A Tutorial on Interfacing the Object Management Group (OMG) Data Distribution Service (DDS) with LabView

Kevin Smith, NASA Kennedy Space Center NE-A2

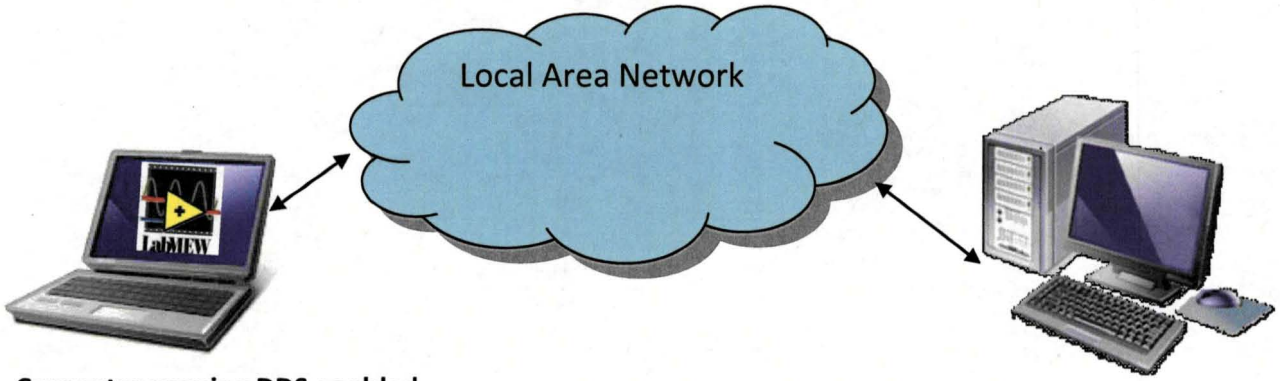
Abstract

This tutorial will explain the concepts and steps for interfacing a National Instruments LabView virtual instrument (VI) running on a Windows platform with another computer via the Object Management Group (OMG) Data Distribution Service (DDS) as implemented by the Twin Oaks Computing CoreDX. This paper is for educational purposes only and therefore, the referenced source code will be simplistic and void of all error checking. Implementation will be accomplished using the C programming language.

Problem to be Solved

The user will enter an array of ten numbers in a LabView virtual instrument (VI). The array will be transmitted to another computer via DDS where data calculations such as determining minimum value, maximum value and the average will be performed. These results will be transmitted back to the VI and displayed for the user.

Architectural Concept



Computer running DDS enabled LabView VI which collects data from user and displays statistics received from remote computer.

Computer with DDS enabled application which performs simple statistics on data received via DDS.

Steps Involved

1. Define the DDS interface. This is called the **Interface Definition Language** and specifies data structures for the exchange of messages across the network.
2. Implement a command line program (`TutorialCalculator.cpp`) that reads from the network via DDS and responds to any user data received. This program will be run from a DOS box.
3. Create a Windows Dynamic Link Library (DLL) – `LabViewDDS_Tutorial.dll`. This is Microsoft's version of a shared library.
4. Create a LabView Virtual Instrument (VI) which references the DLL.

Interface Definition Language

The first step is to define the "topics" needed for the DDS communication. A topic is a message consisting of a name, a data type and a quality of service. The topic is created within the application's source code. The data type is defined in an interface definition language (IDL) file. CoreDX DDS names their IDL files with a .ddl expension. CoreDX includes an IDL compiler that takes the .ddl file and generates several source code files.

tutorial.ddl

```
module TUTORIAL {

    struct UserData {
        float data[10];
    };

    struct StatResults {
        float max;
        float min;
        float average;
    };

};
```

The CoreDX DDL Compiler takes the tutorial.ddl file and generates the following source code:

tutorial.h	tutorial.c
tutorialDataReader.h	tutorialDataReader.c
tutorialDataWriter.h	tutorialDataWriter.c
tutorialTypeSupport.h	tutorialTypeSupport.c

Implementing a DDS Enabled Program

A conventional C program is written (see Appendix A for the complete listing). The DDS unique header files are included in addition to the basic Windows libraries.

```
#include <stdio.h>
#include <windows.h>

#include <dds/dds.h>

#include "tutorial.h"
#include "tutorialDataReader.h"
#include "tutorialDataWriter.h"
#include "tutorialTypeSupport.h"
```

The DDS framework consists of a domain containing publishers, subscribers, topics, data reader and data writer.

```
DDS_DomainParticipant       domain;

DDS_Subscriber              subscriber;
DDS_Publisher               publisher;

DDS_Topic                   userData_Topic;
DDS_Topic                   statResults_Topic;

DDS_DataReader              userData_DataReader;
DDS_DataWriter              statResults_DataWriter;
```

