialize_key) {
HelloWorld typedDst = (HelloWorld) dst;


        deserialize_sample(endpoint_data, dst, src, false, true, endpoint_plugin_qos);

        }


        if (deserialize_encapsulation) {
            src.restoreAlignment(position);
        }
```

```
        return dst;
    }


    public void skip(Object endpoint_data,
                     CdrInputStream src,
                     boolean skip_encapsulation,
                     boolean skip_sample,
                     Object endpoint_plugin_qos)
    {
        int position = 0;


        if (skip_encapsulation) {
            src.skipEncapsulation();


            position = src.resetAlignment();

        }

        if (skip_sample) {

            src.skipString();

        }


        if (skip_encapsulation) {
            src.restoreAlignment(position);
        }

    }

    public Object serialized_sample_to_key(
        Object endpoint_data,
        Object sample,
        CdrInputStream src,
        boolean deserialize_encapsulation,
        boolean deserialize_key,
        Object endpoint_plugin_qos)
    {
        int position = 0;

        if(deserialize_encapsulation) {
            src.deserializeAndSetCdrEncapsulation();

            position = src.resetAlignment();

        }

        if (deserialize_key) {
HelloWorld typedDst = (HelloWorld) sample;
```

```
            deserialize_sample(
                endpoint_data, sample, src, false,
                true, endpoint_plugin_qos);

        }


        if (deserialize_encapsulation) {
            src.restoreAlignment(position);
        }


        return sample;
    }



    // --------------------------------------------------------------------
    // Callbacks
    // --------------------------------------------------------------------

    public Object on_participant_attached(Object registration_data,
                                          TypeSupportParticipantInfo participant_info,
                                          boolean top_level_registration,
                                          Object container_plugin_context,
                                          TypeCode type_code) {
        return super.on_participant_attached(
            registration_data, participant_info, top_level_registration,
            container_plugin_context, type_code);
    }

    public void on_participant_detached(Object participant_data) {
        super.on_participant_detached(participant_data);
    }

    public Object on_endpoint_attached(Object participantData,
                                       TypeSupportEndpointInfo endpoint_info,
                                       boolean top_level_registration,
                                       Object container_plugin_context) {
        return super.on_endpoint_attached(
              participantData,  endpoint_info,
              top_level_registration, container_plugin_context);
    }

    public void on_endpoint_detached(Object endpoint_data) {
        super.on_endpoint_detached(endpoint_data);
    }

    // --------------------------------------------------------------------
    // Protected Methods
    // --------------------------------------------------------------------

    protected DataWriter create_datawriter(long native_writer,
                                           DataWriterListener listener,
                                           int mask) {

        return new HelloWorldDataWriter(native_writer, listener, mask, this);
```

```
    }

    protected DataReader create_datareader(long native_reader,
                                           DataReaderListener listener,
                                           int mask) {

        return new HelloWorldDataReader(native_reader, listener, mask, this);

    }

    // --------------------------------------------------------------------
    // Constructor
    // --------------------------------------------------------------------

    protected HelloWorldTypeSupport() {

        /* If the user data type supports keys, then the second argument
        to the constructor below should be true.  Otherwise it should
        be false. */

        super(TYPE_NAME, false,HelloWorldTypeCode.VALUE,HelloWorld.class,TypeSupportType.TST_STRUCT, PLUGIN

    }

    protected HelloWorldTypeSupport(boolean enableKeySupport) {

        super(TYPE_NAME, enableKeySupport,HelloWorldTypeCode.VALUE,HelloWorld.class,TypeSupportType.TST_STR
    }
}
```

4.34 example_makefile

[$(NDDSHOME)/example/JAVA/helloWorldWAN/example_makefile]

