```
/* little_subscriber.cxx

   A subscription example

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

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

   Example subscription of type LittleMsg 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>/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>
       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"

class LittleMsgListener : 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 LittleMsgListener::on_data_available(DDSDataReader* reader)
{
    LittleMsgDataReader *LittleMsg_reader = NULL;
    LittleMsgSeq data_seq;
    DDS_SampleInfoSeq info_seq;
    DDS_ReturnCode_t retcode;
    int i;

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

    retcode = LittleMsg_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) {
            LittleMsgTypeSupport::print_data(&data_seq[i]);
        }
    }

    retcode = LittleMsg_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 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) {
        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;
    LittleMsgListener *reader_listener = NULL;
    DDSDataReader *reader = NULL;
    DDS_ReturnCode_t retcode;
    const char *type_name = NULL;
    int count = 0;
    DDS_Duration_t receive_period = {4,0};
    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) {
        printf("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) {
        printf("create_subscriber error\n");
        subscriber_shutdown(participant);
        return -1;
```

```
    }

    /* Register the type before creating the 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);
        subscriber_shutdown(participant);
        return -1;
    }

    /* To customize the 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");
        subscriber_shutdown(participant);
        return -1;
    }

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

    /* 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) {
        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("LittleMsg 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

#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("sub", RTI_OSAPI_THREAD_PRIORITY_NORMAL, 0x8, 0x150000, (FUNCPTR)subscrib
er_main, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);

}
#endif
```