In the program's main() function, the domain, one publisher and one subscriber are created. The topic data types are registered with the domain followed by topic creation. And finally, a data reader and data writer are created for the topics.

```
domain = DDS_DomainParticipantFactory_create_participant( 0,
                                    DDS_PARTICIPANT_QOS_DEFAULT,
                                    NULL,
                                    0 );

publisher = DDS_DomainParticipant_create_publisher( domain,
                                    DDS_PUBLISHER_QOS_DEFAULT,
                                    NULL,
                                    0 );

subscriber = DDS_DomainParticipant_create_subscriber( domain,
                                    DDS_SUBSCRIBER_QOS_DEFAULT,
                                    NULL,
                                    0 );

TUTORIAL_UserDataTypeSupport_register_type( domain, NULL );
TUTORIAL_StatResultsTypeSupport_register_type( domain, NULL );
```

```
userData_Topic = DDS_DomainParticipant_create_topic( domain,
                                                     "UserData",
                                                     "UserData",
                                                     DDS_TOPIC_QOS_DEFAULT,
                                                     NULL,
                                                     0 );

statResults_Topic = DDS_DomainParticipant_create_topic( domain,
                                                        "StatResults",
                                                        "StatResults",
                                                        DDS_TOPIC_QOS_DEFAULT,
                                                        NULL,
                                                        0 );

statResults_DataWriter = DDS_Publisher_create_datawriter(
                                        publisher,
                                        statResults_Topic,
                                        DDS_DATAWRITER_QOS_DEFAULT,
                                        NULL,
                                        0 );

userData_DataReader = DDS_Subscriber_create_datareader(
                    subscriber,
                    DDS_Topic_TopicDescription(userData_Topic),
                    DDS_DATAREADER_QOS_DEFAULT,
                    &drListener,
                     DDS_DATA_AVAILABLE_STATUS );
```

Note that in the creation of the data reader, a data reader listener function is specified (`&drListener`). This framework functionality allows the program to be notified asynchronously should any new data arrive.

A function named `dr_on_data_avail` is used as the data reader callback. When this function is called, the newest user data is read, statistics are calculated and then written back into the DDS infrastructure. The read function looks like this:

```
void dr_on_data_avail(DDS_DataReader dr)
{
      TUTORIAL_UserDataPtrSeq             samples;
      DDS_SampleInfoSeq                   samples_info;
      DDS_ReturnCode_t                    retval;

      INIT_SEQ(samples);
      INIT_SEQ(samples_info);

      retval = TUTORIAL_UserDataDataReader_take( dr, &samples,
                                        &samples_info,
                                        DDS_LENGTH_UNLIMITED,
                                        DDS_ANY_SAMPLE_STATE,
                                        DDS_ANY_VIEW_STATE,
                                        DDS_ANY_INSTANCE_STATE );
```

```
      if ( DDS_RETCODE_OK == retval )
      {
            unsigned int i;

            /* iterrate through the samples */
            for ( i = 0;i < samples._length; i++)
            {
                  if ( samples_info._buffer[i]->valid_data)
                  {
                        printf("Tutorial:: User data received:
%4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f\n",
                                          samples._buffer[i]->data[0],
                                          samples._buffer[i]->data[1],
                                          samples._buffer[i]->data[2],
                                          samples._buffer[i]->data[3],
                                          samples._buffer[i]->data[4],
                                          samples._buffer[i]->data[5],
                                          samples._buffer[i]->data[6],
                                          samples._buffer[i]->data[7],
                                          samples._buffer[i]->data[8],
                                          samples._buffer[i]->data[9]);
.
.
.
```

After the data is received, calculations are made and written to the StatResult topic as follows:

```
TUTORIAL_StatResults msg;
int element;
float sum;
msg.max = 0.0;
msg.min = 0.0;

for (element=0; element<10; element++)
{
      sum += samples._buffer[i]->data[element];

      if ( samples._buffer[i]->data[element] < msg.min )
            msg.min = samples._buffer[i]->data[element];

      if ( samples._buffer[i]->data[element] > msg.max )
            msg.max = samples._buffer[i]->data[element];
}

msg.average = sum / 10.0f;

TUTORIAL_StatResultsDataWriter_write( statResults_DataWriter,
                                      &msg,
                                      DDS_HANDLE_NIL );

printf("Statistics: max = %6.2f  min = %6.2f  avg = %6.2f\n",
            msg.max, msg.min, msg.average);
```

Creating the Windows DLL

The Windows DLL is a shared library with one or more functions exposed. The complete listing is in Appendix B. The library also contains global data shared amongst the functions. For this tutorial, there are 4 simple functions: open DDS, read from DDS, write to DDS and close DDS. The function prototypes look like this:

```
__declspec(dllexport) void openDDS( void );

__declspec(dllexport) void writeDDS_data( float * data );

__declspec(dllexport) void readDDS_results( float * max,
                                            float * min,
                                            float * average );

__declspec(dllexport) void closeDDS(void);
```

For this tutorial, the C code is kept as simplistic as possible. Thus, the functions do not return any indicator of success or failure. This would not be acceptable for any code other than for demonstration purposes.

