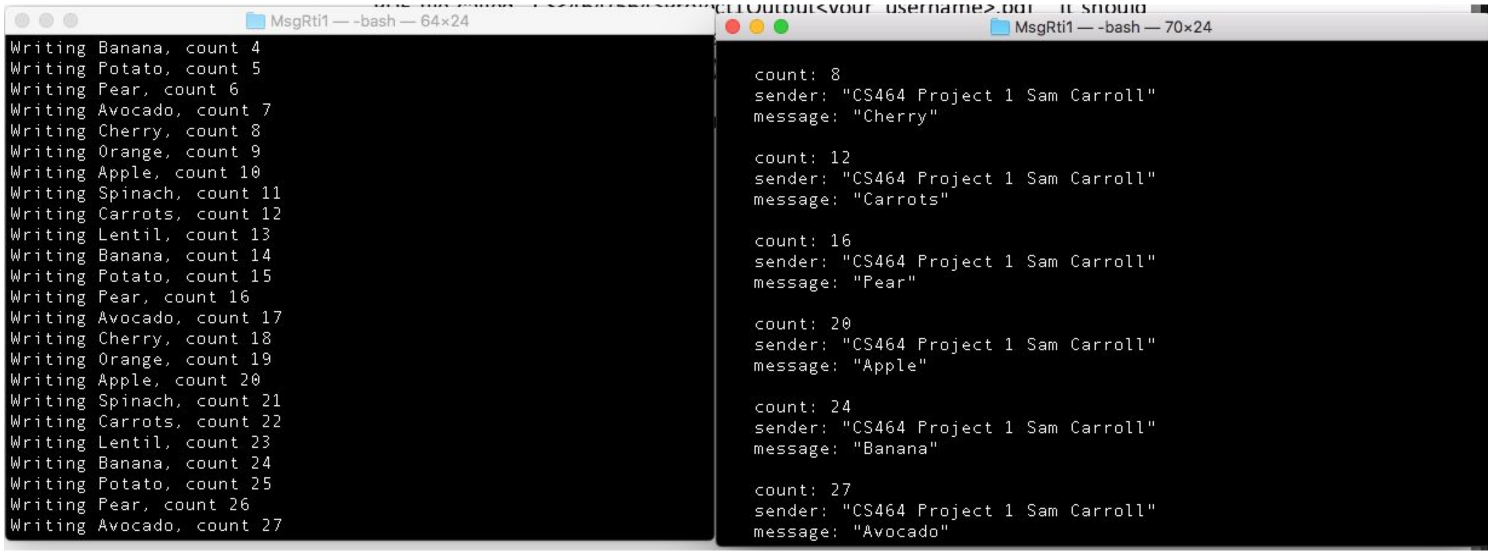
Samuel Carroll
11477450
CptS 464
Project 1

Running Screenshot:

Publisher **Subscriber**



```
Writing Banana, count 4
Writing Potato, count 5
Writing Pear, count 6
Writing Avocado, count 7
Writing Cherry, count 8
Writing Orange, count 9
Writing Apple, count 10
Writing Spinach, count 11
Writing Carrots, count 12
Writing Lentil, count 13
Writing Banana, count 14
Writing Potato, count 15
Writing Pear, count 16
Writing Avocado, count 17
Writing Cherry, count 18
Writing Orange, count 19
Writing Apple, count 20
Writing Spinach, count 21
Writing Carrots, count 22
Writing Lentil, count 23
Writing Banana, count 24
Writing Potato, count 25
Writing Pear, count 26
Writing Avocado, count 27
```

```
   count: 8
   sender: "CS464 Project 1 Sam Carroll"
   message: "Cherry"

   count: 12
   sender: "CS464 Project 1 Sam Carroll"
   message: "Carrots"

   count: 16
   sender: "CS464 Project 1 Sam Carroll"
   message: "Pear"

   count: 20
   sender: "CS464 Project 1 Sam Carroll"
   message: "Apple"

   count: 24
   sender: "CS464 Project 1 Sam Carroll"
   message: "Banana"

   count: 27
   sender: "CS464 Project 1 Sam Carroll"
   message: "Avocado"
```

