TWIN OAKS COMPUTING INC.

PRACTICAL MIDDLEWARE EXPERTISE

Quick Start Guide for C

Version 3.4

Castle Rock, CO

80108

Welcome

Welcome to CoreDX DDS, a high-performance implementation of the OMG Data Distribution Service (DDS) standard. The CoreDX DDS Publish-Subscribe messaging infrastructure provides high-throughput, low-latency data communications.

This Quick Start will guide you through the basic installation of CoreDX DDS, including installation and compiling and running an example C application. You will learn how easy it is to integrate CoreDX DDS into an application. This Quick Start Guide is tailored for C applications, and the examples differ slightly for C++ applications.

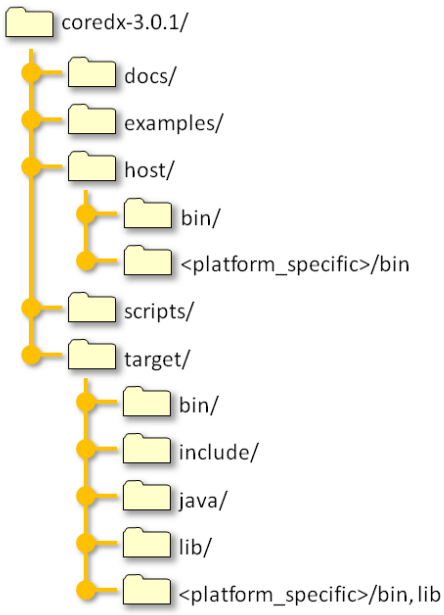
Installation

First things first: get CoreDX DDS onto your development system! Here's what you need to do:

1. Once you have obtained CoreDX DDS from Twin Oaks Computing (or from the Eval CD), unpack the appropriate distribution for your machine somewhere on your system. We'll refer to this directory throughout this guide as COREDX_TOP. For example, on a UNIX system this command will extract the distribution into the current directory:

   ```
   gunzip -c coredx-3.0-Linux_2.6_i686-Release.tar.gz | tar xvf -
   ```

   CoreDX DDS is available for multiple platform architectures, and multiple platform architectures of CoreDX DDS can be installed in the same top level (COREDX_TOP) directory. The directory structure under COREDX_TOP will look like:

2. If you are using an evaluation copy of CoreDX DDS, follow the instructions you received when you downloaded the software to obtain an evaluation license. Otherwise, use the purchased license provided by Twin Oaks Computing. Once you have the license, put this file somewhere on your system. We'll refer to the full path name to this file throughout this guide as LICENSE_FILE.

Building an Application

Next, integrate CoreDX DDS into an application! We've provided a sample data types and applications with the distribution (located in COREDX_TOP/examples). You can use these examples or create your own while going though the following steps. Our example Makefiles were built for **gcc** (UNIX systems) and Visual Studio **cl.exe** (Windows systems).

1. Create the Data Definition Language (DDL) file for the data type(s) you will use for communications. The CoreDX DDS DDL syntax is very similar to the OMG IDL syntax for describing data types. Here is the "hello world" example provided with the distribution:

hello.ddl

```
struct StringMsg
{
  string msg;
};
```

2. Tell the CoreDX DDS DDL compiler where your evaluation license is located. You can copy the license file into the directory where you will run the compiler; or, set the following environment variable, using the full path to your license file. (This assumes a *bash* style shell):

```
% export TWINOAKS_LICENSE_FILE=LICENSE_FILE
```

3. Compile the DDL to generate the type specific code using the CoreDX DDS DDL compiler. The DDL compiler is a *host* tool, and is located in the host subdirectory. Actually, there may be more than one DDL compiler, if you have multiple platform versions of CoreDX DDS installed. In this case, choose the appropriate compiler for your architecture. Assuming we are using a Linux distribution and the name of the DDL file is hello.ddl:

```
% COREDX_TOP/host/bin/Linux_26_x86_gcc43_coredx_ddl -f
hello.ddl
```

   The compilation will generate the following files (names are based on the DDL filename):

   - hello.h and .c
   - helloTypeSupport.h and .c
   - helloDataReader.h and .c
   - helloDataWriter.h and .c

4. Create code to publish data of this data type. Our sample Hello World publisher is located in COREDX_TOP/examples/hello_c/hello_pub.c.