```
#########################################################################
# makefile_HelloWorld_sparcSol2.10jdk
#
# (c) Copyright, Real-Time Innovations, 2012.  All rights reserved.
# No duplications, whole or partial, manual or electronic, may be made
# without express written permission.  Any such copies, or
# revisions thereof, must display this notice unaltered.
# This code contains trade secrets of Real-Time Innovations, Inc.
#
#
# This makefile was automatically generated by rtiddsgen.
#
# To compile, type:
#       gmake -f makefile_HelloWorld_sparcSol2.10jdk
#
# Note: This makefile is only meant to build our example applications and
#       may require alterations to build on your system.
#
# Make sure that javac and java are in your path.
#########################################################################

JAVA_PATH = java
JAVAC_PATH = javac

JAVA_SOURCES =  ./HelloWorld.java ./HelloWorldSeq.java ./HelloWorldTypeSupport.java ./HelloWorldTypeCode.java ./HelloWo

CLASS_FILES  = $(JAVA_SOURCES:%.java=%.class)

RTI_CLASSPATH := $(NDDSHOME)/class/nddsjava.jar

%.class : %.java
        $(JAVAC_PATH)  -classpath .:$(RTI_CLASSPATH) $<

all: $(CLASS_FILES)

#
# Convenient way to run the java programs
#

export LD_LIBRARY_PATH := $(NDDSHOME)/lib/sparcSol2.10jdk:/usr/lib/lwp:$(LD_LIBRARY_PATH)

HelloWorldPublisher: ./HelloWorldPublisher.class
        $(JAVA_PATH) -classpath ".:$(RTI_CLASSPATH)"  HelloWorldPublisher $(ARGS)

