```
/* little_publisher.cxx

   A publication of data of type LittleMsg

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

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

Example publication of type LittleMsg 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>/little_subscriber <domain_id> <sample_count>

(3) Start the publication with the command
objs/<arch>/little_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>/little_publisher <domain_id> o
objs/<arch>/little_subscriber <domain_id>

On Windows:

objs\<arch>\little_publisher <domain_id>
objs\<arch>\little_subscriber <domain_id>


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

#include <stdio.h>
#include <stdlib.h>
#ifdef RTI_VX653
#include <vThreadsData.h>
#endif
#include "little.h"
#include "littleSupport.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) {
                        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. 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;
        LittleMsgDataWriter * LittleMsg_writer = NULL;
        LittleMsg *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 = {4,0};

        /* 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) {
                printf("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) {
                printf("create_publisher error\n");
                publisher_shutdown(participant);
                return -1;
        }

        /* Register type before creating topic */
        type_name = LittleMsgTypeSupport::get_type_name();
        retcode = LittleMsgTypeSupport::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
           the configuration file USER_QOS_PROFILES.xml */
        topic = participant->create_topic(
                        "Example LittleMsg",
                        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
           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) {
                printf("create_datawriter error\n");
                publisher_shutdown(participant);
                return -1;
        }
        LittleMsg_writer = LittleMsgDataWriter::narrow(writer);
        if (LittleMsg_writer == NULL) {
                printf("DataWriter narrow error\n");
                publisher_shutdown(participant);
                return -1;
        }

        /* Create data sample for writing */

        instance = LittleMsgTypeSupport::create_data();

        if (instance == NULL) {
                printf("LittleMsgTypeSupport::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 = LittleMsg_writer->register_instance(*instance);
           */

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

                printf("Writing LittleMsg, #%d\n", count);
                instance->sender = "bhanders";
                instance->message = "hello world";

                /* Modify the data to be sent here */

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

                NDDSUtility::sleep(send_period);
        }

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

        /* Delete data sample */
        retcode = LittleMsgTypeSupport::delete_data(instance);
        if (retcode != DDS_RETCODE_OK) {
                printf("LittleMsgTypeSupport::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

#ifdef RTI_VX653
const unsigned char* __ctype = *(__ctypePtrGet());

extern "C" void usrAppInit ()
{
#ifdef  USER_APPL_INIT
        USER_APPL_INIT;         /* for backwards compatibility */
#endif

        /* add application specific code here */
        taskSpawn("pub", RTI_OSAPI_THREAD_PRIORITY_NORMAL, 0x8, 0x150000, (FUNCPTR)publ
isher_main, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);

}
#endif
```