5. Create code to subscribe to data of this data type. Our sample Hello World subscriber is located in COREDX_TOP/examples/hello_c/hello_sub.c.

6. Compile your application(s). Our Hello World example creates two applications, one for the publisher and one for the subscriber. This is not necessary, and is completely dependent on your application architecture. Your application will require the objects from the generated type support code above, as well as your publisher and/or subscriber code.

CoreDX DDS will require the following paths and libraries when compiling your application:

```
Include Path:        -I${COREDX_TOP}/target/include
Library Path:  \
     -L${COREDX_TOP}/target/${COREDX_TARGET}/lib
Static Libraries:    -ldds
```

We've provided a Makefile and NMakefile for compiling our example, along with a Windows VisualStudio solution file. You can use these as a reference for compiling your application. Our Makefile requires three environment variables:

1. COREDX_TOP
2. COREDX_HOST
3. COREDX_TARGET

The COREDX_TOP is a path name to the location of your CoreDX DDS distribution(s). COREDX_HOST and COREDX_TARGET are the platform architectures you are compiling on and compiling for (these can be the same). These values can be set manually, or determined by running the script: COREDX_TOP/scripts/cdxenv.sh or cdxenv.bat.

Using our Makefiles

Our Makefiles will run the DDL compiler to generate the type specific code as well as compile the applications. To compile the Hello World sample application using our Makefiles you will need gcc and coredx_ddl in your path for UNIX, or the MS Visual Studio environment and coredx_ddl in your path for Windows. Then, simply type 'make' for UNIX or 'nmake –f NMakefile' for Windows in the appropriate directory.

This will compile two applications: hello_pub and hello_sub.

Using Visual Studio

To compile using Microsoft's Visual Studio, the environment variables listed above, including the TWINOAKS_LICENSE_FILE variable must be set. They can be set as part of your user configuration, or you can set them in a Visual Studio command prompt. If you set the environment variables in a Visual Studio command prompt, you must then run Visual Studio from that same command prompt.

Next, open the example Visual Studio project solution file. This solution contains 2 projects: hello_pub and hello_sub.

```
File->Open
```

```
Navigate to COREDX_TOP\examples\hello_c, and then
select "hello_c.sln"
```

This solution was built using VisualStudio 2005. If you are using Visual Studio 2008 or later, you will be prompted to convert the solution to Visual Studio 2008 (follow the prompts to complete the conversion).

```
In the "Solution Explorer", Right click on the
'hello_c' solution
Select "build solution"
```

This will compile two applications: hello_pub and hello_sub.

Running a Test Application

You've written some code, generated some code, and compiled it all. Now for seeing it all work! You will need at least one environment variable to run:

TWINOAKS_LICENSE_FILE = The full path to your evaluation license

Run your application(s). The sample Hello World has two applications: hello_pub and hello_sub:

```
COREDX_TOP/examples/hello_c/hello_sub
COREDX_TOP/examples/hello_c/hello_pub
```

Congratulations! You have now built and run two applications that are communicating using CoreDX DDS.

Figure 1 shows a picture of what you have built:

Figure 1: Example Components

You can run multiple Publishers and multiple Subscribers to immediately see the dynamic nature of the DDS network infrastructure. These Publishers and Subscribers can be run on the same host or on multiple hosts across a network.

A few notes about the Transport

The CoreDX DDS transport conforms to the Real-Time Publish-Subscribe (RTPS) Wire Protocol. This transport does not use a stand-alone transport daemon, and does not require configuration of any operating system services.

Configuring the Transport

The CoreDX DDS RTPS transport can be configured using the following environment variables.