HelloWorldSubscriber: ./HelloWorldSubscriber.class
        $(JAVA_PATH) -classpath ".:$(RTI_CLASSPATH)"  HelloWorldSubscriber $(ARGS)
```

Chapter 5

Data Structure Documentation

5.1 NDDS_Transport_DTLS_Property_t Struct Reference

DTLS transport plugin property.

Data Fields

- struct **NDDS_Transport_Property_t parent**

  basic transport property

- RTI_INT32 **send_socket_buffer_size**

  Size in bytes of the send buffer of a socket used for sending.

- RTI_INT32 **recv_socket_buffer_size**

  Size in bytes of the receive buffer of a socket used for sending.

- RTI_INT32 **ignore_loopback_interface**

  Prevent the Transport Plugin from using the IP loopback interface.

- RTI_INT32 **ignore_nonrunning_interfaces**

  Prevent the Transport Plugin from using a network interface that is not reported as RUNNING by the operating system.

- RTI_UINT32 **transport_priority_mask**

Set mask for use of transport priority field.

- RTI_INT32 **transport_priority_mapping_low**

  Set low value of output range to IPv4 TOS.

- RTI_INT32 **transport_priority_mapping_high**

  Set high value of output range to IPv4 TOS.

- RTI_INT32 **recv_decode_buffer_size**

  Size of buffer for decoding packets from wire.

- RTI_INT32 **port_offset**

  Port offset to allow coexistence with non-secure UDP transport.

- RTI_UINT32 **dtls_handshake_resend_interval**

  DTLS handshake retransmission interval in milliseconds.

- RTI_UINT32 **dtls_connection_liveliness_interval**

  Liveliness interval (multiple of resend interval).

- struct **NDDS_Transport_TLS_OpenSSL_Configuration tls**

  OpenSSL TLS parameters.

5.1.1 Detailed Description

DTLS transport plugin property.

Should be initialized with **NDDS_Transport_DTLS_Plugin::NDDS_TRANSPORT_DTLS_PROPERTY_DEFAULT** (p. 28)

5.1.2 Field Documentation

5.1.2.1 struct NDDS_Transport_Property_t NDDS_Transport_DTLS_Property_t::parent [read]

basic transport property

[default] Refer to **NDDS_Transport_DTLS_Plugin::NDDS_TRANSPORT_DTLS_PROPERTY_DEFAULT** (p. 28)

5.1.2.2 RTI_INT32 NDDS_Transport_DTLS_Property_t::send_socket_buffer_size

Size in bytes of the send buffer of a socket used for sending.

See **NDDS_Transport_UDPv4_Property_t::send_socket_buffer_size** for more details

[**default**] **NDDS_Transport_DTLS_Plugin::NDDS_TRANSPORT_DTLS_MESSAGE_SIZE_MAX_DEFAULT** (p. 27)

5.1.2.3 RTI_INT32 NDDS_Transport_DTLS_Property_t::recv_socket_buffer_size

Size in bytes of the receive buffer of a socket used for sending.

See **NDDS_Transport_UDPv4_Property_t::recv_socket_buffer_size** for more details

[**default**] **NDDS_Transport_DTLS_Plugin::NDDS_TRANSPORT_DTLS_MESSAGE_SIZE_MAX_DEFAULT** (p. 27)

5.1.2.4 RTI_INT32 NDDS_Transport_DTLS_Property_t::ignore_loopback_interface

Prevent the Transport Plugin from using the IP loopback interface.

See **NDDS_Transport_UDPv4_Property_t::ignore_loopback_interface** for more details

[**default**] -1 i.e. Automatic RTI Connext policy

5.1.2.5 RTI_INT32 NDDS_Transport_DTLS_Property_t::ignore_nonrunning_interfaces

Prevent the Transport Plugin from using a network interface that is not reported as RUNNING by the operating system.

See **NDDS_Transport_UDPv4_Property_t::ignore_nonrunning_interfaces** for more details

[**default**] 0

5.1.2.6 RTI_UINT32 NDDS_Transport_DTLS_Property_t::transport_priority_mask

Set mask for use of transport priority field.

See **NDDS_Transport_UDPv4_Property_t::transport_priority_mask** for more details

[default] 0

5.1.2.7 RTI_INT32 NDDS_Transport_DTLS_Property_t::transport_priority_mapping_low

Set low value of output range to IPv4 TOS.

See **NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_low** for more details

[default] 0

5.1.2.8 RTI_INT32 NDDS_Transport_DTLS_Property_t::transport_priority_mapping_high

Set high value of output range to IPv4 TOS.

See **NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_high** for more details

[default] 0xff

5.1.2.9 RTI_INT32 NDDS_Transport_DTLS_Property_t::recv_decode_buffer_size

Size of buffer for decoding packets from wire.