As previously described for the stand alone statistics calculator program, openDDS() creates the domain, publisher, subscriber, data types, topics data writers and data readers.

The writeDDS() function receives an array of floating point data from LabVIEW and writes it to the DDS domain. The array is assumed to contain 10 elements – an assumption only acceptable for a tutorial like this.

```
__declspec(dllexport) void writeDDS_data( float * data )
{
      TUTORIAL_UserData cmd;
      int i;

      for (i=0; i<10; i++)
      {
            cmd.data[i] = data[i];
      }

      TUTORIAL_UserDataDataWriter_write( userData_DataWriter,
                                         &cmd,
                                         DDS_HANDLE_NIL );
}
```

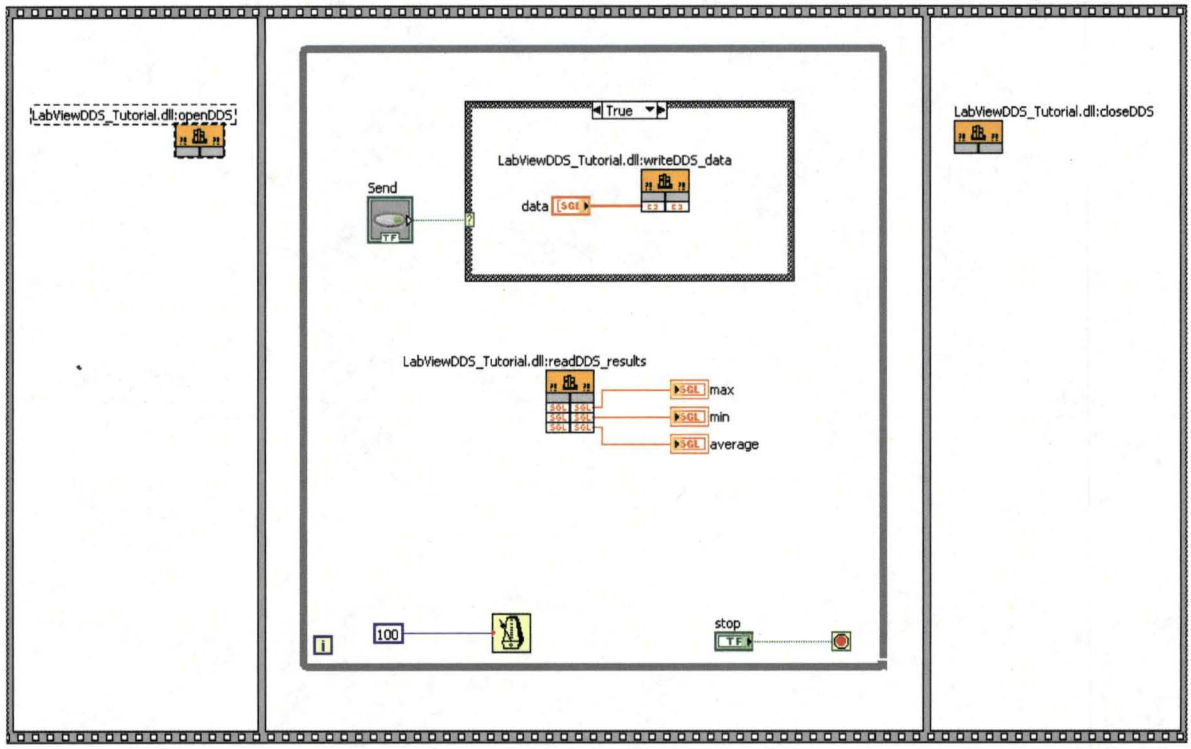
The readDDS() function asks the framework for any available data. The take() version of this call is non-blocking. Following the take(), the results must be checked to determine if there is any data to read available.

```
__declspec(dllexport) void readDDS_results( float * max,
                                            float * min,
                                            float * average )
{
      TUTORIAL_StatResultsPtrSeq          samples;
      DDS_SampleInfoSeq                   samples_info;

      INIT_SEQ(samples);
      INIT_SEQ(samples_info);

      TUTORIAL_StatResultsDataReader_take(
                        statResults_DataReader,
                        &samples,
                        &samples_info,
                        DDS_LENGTH_UNLIMITED,
                        DDS_ANY_SAMPLE_STATE,
                        DDS_ANY_VIEW_STATE,
                        DDS_ANY_INSTANCE_STATE );

      if( samples._length > 0 &&
            samples_info._buffer[0]->valid_data )
      {
            /* Data is available for use */
            *max = samples._buffer[0]->max;
            *min = samples._buffer[0]->min;
            *average = samples._buffer[0]->average;

            TUTORIAL_StatResultsDataReader_return_loan(
                                    statResults_DataReader,
                                    &samples,
                                    &samples_info );
      }
}
```

The DLL is created using Microsoft Visual C++ 2008 Express Edition (available for free from Microsoft).