Environment Variable	Values	Description
COREDX_IP_ADDR	Valid, Local IP address	As part of discovery, each DDS DomainParticipant advertises local IP addresses that peers can use to communicate with it. If your machine has multiple network interfaces, CoreDX DDS will by default advertise (and use) all interfaces for DDS communications. This may generate unnecessary network traffic on some of those networks. This environment variable will limit DDS traffic to just one interface – the interfaces specified by the IP address.
COREDX_USE_MULTICAST	"YES", "NO"	By default, CoreDX DDS RTPS will use multicast for data communications between DDS participants where it can. To specify unicast data communications, set this environment variable to "NO". This only effects data communications, discovery will still use multicast.
COREDX_MULTICAST_TTL	integer > 0	This is the time to live (TTL) set on all the multicast packets (including those used for discovery). By default, COREDX DDS uses the operating system configured TTL (generally "1"). Increasing this allows the multicast packets to survive going through network routers.
COREDX_MIX_TX_BUFFER_SIZE	400 - 65400	(in bytes) The CoreDX DDS RTPS transport will combine multiple data packets to send over the network to reduce network overhead and improve performance. By default, the transmit buffer size is *dynamic*. For cases where IP fragmentation and reassembly is not implemented well in either the network hardware or operating system, it is necessary to fix the transmit buffer to a smaller size. This environment variable controls the minimum size of the transmit buffer on every DataWriter within

Environment Variable	Values	Description
		a DomainParticipant (including Built-in DataWriters).
COREDX_MAX_TX_BUFFER_SIZE	401-65400	(in bytes) The CoreDX DDS RTPS transport will combine multiple data packets to send over the network to reduce network overhead and improve performance. By default, the transmit buffer size is *dynamic*. For cases where IP fragmentation and reassembly is not implemented well in either the network hardware or operating system, it is necessary to fix the transmit buffer to a smaller size. This environment variable controls the maximum size of the transmit buffer on every DataWriter within a DomainParticipant (including Built-in DataWriters).
COREDX_RX_BUFFER_SIZE	401 - 65536	This environment variable determines the receive buffer size for every DomainParticipant. By default, this buffer is dynamically sized based on the data packets received by the DomainParticipant.

A few notes about License Files

CoreDX DDS uses development and run-time licenses. A development license is required for using the CoreDX DDS DDL compiler (coredx_ddl). A run-time license is required for making CoreDX DDS library function calls. Both licenses are contained in a license file provided by Twin Oaks Computing. Here is an example license file containing evaluation licenses for both development and run-time:

coredx.lic

```
#=====================================================================
# CoreDX DDS Evaluation License file
#
# Created:  <today> by Twin Oaks Computing, Inc.
# Contains: 30 day evaluation development licenses, evaluation run-time licenses
#
#=====================================================================

LICENSE PRODUCT=coredx_ddl BUILD=Evaluation EXP=31-Aug-2008
CUSTOMER=Company_X SIG=
4ccad329d5a10b93460ff3b249cea6733f6bd408d22b5fe9cb2a2c69b0d575e69a5d
c14b436b90c2ed6b516930452b862133cf7d2a9301d46ce99865f78c998311adeb99
3f68da82b74f1583511edab1d0de61dbe065f38955dd6596f0b564639fed231b1af8
61b6df122040173804e0e61b0dba37d6913cfc66d319217df099
LICENSE PRODUCT=coredx_c BUILD=Evaluation EXP=31-Aug-2008
CUSTOMER=Company_X SIG=
30b3c5d6f941c5ff5e46384eb1b74bd1809dfbd53ca11fa4d7442054bb260846588
c4bd7a5c7f7a986a12905b22dbdc428a67ee2d2c806ed5f1a14c35deb03e3a8ce6a
2fda8fdb7e5728c3103f239b51aca3b3911901e2e959fe020a21b7b7cb72dee8ca8
da8fa73cc69d3572738259025c212815aef2f94111580f51583e437
```

This evaluation license file contains two LICENSE lines. The first is the development license for the CoreDX DDS DDL compiler. The second is the run-time license for the CoreDX DDS library. All evaluation licenses have expiration dates. In the example file above, the licenses expire on Aug 31, 2008.

The CoreDX DDS software requires a license environment variable be set: TWINOAKS_LICENSE_FILE. This environment variable can contain either:

- The fully qualified name of the license file
- The entire LICENSE line from the license file contained in angle brackets: < >

For development (to run the coredx_ddl compiler), you must set the TWINOAKS_LICENSE_FILE environment variable to the license file.