An extra buffer is required for storage of encrypted data

[default] **NDDS_Transport_DTLS_Plugin::NDDS_TRANSPORT_DTLS_MESSAGE_SIZE_MAX_DEFAULT** (p. 27)

5.1.2.10 RTI_INT32 NDDS_Transport_DTLS_Property_t::port_offset

Port offset to allow coexistence with non-secure UDP transport.

[default] 144

5.1.2.11 RTI_UINT32 NDDS_Transport_DTLS_Property_t::dtls_handshake_resend_interval

DTLS handshake retransmission interval in milliseconds.

[**default**] 1000 (1 sec)

5.1.2.12 RTI_UINT32 NDDS_Transport_DTLS_Property_t::dtls_connection_liveliness_interval

Liveliness interval (multiple of resend interval).

The connection will be dropped if no message from the peer is received in this amount of time. This enables cleaning up state for peers that are no longer responding. A secure keep-alive message will be sent every half-interval if no other sends have occurred for a given DTLS connection during that time.

[**default**] 60

5.1.2.13 struct NDDS_Transport_TLS_OpenSSL_Configuration NDDS_Transport_DTLS_Property_t::tls [read]

OpenSSL TLS parameters.

[**default**] **NDDS_Transport_TLS_Plugin::NDDS_TRANSPORT_TLS_OPENSSL_CONFIGURATION_DEFAULT** (p. 33)

5.2 NDDS_Transport_TLS_Ciphers Struct Reference

Set of TLS properties for cipher configuration.

Data Fields

- char * **cipher_list**

  List of available (D)TLS ciphers.

- **DDS_Long dh_param_files_length**

  Number of DH key files supplied.

- struct **NDDS_Transport_TLS_DHParamFile** * **dh_param_files**

  List of available DH key files.

- char * **engine_id**

  ID of OpenSSL cipher engine to request.

5.2.1 Detailed Description

Set of TLS properties for cipher configuration.

5.2.2 Field Documentation

5.2.2.1 char* NDDS_Transport_TLS_Ciphers::cipher_list

List of available (D)TLS ciphers.

See the OpenSSL manual page for SSL_set_cipher_list for more information on the format of this string

[default] NULL

5.2.2.2 DDS_Long NDDS_Transport_TLS_Ciphers::dh_param_files_length

Number of DH key files supplied.

[default] 0

5.2.2.3 struct NDDS_Transport_TLS_DHParamFile* NDDS_Transport_TLS_Ciphers::dh_param_files [read]

List of available DH key files.

[**default**] NULL

5.2.2.4 char* NDDS_Transport_TLS_Ciphers::engine_id

ID of OpenSSL cipher engine to request.

[**default**] NULL

5.3 NDDS_Transport_TLS_DHParamFile Struct Reference

Name of a Diffie-Helman (DH) key file and the length of the contained key in bits.

Data Fields

- char * **file**

  Name of DH key file.

- **DDS_Long bits**

  Length of DH key in bits.

5.3.1 Detailed Description

Name of a Diffie-Helman (DH) key file and the length of the contained key in bits.

5.3.2 Field Documentation

5.3.2.1 char* NDDS_Transport_TLS_DHParamFile::file

Name of DH key file.

[default] 0

5.3.2.2 DDS_Long NDDS_Transport_TLS_DHParamFile::bits

Length of DH key in bits.

[default] NULL

5.4 NDDS_Transport_TLS_Identity Struct Reference

Set of TLS properties for identity.

Data Fields

- char * **certificate_chain_file**

  File containing identifying certificate chain (in PEM format).

- char * **private_key_password**

  Password for private key.

- char * **private_key_file**

  File containing private key (in PEM format).

- char * **rsa_private_key_file**

  File containing RSA private key (in PEM format).

5.4.1 Detailed Description

Set of TLS properties for identity.

5.4.2 Field Documentation

5.4.2.1 char* NDDS_Transport_TLS_Identity::certificate_chain_file

File containing identifying certificate chain (in PEM format).

An identifying certificate is required for secure communication. The file must be sorted starting with the certificate to the highest level (root CA). If no private key is specified, this file will be used to load a non-RSA private key.

[**default**] NULL

5.4.2.2 char* NDDS_Transport_TLS_Identity::private_key_password

Password for private key.

[**default**] NULL (no password)

5.4.2.3 char* NDDS_Transport_TLS_Identity::private_key_file

File containing private key (in PEM format).