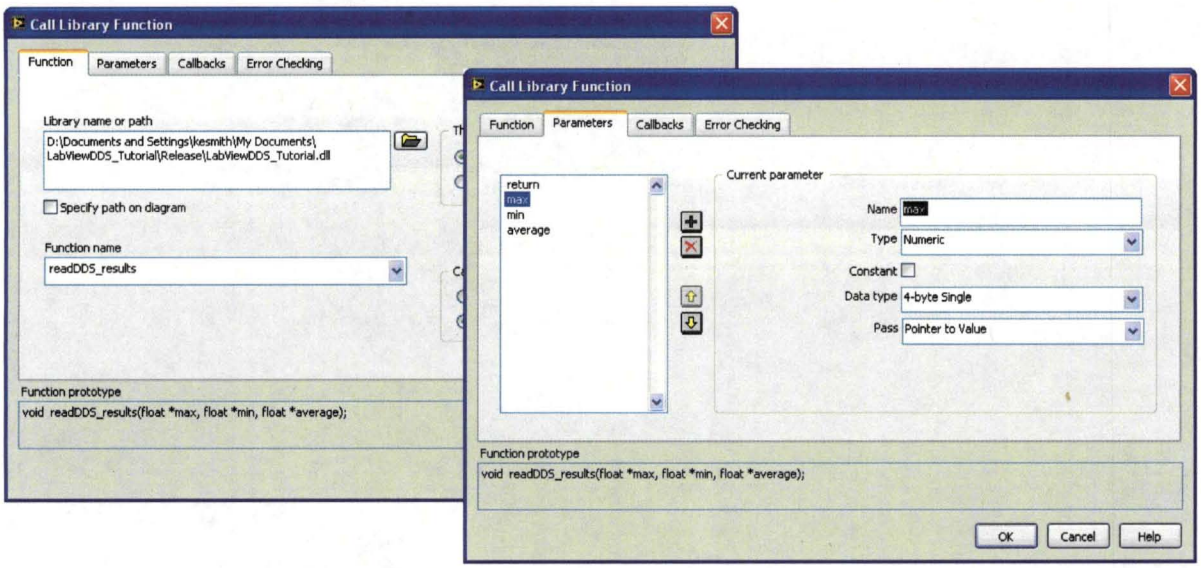
Calling the DLL from LabVIEW

The DLL is called from LabVIEW using the "Call Library Function" block. The block diagram of a simple VI looks like this:

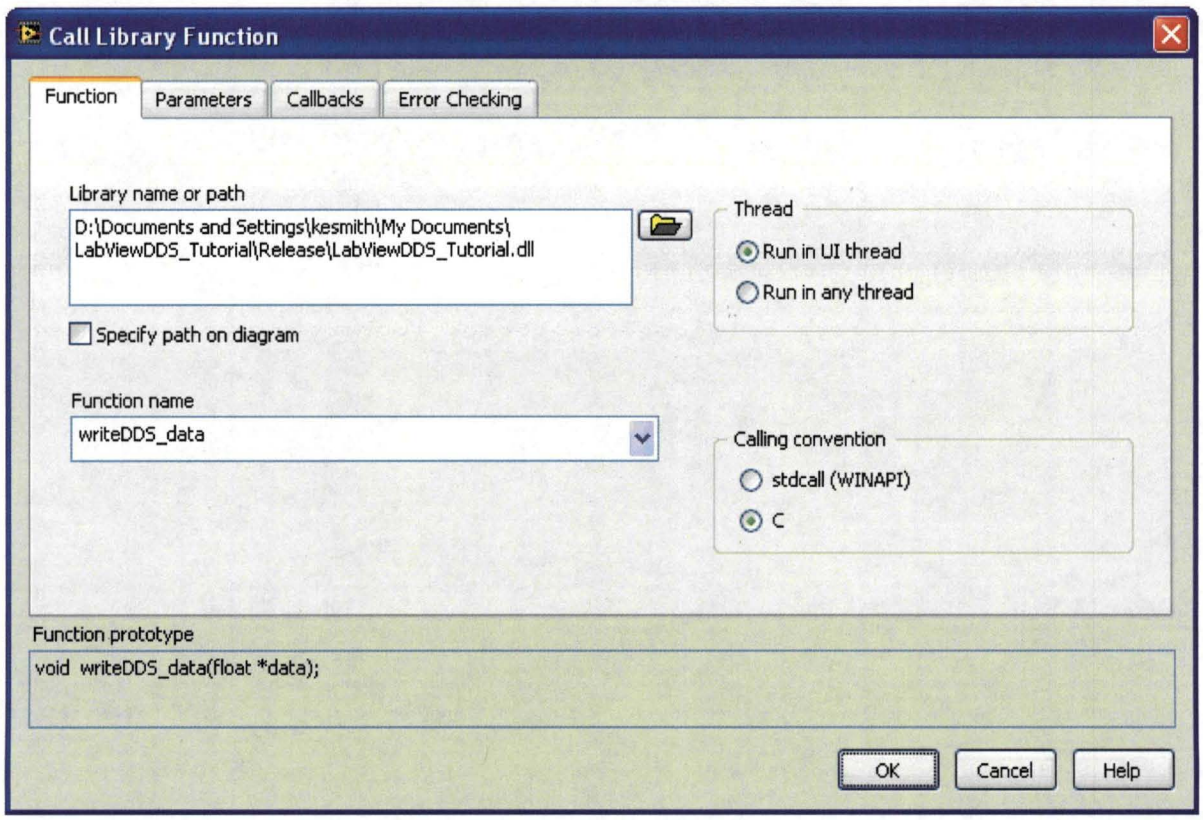


The outer container is a flat sequence. The first frame opens or instantiates the DDS framework. The second frame has a while loop that runs indefinitely until the user decides to quit. The third frame is only reached after the user has decided to quit and contains a call to the function closeDDS() that deletes the DDS domain.

The readDDS "Call Library Function" block is wired to 3 numeric indicators. This function is called every iteration of the while loop. A metronome limits the loop time to 100ms. The readDDS block is created from the following LabVIEW dialogs:

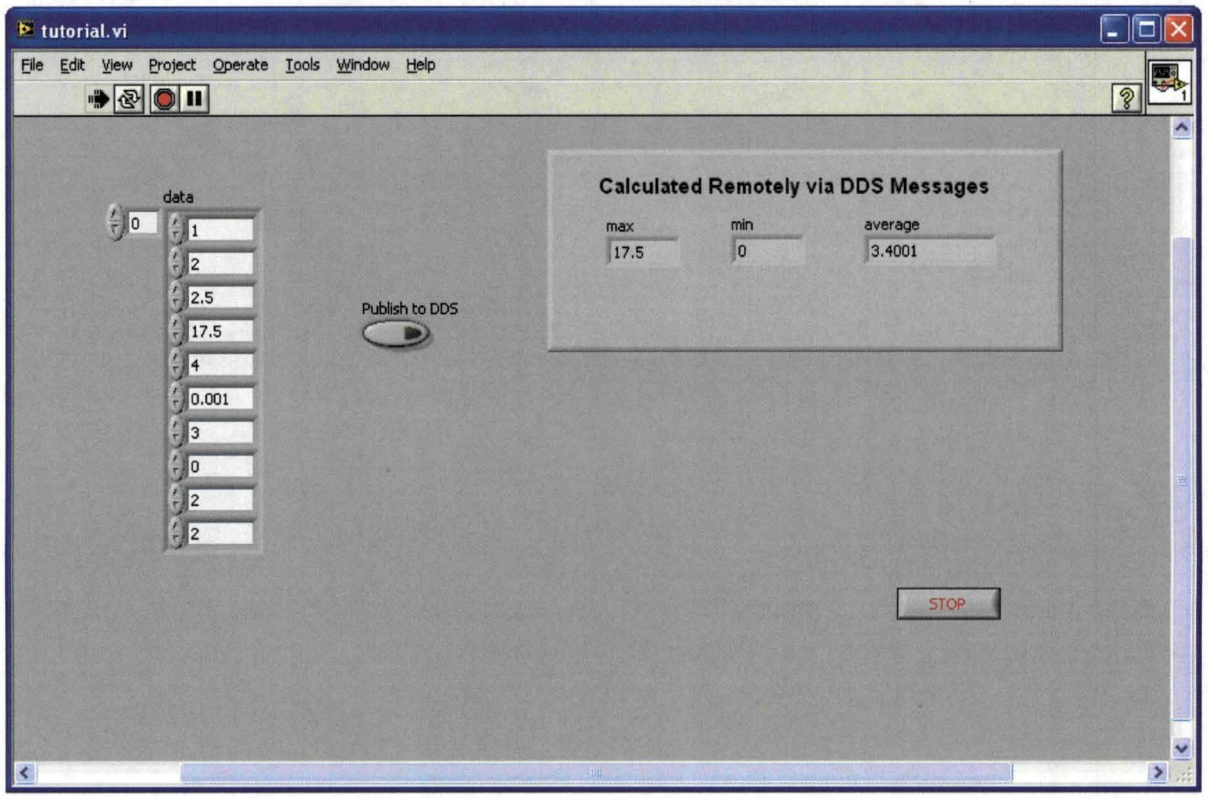


The writeDDS block is configured as follows:

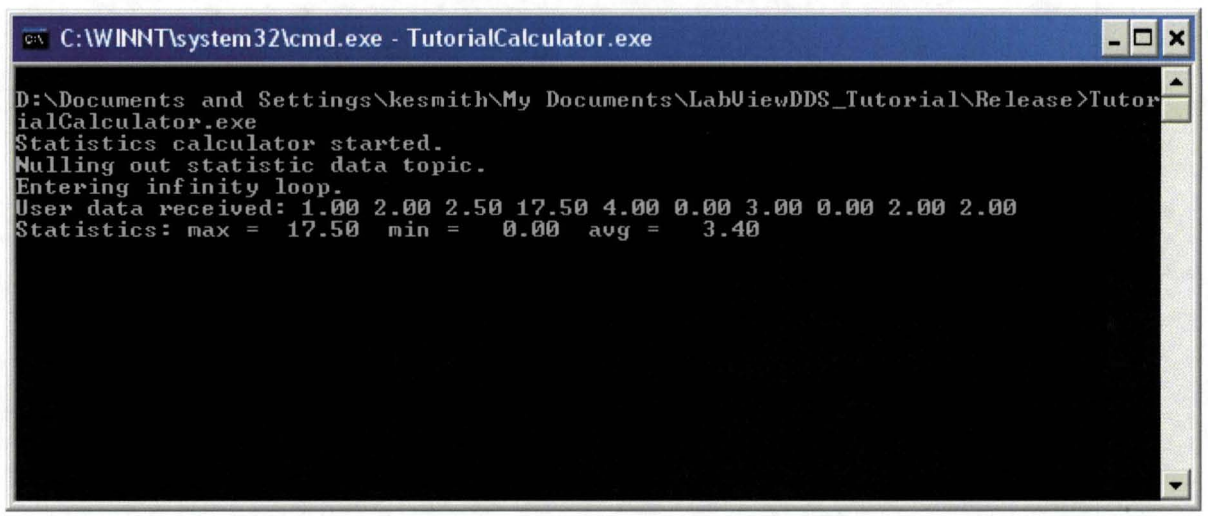


Conclusion

Finally, a few snap shots of this tutorial in use. First the LabVIEW VI:



And the statistics calcluator standalone program.



*This tutorial assumed installation of the CoreDX libraries and proper Microsoft Visual C++ configuration.