For run-time, you can use either method listed above. If you have access to the license file from your run-time environment, this is the simplest way to use the license. Simply set a TWINOAKS_LICENSE_FILE environment variable to the license file.

If you do not have access to the license file at run-time, you can set the TWINOAKS_LICENSE_FILE environment variable to the LICENSE line. For the run-time license in the above example license file, set your TWINOAKS_LICENSE_FILE like:

```
% export TWINOAKS_LICENSE_FILE="<LICENSE PRODUCT=coredx_c BUILD=Evaluation
EXP=31-Aug-2008 CUSTOMER=Company_X SIG=30b3c5d6f941c5ff5e46384eb1b74bd1
809dfbd53ca11fa4d7442054bb260846588c4bd7a5c7f7a986a12905b22dbdc428a67ee
2d2c806ed5f1a14c35deb03e3a8ce6a2fda8fdb7e5728c3103f239b51aca3b3911901e2
e959fe020a21b7b7cb72dee8ca8da8fa73cc69d3572738259025c212815aef2f9411158
0f51583e437>"
```

Next Step: Enhancing the 'hello.ddl' DDL

The CoreDX DDS DDL syntax is virtually identical to the OMG IDL syntax, and supports basic and constructed data types, including sequences, fixed-size arrays, and structures. Here, we show how you could enhance the Hello World example to include some additional fields in the data structure.

First, edit the DDL to look like:

hello.ddl

```
struct SenderType
{
     string firstname;
     string lastname;
};

struct StringMsg
{
     SenderType        sender;
     long              time_sent;
     sequence<string>  old_msgs;
     string            msg;
};
```

Next, add some code to the publisher to populate the additional fields in the hello data type. For example:

hello_pub.c

```
/* Additional includes required */
#include <stdlib.h>  /* for malloc() */
#include <string.h>  /* for strcpy() */
#include <time.h>    /* for time()   */

int count=0;

/* Create some data to write */
stringMsg.sender.firstname = "Bob";
stringMsg.sender.lastname  = "Builder";
stringMsg.time_sent        = time(0);
INIT_SEQ(stringMsg.old_msgs);
stringMsg.msg = (char *)malloc(35);
strcpy(stringMsg.msg, "Hello WORLD!");
while ( count < 100 )
{
    DDS_ReturnCode_t ret;

    count++;
    ret = DDS_DataWriter_write(…);
    // check return code
    // sleep
    // update msg
```

```
    stringMsg.time_sent = time(0);
    seq_add(&stringMsg.old_msgs, &stringMsg.msg);
    stringMsg.msg = (char *)malloc(35);
    sprintf(stringMsg.msg, "Hello #%d to the world!", count );
}

/* Cleanup */
for ( count=0 ; count<seq_get_length(&stringMsg.old_msgs) ;
      count++ )
{
    free( stringMsg.old_msgs._buffer[count] );
}
free( stringMsg.msg );
```

Then, add some code to the subscriber to handle the additional fields in the hello data type (look at the `dr_on_data_avail()` method).

Compile and re-run both the hello_pub and hello_sub applications to see the results of the additional StringMsg data.

Changing a Quality-of-Service (QoS) setting

The example Hello World program used all default Quality of Service (QoS) settings. For a complete discussion on all QoS parameters, refer to the CoreDX DDS Programmer's Guide, Chapter 12). In addition, section 7.1.3 of the DDS Specification from OMG provides a good overview to the different QoS settings.

The default History QoS policy for DataReaders is KEEP_LAST with a depth of 1. With this QoS setting, the DataReader will only keep the most recent single data **sample** for each instance. To change this policy to keep the 5 most recent samples for each instance, modify the hello_sub.c code that creates a DataReader, as follows:

hello_sub.c

```
/* In the on_data_available method, modify the read() call
 * to get all samples, instead of just those samples that
 * have not yet been read.
 */
retval = StringMsgDataReader_read( dr, &samples, &samples_info,
                                   DDS_LENGTH_UNLIMITED,
                                   DDS_ANY_SAMPLE_STATE,
                                   DDS_ANY_VIEW_STATE,
                                   DDS_ALIVE_INSTANCE_STATE );


/* In main(), where we create the data reader,
 * create with a customized quality of service
 */

/* Additional variables required */
DDS_ReturnCode_t     retval;
DDS_DataReaderQos    dr_qos;

/* create a DDS_DataReader with a listener */
/* first, create the customized QoS, based on the default */
retval = DDS_Subscriber_get_default_datareader_qos(subscriber,
                                                   &dr_qos );
if ( retval != DDS_RETCODE_OK )
{
    printf("Error getting default dr qos\n");
    return -1;
}
dr_qos.history.depth = 5;
/* Use the customized QoS to create the DDS_DataReader */
dr = DDS_Subscriber_create_datareader( subscriber,
                       DDS_Topic_TopicDescription(topic),
                       &dr_qos,
                       &drListener,
                       DDS_DATA_AVAILABLE_STATUS);
```

A Few Notes About DDS Keys

DDS provides a facility for user defined data types to identify one or more fields as a "key". By specifying a key, the published data will be categorized into instances. An instance refers to a collection of samples, where each sample has an identical key value. For example, consider a data type that has a field 'x' identified as the key and the type of 'x' is a long integer. All samples with x=5, belong to the same instance. A DataWriter can publish several samples with x=5 and they all refer to the same instance. The DDS middleware can be configured, via the History QoS, to keep all samples of an instance, or only a certain number of samples.

You specify the key for your data type by inserting some additional information into the DDL file. This provides an indication to the DDL compiler that it should generate additional code to correctly handle the key information.

For example:

Hello_key.ddl

```
#ifdef DDS_IDL
#define DDS_KEY __dds_key
#else
#define DDS_KEY
#endif

struct StringMsg
{
  DDS_KEY long    id;
          string  msg;
};
```

This example adds a key called 'id'. The #ifdef block at the beginning simply removes the "DDS_KEY" symbol if the file is not being processed by the CoreDX DDS DDL compiler. This ensures that **ddl** files maintain compatibility with standard IDL syntax.

Several fields can be marked with the DDS_KEY modifier. In this case, all 'key' fields are used together as the key for the data type.

A few notes about the DDL Compiler (coredx_ddl)

The coredx_ddl compiler handles a few command line arguments. The following briefly describes the commonly used command line options and arguments. (The '-h' argument can be used to list all the command line arguments accepted by the coredx_ddl compiler.)

-f <filename>	Specifies the DDL file to compile. This is a required argument.
-l <language>	Specifies the language to generate: 'c' for C code, 'cpp' for C++ code. The default if not specified is 'c'.
-d <output directory>	Specifies the output directory where the generated source code should be placed. By default, the files are placed in the current working directory.
-D <preprocessor symbol>	This option is used to specify preprocessor defines.
-I <include path>	This option provides a path that will be searched to satisfy '#include' directives found in the DDL file(s).

Changes from Previous Release

For current release notes, visit the Twin Oaks Computing website at:
http://www.twinoakscomputing.com/documents/CoreDX_DDS_release_notes.txt

Contact Information

Have a question? Don't hesitate to contact us by any means convenient for you:

Web Site: http://www.twinoakscomputing.com

Support:

Email: support@twinoakscomputing.com

Phone: 720.733.7906

Twitter: TOC_CoreDX

Online Forum: http://www.twinoakscomputing.com/forum

Primary Sales Office:

Email: sales@twinoakscomputing.com

Phone: 720.733.7906

EMEA Sales Office:

Email: emea.sales@twinoakscomputing.com

Phone: +33 (0)9 62 23 72 20

About Twin Oaks Computing

With corporate headquarters located in Castle Rock, Colorado, USA, Twin Oaks Computing is a company dedicated to developing and delivering quality software solutions. We leverage our technical experience and abilities to provide innovative and useful services in the domain of data communications. Founded in 2005, Twin Oaks Computing, Inc delivered the first version of CoreDX DDS in 2008. The next two years saw deliveries to over 100 customers around the world. We continue to provide world class support to these customers while ever expanding.

Contact

Twin Oaks Computing, Inc.
(720) 733-7906
+33 (0)9 62 23 72 20
755 Maleta Lane
Suite 203
Castle Rock, CO. 80108
www.twinoakscomputing.com