If no private key is specified (all values are NULL), this value will default to the same file as the specified certificate chain file.

[default] NULL

5.4.2.4 char* NDDS_Transport_TLS_Identity::rsa_private_key_file

File containing RSA private key (in PEM format).

[default] NULL

5.5 NDDS_Transport_TLS_OpenSSL_-Configuration Struct Reference

Full set of TLS properties.

Data Fields

- struct **NDDS_Transport_TLS_Verification verify**

  Verification properties.

- struct **NDDS_Transport_TLS_Identity identity**

  Identity properties.

- struct **NDDS_Transport_TLS_Ciphers cipher**

  Cipher properties.

5.5.1 Detailed Description

Full set of TLS properties.

Should be initialized with **NDDS_Transport_TLS_Plugin::NDDS_-TRANSPORT_TLS_OPENSSL_CONFIGURATION_DEFAULT** (p. 33)

5.5.2 Field Documentation

5.5.2.1 struct NDDS_Transport_TLS_Verification NDDS_Transport_TLS_OpenSSL_Configuration::verify [read]

Verification properties.

5.5.2.2 struct NDDS_Transport_TLS_Identity NDDS_-Transport_TLS_OpenSSL_Configuration::identity [read]

Identity properties.

5.5.2.3 struct NDDS_Transport_TLS_Ciphers NDDS_Transport_TLS_OpenSSL_Configuration::cipher [read]

Cipher properties.

5.6 NDDS_Transport_TLS_Verification Struct Reference

Set of TLS properties for certificate authorities (CAs) and verification.

Data Fields

- char * **ca_file**

  Name of file containing Certificate Authority certificates.

- char * **ca_path**

  Paths to directories containing Certificate Authority certificates.

- **DDS_Long verify_depth**

  Maximum certificate chain length for verification.

- **DDS_Long verify_peer**

  If non-zero, use mutual authentication when performing TLS handshake; if zero, only client will verify server certificate.

- **NDDS_Transport_TLS_Verify_Callback callback**

  Callback used to verify peer certificates.

5.6.1 Detailed Description

Set of TLS properties for certificate authorities (CAs) and verification.

5.6.2 Field Documentation

5.6.2.1 char* NDDS_Transport_TLS_Verification::ca_file

Name of file containing Certificate Authority certificates.

File should be in PEM format. See the OpenSSL manual page for SSL_load_verify_locations for more information.

At least one of ca_file and ca_path must be specified; both may be specified.

[default] NULL

5.6.2.2 char* NDDS_Transport_TLS_Verification::ca_path

Paths to directories containing Certificate Authority certificates.

Files should be in PEM format, and follow the OpenSSL-required naming conventions. See the OpenSSL manual page for SSL_CTX_load_verify_locations for more information.

At least one of ca_file and ca_path must be specified; both may be specified.

[default] NULL

5.6.2.3 DDS_Long NDDS_Transport_TLS_Verification::verify_depth

Maximum certificate chain length for verification.

[default] -1 (no limit)

5.6.2.4 DDS_Long NDDS_Transport_TLS_Verification::verify_peer

If non-zero, use mutual authentication when performing TLS handshake; if zero, only client will verify server certificate.

[default] 0 (non-mutual verify)

5.6.2.5 NDDS_Transport_TLS_Verify_Callback NDDS_Transport_TLS_Verification::callback

Callback used to verify peer certificates.

See the OpenSSL manual page for SSL_set_verify for more information. There are a number of default callbacks included in the Secure Transport. See **NDDS_Transport_TLS_default_verify_callback()** (p. 34) , **NDDS_Transport_TLS_verbose_verify_callback()** (p. 34) .

[default] NULL (use **NDDS_Transport_TLS_default_verify_callback()** (p. 34))

5.7 NDDS_Transport_WAN_Property_t Struct Reference

WAN transport plugin property.

Data Fields

- struct **NDDS_Transport_UDPv4_Property_t parent**
  UDPv4 transport plugin property.
- RTI_INT32 **enable_security**
  Set to 1 to enable secure transport using DTLS.
- RTI_INT32 **recv_decode_buffer_size**
  Size of buffer for decoding packets from wire when security-enabled.
- RTI_INT32 **port_offset**
  Port offset to allow coexistence with non-WAN UDP transport.
- RTI_UINT32 **dtls_handshake_resend_interval**