Appendix A – TutorialCalculator.cpp

```
//
// TutorialCalculator.cpp

#if defined (__cplusplus)
extern "C" {
#endif

#include <stdio.h>
#include <windows.h>

#include <dds/dds.h>

#include "tutorial.h"
#include "tutorialDataReader.h"
#include "tutorialDataWriter.h"
#include "tutorialTypeSupport.h"


/*
 * Declare some global variables needed for the DDS interface.
 */
DDS_DomainParticipant          domain;

DDS_Subscriber                 subscriber;
DDS_Publisher                  publisher;

DDS_Topic                      userData_Topic;
DDS_Topic                      statResults_Topic;

DDS_DataReader                 userData_DataReader;
DDS_DataWriter                 statResults_DataWriter;


void dr_on_data_avail(DDS_DataReader dr)
{
        TUTORIAL_UserDataPtrSeq              samples;
        DDS_SampleInfoSeq                    samples_info;
        DDS_ReturnCode_t                     retval;

        INIT_SEQ(samples);
        INIT_SEQ(samples_info);

        retval = TUTORIAL_UserDataDataReader_take( dr, &samples, &samples_info,
                                                   DDS_LENGTH_UNLIMITED,
                                                   DDS_ANY_SAMPLE_STATE,
                                                   DDS_ANY_VIEW_STATE,
                                                   DDS_ANY_INSTANCE_STATE );

        if ( DDS_RETCODE_OK == retval )
        {
                unsigned int i;

                /* iterrate through the samples */
                for ( i = 0;i < samples._length; i++)
                {
                        if ( samples_info._buffer[i]->valid_data)
                        {
                                TUTORIAL_StatResults msg;
                                int element;
                                float sum;

                                printf("User data received: %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f
%4.2f %4.2f %4.2f %4.2f\n",
                                                                samples._buffer[i]->data[0],
                                                                samples._buffer[i]->data[1],
                                                                samples._buffer[i]->data[2],
                                                                samples._buffer[i]->data[3],
                                                                samples._buffer[i]->data[4],
```

```
                                                samples._buffer[i]->data[5],
                                                samples._buffer[i]->data[6],
                                                samples._buffer[i]->data[7],
                                                samples._buffer[i]->data[8],
                                                samples._buffer[i]->data[9]);

                                msg.max = 0.0;
                                msg.min = 0.0;
                                for (element=0; element<10; element++)
                                {
                                        sum += samples._buffer[i]->data[element];

                                        if ( samples._buffer[i]->data[element] < msg.min )
                                                msg.min = samples._buffer[i]->data[element];

                                        if ( samples._buffer[i]->data[element] > msg.max )
                                                msg.max = samples._buffer[i]->data[element];

                                }

                                msg.average = sum / 10.0f;

                                TUTORIAL_StatResultsDataWriter_write( statResults_DataWriter,
                                                                      &msg,
                                                                      DDS_HANDLE_NIL );

                                printf("Statistics: max = %6.2f  min = %6.2f  avg = %6.2f\n",
                                                msg.max,
                                                msg.min,
                                                msg.average);
                        }
                }

                TUTORIAL_UserDataDataReader_return_loan( dr, &samples, &samples_info );
        }
}

DDS_DataReaderListener drListener =
        { NULL, NULL, NULL, NULL, dr_on_data_avail, NULL, NULL};

int main()
{
        printf("Statistics calculator started.\n");

        /*
         * Create the DDS domain
         */
        domain =
                DDS_DomainParticipantFactory_create_participant( 0,
                                                                 DDS_PARTICIPANT_QOS_DEFAULT,
                                                                 NULL,
                                                                 0 );

        /*
         * Create a publisher for the domain
         */
        publisher =
                DDS_DomainParticipant_create_publisher( domain,
                                                        DDS_PUBLISHER_QOS_DEFAULT,
                                                        NULL,
                                                        0 );

        /*
         * Create a subscriber for the domain
         */
        subscriber =
                DDS_DomainParticipant_create_subscriber( domain,
                                                         DDS_SUBSCRIBER_QOS_DEFAULT,
                                                         NULL,
                                                         0 );
```

```
        /*
         * Register the data types we need with the domain.
         */
        TUTORIAL_UserDataTypeSupport_register_type( domain, NULL );
        TUTORIAL_StatResultsTypeSupport_register_type( domain, NULL );

        /*
         * Create our topics within the domain
         */
        userData_Topic =
                DDS_DomainParticipant_create_topic( domain,
                                                    "UserData",
                                                    "UserData",
                                                    DDS_TOPIC_QOS_DEFAULT,
                                                    NULL,
                                                    0 );
        statResults_Topic =
                DDS_DomainParticipant_create_topic( domain,
                                                    "StatResults",
                                                    "StatResults",
                                                    DDS_TOPIC_QOS_DEFAULT,
                                                    NULL,
                                                    0 );

        /*
         * Create a data writer attached to our publisher for
         * our specific topic
         */
        statResults_DataWriter =
                DDS_Publisher_create_datawriter( publisher,
                                                 statResults_Topic,
                                                 DDS_DATAWRITER_QOS_DEFAULT,
                                                 NULL,
                                                 0 );

        /*
         * Create a data reader attached to our subscriber for
         * our specific topic
         */
        userData_DataReader =
                DDS_Subscriber_create_datareader( subscriber,
                                                  DDS_Topic_TopicDescription(userData_Topic),
                                                  DDS_DATAREADER_QOS_DEFAULT,
                                                  &drListener,
                                                  DDS_DATA_AVAILABLE_STATUS );

        printf("Nulling out statistic data topic.\n");

        TUTORIAL_StatResults msg;

        msg.average = 0.0;
        msg.max = 0.0;
        msg.min = 0.0;

        TUTORIAL_StatResultsDataWriter_write( statResults_DataWriter,
                                              &msg,
                                              DDS_HANDLE_NIL );

        printf("Entering infinity loop.\n");
        while (1)
        {
                Sleep(100);
        }

        return 0;
}


#if defined (__cplusplus)
}
#endif
```