Msg_Subscriber.cxx file:

```
/* Msg_subscriber.cxx

A subscription example

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

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

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

(1) Compile this file and the example publication.

(2) Start the subscription with the command
objs/<arch>/Msg_subscriber <domain_id> <sample_count>
```

```
(3) Start the publication with the command
objs/<arch>/Msg_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>/Msg_publisher <domain_id>
objs/<arch>/Msg_subscriber <domain_id>

On Windows:

objs\<arch>\Msg_publisher <domain_id>
objs\<arch>\Msg_subscriber <domain_id>

*/

#include <stdio.h>
#include <stdlib.h>

#include "Msg.h"
#include "MsgSupport.h"
#include "ndds/ndds_cpp.h"

class MsgListener : 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);
};

Msg* hold;

void MsgListener::on_data_available(DDSDataReader* reader)
{
   MsgDataReader *Msg_reader = NULL;
   MsgSeq data_seq;
   DDS_SampleInfoSeq info_seq;
   DDS_ReturnCode_t retcode;
   int i;

   Msg_reader = MsgDataReader::narrow(reader);
   if (Msg_reader == NULL) {
       fprintf(stderr, "DataReader narrow error\n");
       return;
   }

   retcode = Msg_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) {
       fprintf(stderr, "take error %d\n", retcode);
       return;
   }

   for (i = 0; i < data_seq.length(); ++i) {
       if (info_seq[i].valid_data) {
           //printf("Received data\n");
           hold = &data_seq[i];
           //MsgTypeSupport::print_data(&data_seq[i]);
       }
   }
```

```
   retcode = Msg_reader->return_loan(data_seq, info_seq);
   if (retcode != DDS_RETCODE_OK) {
       fprintf(stderr, "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) {
           fprintf(stderr, "delete_contained_entities error %d\n", retcode);
           status = -1;
       }

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

   /* RTI Connext provides the finalize_instance() method on
   domain participant factory for people who want to release memory used
   by the participant factory. Uncomment the following block of code for
   clean destruction of the singleton. */
   /*

   retcode = DDSDomainParticipantFactory::finalize_instance();
   if (retcode != DDS_RETCODE_OK) {
       fprintf(stderr, "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;
   MsgListener *reader_listener = NULL;
   DDSDataReader *reader = NULL;
```

```
DDS_ReturnCode_t retcode;
const char *type_name = NULL;
int count = 0;
DDS_Duration_t receive_period = {1,0};
int status = 0;

/* To customize the participant QoS, use
the configuration file USER_QOS_PROFILES.xml */
participant = DDSTheParticipantFactory->create_participant(
    domainId, DDS_PARTICIPANT_QOS_DEFAULT,
    NULL /* listener */, DDS_STATUS_MASK_NONE);
if (participant == NULL) {
    fprintf(stderr, "create_participant error\n");
    subscriber_shutdown(participant);
    return -1;
}

/* To customize the subscriber QoS, use
the configuration file USER_QOS_PROFILES.xml */
subscriber = participant->create_subscriber(
    DDS_SUBSCRIBER_QOS_DEFAULT, NULL /* listener */, DDS_STATUS_MASK_NONE);
if (subscriber == NULL) {
    fprintf(stderr, "create_subscriber error\n");
    subscriber_shutdown(participant);
    return -1;
}

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

/* To customize the topic QoS, use
the configuration file USER_QOS_PROFILES.xml */
topic = participant->create_topic(
    "Example Msg",
    type_name, DDS_TOPIC_QOS_DEFAULT, NULL /* listener */,
    DDS_STATUS_MASK_NONE);
if (topic == NULL) {
    fprintf(stderr, "create_topic error\n");
    subscriber_shutdown(participant);
    return -1;
}
```

```
    /* Create a data reader listener */
    reader_listener = new MsgListener();

    /* To customize the data reader QoS, use
    the configuration file USER_QOS_PROFILES.xml */
    reader = subscriber->create_datareader(
        topic, DDS_DATAREADER_QOS_DEFAULT, reader_listener,
        DDS_STATUS_MASK_ALL);
    if (reader == NULL) {
        fprintf(stderr, "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("Msg subscriber sleeping for %d sec...\n",
        //receive_period.sec);
        if(hold == NULL)
        {
            printf("Msg subscriber sleeping for %d sec...\n");
        }
        else
        {
            MsgTypeSupport::print_data(hold);
        }

        NDDSUtility::sleep(receive_period);
    }

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

    return status;
}

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);
}
```