  DTLS handshake retransmission interval in milliseconds.
- struct **NDDS_Transport_TLS_OpenSSL_Configuration tls**
  OpenSSL TLS parameters.
- unsigned char **transport_instance_id** [NDDS_TRANSPORT_WAN_TRANSPORT_INSTANCE_ID_LENGTH]
  The WAN ID.
- char * **interface_address**
  Interface IP address for the transport sockets.
- char * **server**
  WAN Rendezvous Server address.
- NDDS_Transport_Port_t **server_port**
  WAN Rendezvous Server port.
- RTI_UINT32 **stun_retransmission_interval**
  STUN Retransmission interval in millisecs.
- RTI_UINT32 **stun_number_of_retransmissions**

STUN Retransmission count.

ˆ RTI_UINT32 **stun_liveliness_period**

STUN Liveliness period in millisecs.

5.7.1 Detailed Description

WAN transport plugin property.

Should be initialized with **NDDS_Transport_WAN_Plugin::NDDS_TRANSPORT_WAN_PROPERTY_DEFAULT** (p. 17)

5.7.2 Field Documentation

5.7.2.1 struct NDDS_Transport_UDPv4_Property_t NDDS_Transport_WAN_Property_t::parent [read]

UDPv4 transport plugin property.

[default] Refer to **NDDS_Transport_WAN_Plugin::NDDS_TRANSPORT_WAN_PROPERTY_DEFAULT** (p. 17)

5.7.2.2 RTI_INT32 NDDS_Transport_WAN_Property_t::enable_security

Set to 1 to enable secure transport using DTLS.

If enabled, there must be valid OpenSSL properties in **NDDS_Transport_WAN_Property_t::tls** (p. 185)

[default] 0 (off)

5.7.2.3 RTI_INT32 NDDS_Transport_WAN_Property_t::recv_decode_buffer_size

Size of buffer for decoding packets from wire when security-enabled.

An extra buffer is required for storage of encrypted data

[default] **NDDS_Transport_WAN_Plugin::NDDS_TRANSPORT_WAN_DTLS_FRAGMENT_SIZE_MAX_DEFAULT** (p. 16)

5.7.2.4 RTI_INT32 NDDS_Transport_WAN_Property_t::port_offset

Port offset to allow coexistence with non-WAN UDP transport.

[**default**] 144

5.7.2.5 RTI_UINT32 NDDS_Transport_WAN_Property_t::dtls_handshake_resend_interval

DTLS handshake retransmission interval in milliseconds.

[**default**] 1000 (1 sec)

5.7.2.6 struct NDDS_Transport_TLS_OpenSSL_Configuration NDDS_Transport_WAN_Property_t::tls [read]

OpenSSL TLS parameters.

[**default**] **NDDS_Transport_TLS_Plugin::NDDS_TRANSPORT_TLS_OPENSSL_CONFIGURATION_DEFAULT** (p. 33)

5.7.2.7 unsigned char NDDS_Transport_WAN_Property_t::transport_instance_id[NDDS_TRANSPORT_WAN_TRANSPORT_INSTANCE_ID_LENGTH]

The WAN ID.

This value must be unique for all transport instances communicating with the same WAN Rendezvous Server.

[**default**] {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}

[**range**] {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0} to {0xfe,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff}

5.7.2.8 char* NDDS_Transport_WAN_Property_t::interface_address

Interface IP address for the transport sockets.

The WAN transport can be configured to bind all sockets to a specified interface. This is required for consistent WAN communication on multi-homed systems.

If the value is NULL, the sockets will be bound to the special IP address INADDR_ANY. This address allows the sockets to receive packets destined to any of the interfaces.

[**default**] NULL

5.7.2.9 char* NDDS_Transport_WAN_Property_t::server

WAN Rendezvous Server address.

[default] NULL

5.7.2.10 NDDS_Transport_Port_t NDDS_Transport_WAN_Property_t::server_port

WAN Rendezvous Server port.

[default] 3478

5.7.2.11 RTI_UINT32 NDDS_Transport_WAN_Property_t::stun_retransmission_interval

STUN Retransmission interval in millisecs.

STUN request messages requiring a response are re-sent with this interval. The interval is doubled after each retransmission.

[default] 100

5.7.2.12 RTI_UINT32 NDDS_Transport_WAN_Property_t::stun_number_of_retransmissions

STUN Retransmission count.

STUN messages are resent up to this number of times until a response is received

[default] 7

5.7.2.13 RTI_UINT32 NDDS_Transport_WAN_Property_t::stun_liveliness_period

STUN Liveliness period in millisecs.

This is the period at which messages are sent to peers to keep NAT holes open; and to the WAN server to refresh bound ports

[default] 15000