Appendix B – LabViewDDS_Tutorial.cpp

```
/*
 * LabViewDDS_Tutorial.cpp
 *
 *   Synopsis: A tutorial for using DDS in LabView.  Provides an interface to the
 *             Twin Oaks CoreDX implementation of the Object Management
 *             Group's (OMG's) Data Distribution System (DDS).  In addition,
 *             a context of global DDS entities is created to allow the
 *             interface to function.  The functions are designed to be
 *             called from within a LabView Virtual Instrument (VI).  This
 *             collection of functions is built into a Windows DLL and called
 *             from LabView using LabView's 'call library function'.
 *
 */

#if defined (__cplusplus)
extern "C" {
#endif

#include <stdio.h>
#include <dds/dds.h>

/**
 * DDL Supporting Files
 *
 *     Note: Assumes IDL is contained in a file named "tutorial.ddl".
 */
#include "tutorial.h"
#include "tutorialDataReader.h"
#include "tutorialDataWriter.h"
#include "tutorialTypeSupport.h"


/*
 * Library Functions Exposed Publicly from the
 * Dynamic Link Library (DLL).
 */
__declspec(dllexport) void openDDS( void );
__declspec(dllexport) void writeDDS_data( float * data );
__declspec(dllexport) void readDDS_results( float * max, float * min, float * average );
__declspec(dllexport) void closeDDS(void);


/*
 * Declare some global variables needed for the DDS interface.
 */
DDS_DomainParticipant         domain;

DDS_Subscriber                subscriber;
DDS_Publisher                 publisher;

DDS_Topic                     userData_Topic;
DDS_Topic                     statResults_Topic;

DDS_DataWriter                userData_DataWriter;
DDS_DataReader                statResults_DataReader;


/*
 * openDDS()      - Creates the domain participant, a publisher for the domain
 *                  and a subscriber for the domain.  The data types for the
 *                              topics are registered with the domain and then the
 *                              corresponding topics are created.  Finally, a data writer
 *                              and a data reader are created.
 *
 */
__declspec(dllexport) void openDDS( void )
{
       /*
```

```
 * Create the DDS domain
 */
domain =
       DDS_DomainParticipantFactory_create_participant( 0,

 DDS_PARTICIPANT_QOS_DEFAULT,

 NULL,

 0 );

/*
 * Create a publisher for the domain
 */
publisher =
       DDS_DomainParticipant_create_publisher( domain,
                                               DDS_PUBLISHER_QOS_DEFAULT,
                                               NULL,
                                               0 );

/*
 * Create a subscriber for the domain
 */
subscriber =
       DDS_DomainParticipant_create_subscriber( domain,
                                               DDS_SUBSCRIBER_QOS_DEFAULT,
                                               NULL,
                                               0 );

/*
 * Register the data types we need with the domain.
 */
TUTORIAL_UserDataTypeSupport_register_type( domain, NULL );
TUTORIAL_StatResultsTypeSupport_register_type( domain, NULL );

/*
 * Create our topics within the domain
 */
userData_Topic =
       DDS_DomainParticipant_create_topic( domain,
                                               "UserData",
                                               "UserData",
                                               DDS_TOPIC_QOS_DEFAULT,
                                               NULL,
                                               0 );
statResults_Topic =
       DDS_DomainParticipant_create_topic( domain,
                                               "StatResults",
                                               "StatResults",
                                               DDS_TOPIC_QOS_DEFAULT,
                                               NULL,
                                               0 );

/*
 * Create a data writer attached to our publisher for
 * our specific topic
 */
userData_DataWriter =
       DDS_Publisher_create_datawriter( publisher,
                                               userData_Topic,
                                               DDS_DATAWRITER_QOS_DEFAULT,
                                               NULL,
                                               0 );

/*
 * Create a data reader attached to our subscriber for
 * our specific topic
 */
statResults_DataReader =
       DDS_Subscriber_create_datareader( subscriber,
                                       DDS_Topic_TopicDescription(statResults_Topic),
```

```
                                        DDS_DATAREADER_QOS_DEFAULT,
                                        NULL,
                                        0 );
}


__declspec(dllexport) void writeDDS_data( float * data )
{
        TUTORIAL_UserData cmd;
        int i;

        for (i=0; i<10; i++)
        {
                cmd.data[i] = data[i];
        }

        TUTORIAL_UserDataDataWriter_write( userData_DataWriter,
                                        &cmd,
                                        DDS_HANDLE_NIL );
}


__declspec(dllexport) void readDDS_results( float * max, float * min, float * average )
{
        TUTORIAL_StatResultsPtrSeq          samples;
        DDS_SampleInfoSeq                   samples_info;

        INIT_SEQ(samples);
        INIT_SEQ(samples_info);

        TUTORIAL_StatResultsDataReader_take( statResults_DataReader,
                                         &samples,
                                         &samples_info,
                                         DDS_LENGTH_UNLIMITED,
                                         DDS_ANY_SAMPLE_STATE,
                                         DDS_ANY_VIEW_STATE,
                                         DDS_ANY_INSTANCE_STATE );

        if( samples._length > 0 && samples_info._buffer[0]->valid_data )
        {
                /* Data is available for use */
                *max = samples._buffer[0]->max;
                *min = samples._buffer[0]->min;
                *average = samples._buffer[0]->average;

                TUTORIAL_StatResultsDataReader_return_loan( statResults_DataReader,

        &samples,

        &samples_info );
        }
}


/*
 * closeDDS()     - Cleans up when we're finished with the DDS interface.
 *
 */
__declspec(dllexport) void closeDDS(void)
{
        DDS_DomainParticipant_delete_contained_entities(domain);
        DDS_DomainParticipantFactory_delete_participant(domain);
}


#if defined (__cplusplus)
}
#endif
```

References

An Overview of Accessing DLLs or Shared Libraries from LabVIEW
< http://zone.ni.com/devzone/cda/tut/p/id/3009>

CoreDX Data Distribution Service, Programmer's Guide, Version 3.2
< http://www.twinoakscomputing.com/documents/CoreDX_DDS_Programmers_Guide.pdf>

Quick Start Guide for C, Version 3.0, Twin Oaks Computing, Inc.,
< http://www.twinoakscomputing.com/documents/CoreDX_DDS_C_QuickStartGuide.pdf>