Msg_Publisher.cxx

```
/* Msg_publisher.cxx

A publication of data of type Msg

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

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

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

(1) Compile this file and the example subscription.

(2) Start the subscription with the command
objs/<arch>/Msg_subscriber <domain_id> <sample_count>

(3) Start the publication with the command
objs/<arch>/Msg_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>/Msg_publisher <domain_id> o
objs/<arch>/Msg_subscriber <domain_id>

On Windows:

objs\<arch>\Msg_publisher <domain_id>
objs\<arch>\Msg_subscriber <domain_id>

*/

#include <stdio.h>
#include <stdlib.h>

#include "Msg.h"
#include "MsgSupport.h"
#include "ndds/ndds_cpp.h"

/* 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) {
           fprintf(stderr, "delete_contained_entities error %d\n", retcode);
           status = -1;
       }

       retcode = DDSTheParticipantFactory->delete_participant(participant);
       if (retcode != DDS_RETCODE_OK) {
           fprintf(stderr, "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. Uncomment the following block of code for
   clean destruction of the singleton. */
   /*

   retcode = DDSDomainParticipantFactory::finalize_instance();
   if (retcode != DDS_RETCODE_OK) {
```

```
            fprintf(stderr, "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;
        MsgDataWriter * Msg_writer = NULL;
        Msg *instance = NULL;
        DDS_ReturnCode_t retcode;
        DDS_InstanceHandle_t instance_handle = DDS_HANDLE_NIL;
        const char *type_name = NULL;
        int count = 0;
        DDS_Duration_t send_period = {0,250000000};
        std::string fruits[10] = {"Apple", "Spinach", "Carrots", "Lentil", "Banana",
"Potato", "Pear", "Avocado", "Cherry", "Orange"};
        /* To customize participant QoS, use
        the configuration file USER_QOS_PROFILES.xml */
        participant = DDSTheParticipantFactory->create_participant(
            domainId, DDS_PARTICIPANT_QOS_DEFAULT,
            NULL /* listener */, DDS_STATUS_MASK_NONE);
        if (participant == NULL) {
            fprintf(stderr, "create_participant error\n");
            publisher_shutdown(participant);
            return -1;
        }

        /* To customize publisher QoS, use
        the configuration file USER_QOS_PROFILES.xml */
        publisher = participant->create_publisher(
            DDS_PUBLISHER_QOS_DEFAULT, NULL /* listener */, DDS_STATUS_MASK_NONE);
        if (publisher == NULL) {
            fprintf(stderr, "create_publisher error\n");
            publisher_shutdown(participant);
            return -1;
        }

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

    /* To customize topic QoS, use
    the configuration file USER_QOS_PROFILES.xml */
    topic = participant->create_topic(
        "Example Msg",
        type_name, DDS_TOPIC_QOS_DEFAULT, NULL /* listener */,
        DDS_STATUS_MASK_NONE);
    if (topic == NULL) {
        fprintf(stderr, "create_topic error\n");
        publisher_shutdown(participant);
        return -1;
    }

    /* To customize data writer QoS, use
    the configuration file USER_QOS_PROFILES.xml */
    writer = publisher->create_datawriter(
        topic, DDS_DATAWRITER_QOS_DEFAULT, NULL /* listener */,
        DDS_STATUS_MASK_NONE);
    if (writer == NULL) {
        fprintf(stderr, "create_datawriter error\n");
        publisher_shutdown(participant);
        return -1;
    }
    Msg_writer = MsgDataWriter::narrow(writer);
    if (Msg_writer == NULL) {
        fprintf(stderr, "DataWriter narrow error\n");
        publisher_shutdown(participant);
        return -1;
    }

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

    /* For a data type that has a 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 = Msg_writer->register_instance(*instance);
    */

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

        printf("Writing %s, count %d\n", fruits[count % 10].c_str(), count);

        /* Modify the data to be sent here */
        instance->sender = "CS464 Project 1 Sam Carroll";
        instance->message = (char*)fruits[count%10].c_str();
        instance->count = count;
        retcode = Msg_writer->write(*instance, instance_handle);
        if (retcode != DDS_RETCODE_OK) {
            fprintf(stderr, "write error %d\n", retcode);
        }

        NDDSUtility::sleep(send_period);
    }

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

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

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

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);
}
```

Msg.idl

```
const long MSG_LEN=256;

struct Msg{
       long count;
       string<MSG_LEN> sender;
       string<MSG_LEN> message;
};
```

Subscriber Output File:

```
Msg subscriber sleeping for 0 sec...
Msg subscriber sleeping for 0 sec...
Msg subscriber sleeping for 0 sec...
Msg subscriber sleeping for 0 sec...

   count: 4
   sender: "CS464 Project 1 Sam Carroll"
   message: "Banana"

   count: 8
   sender: "CS464 Project 1 Sam Carroll"
   message: "Cherry"

   count: 12
   sender: "CS464 Project 1 Sam Carroll"
   message: "Carrots"

   count: 16
   sender: "CS464 Project 1 Sam Carroll"
   message: "Pear"

   count: 20
   sender: "CS464 Project 1 Sam Carroll"
   message: "Apple"

   count: 23
   sender: "CS464 Project 1 Sam Carroll"
   message: "Lentil"
```

Publisher Output File

```
Writing Apple, count 0
Writing Spinach, count 1
Writing Carrots, count 2
Writing Lentil, count 3
Writing Banana, count 4
Writing Potato, count 5
Writing Pear, count 6
Writing Avocado, count 7
Writing Cherry, count 8
Writing Orange, count 9
Writing Apple, count 10
Writing Spinach, count 11
Writing Carrots, count 12
Writing Lentil, count 13
Writing Banana, count 14
Writing Potato, count 15
Writing Pear, count 16
Writing Avocado, count 17
Writing Cherry, count 18
Writing Orange, count 19
Writing Apple, count 20
Writing Spinach, count 21
Writing Carrots, count 22
Writing Lentil, count 23
```